Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

Bakken, Glimsdal, and Tvinnereim

Heuristic Methods for a Periodic Multi-Trip Vehicle Routing Problem with Time Windows in the Food Distribution Industry

Master's thesis in Industrial Economics and Technology Management

Supervisor: Magnus Stålhane

June 2020

NTNU
Norwegian University of Science and Technology

Bakken, Glimsdal, and Tvinnereim

Heuristic Methods for a Periodic Multi-Trip Vehicle Routing Problem with Time Windows in the Food Distribution Industry

Master's thesis in Industrial Economics and Technology Management
Supervisor: Magnus Stålhane
June 2020

Norwegian University of Science and Technology
Faculty of Economics and Management
Dept. of Industrial Economics and Technology Management

Preface

This is the thesis for our master's degree at the department of Managerial Economics and Operations Research at Norwegian University of Science and Technology (NTNU) during the spring 2020. The thesis is carried out in collaboration with *ASKO Norge AS*. The thesis extends the work in Bakken et al. (2019), and propose heuristic solution methods to solve the periodic multi-trip vehicle routing problem with time windows (PMTVRPTW), incompatible goods, and a heterogeneous fleet of vehicles. In particular, we develop and study four heuristic methods. The methods are compared and evaluated compared as decision support tools in vehicle-driven distribution systems.

We would like to thank our supervisor, Associate Professor Magnus Stålhane (NTNU, Department of Industrial Economics and Technology Management), for his guidance and support. Appreciation also goes to Øyvind Gravdal and Olav Løfshus - both ASKO employees representing our industrial partner - that have provided industry insight and research guidance.

Abstract

The periodic multi-trip vehicle routing problem with time windows (PMTVRPTW), incompatible commodities, and a heterogeneous fleet arise when ASKO, a leading food-service distributor in Norway, schedule their weekly operations. In advance of a planning period, customers request a set of different commodities. Commodities are either dividable across planning periods, or restricted to be delivered in one single period. The objective is to minimize costs related to vehicle usage, i.e. driving time and fixed usage costs, while reducing the amount of overtime incurred at the warehouse. This particular combination of problem extensions is poorly covered in literature, and no solution methods are proposed to solve this VRP class. A problem-solution both assigns sequences of customers to vehicles in each planning period, and allocates commodities to each vehicle. Today, ASKO schedules routes and allocates commodities sequentially. This thesis shows that a simultaneous optimization approach can provide decision support for real-size instances within a practical amount of time.

A mathematical model of the PMTVRPTW is presented. As exact methods are unable to solve the problem for real-size instances, four different heuristic methods are developed. Inspired by the work in recent literature (e.g Vidal et al., 2012, Cattaruzza et al., 2016a), we first propose a hybrid genetic algorithm (HGA) for the PMTVRTW. The multi-periodic HGA (MPHGA) arises when the HGA is decomposed to solve the problem for each planning period separately. Third, we propose a swarm-inspired multi-periodic artificial bee colony (MPABC) algorithm which adopts the decomposition structure. Finally, the combinatorial journey-generating model (CJGM) is presented, which is a matheuristic combining partial solutions generated by the MPABC and MPHGA, with an exact method.

A computational study is conducted to evaluate and compare the proposed heuristics. Test instances are generated from real-life data provided by ASKO. All heuristics show minor deviations from the objectives obtained by an exact method on small-sized instances. Results on real-size instances (i.e. up to 115 customers) show that the CJGM is the best performing method in terms of ability to find quality solutions for all instance sizes, with a low coefficient of variation. The MPHGA also scales well to real-size instances, but has on average a larger coefficient of variation than the CJGM. The MPABC suffers from an inefficient local search operator for larger instances. The HGA reports a large average deviation from solutions obtained by the CJGM across all instance sizes. The contribution of incorporating an exact method which combines partial solutions obtained by the heuristic methods in the CJGM is analyzed, and shown to improve solution quality on all customer sizes, particularly for large instances. Thus, the CJGM is expected to serve as decision support in dynamic planning procedures, when solutions for real-size instances must be obtained rapidly without the need for complex calibration procedures.

Sammendrag

Det periodiske multi-trip vehicle routing problemet med tidsvinduer (PMTVRPTW), inkompatible varer, og en heterogen kjøretøysflåte oppstår når ASKO, det ledende selskapet for dagligvaredistribusjon i Norge, planlegger ukentlige kjøreruter. I forkant av hver planleggingsperiode bestiller kunder en mengde forskjellige varer. Hver varemengde må enten leveres i sin helhet på en leveranse, eller deles i flere leverenser på flere dager. En løsning på problemet tildeler kunder en rekkefølge i en *rundtur* som gjennomføres av et kjøretøy, for hver dag i planleggingshorisonten. I tillegg allokeres varer til hver rundtur. I dag optimerer ASKO ruteplanlegging og vareallokering sekvensielt. I denne oppgaven forsøkes det å optimere rutene og vareallokeringen simultant. Objektivet er å minimere kostnader ved bruk av kjøretøy, bestående av kjørekostnader og en fast kostnad for bruk, samtidig som man minimerer antall overtidstimer på varelageret. Denne kombinasjonen av problemutvidelser er sjelden i litteraturen, og ingen løsningsmetoder er foreslått.

En matematisk modell for ASKOs PMTVRPTW er presentert i denne avhandlingen. Fordi eksakte metoder ikke er i stand til å løse problemet for reelle kundestørrelser, er fire forskjellige heurisitiske metoder foreslått. Først presenterer vi en hybrid genetisk algoritme (HGA), inspirert av tidligere arbeid fra blant annet Vidal et al. (2012) og Cattaruzza et al. (2016a). Ved å dekomponerer problemet på periodenivå, kan hver enkelt periode løses separat med en multiperiodisk HGA (MPHGA). Videre presenterer vi en svorminspirert multiperiodisk artificial bee colony (MPABC) algoritme, som også benytter seg av en periodisk dekomponering. Til slutt blir en kombinatorisk rutegenereringsmodell (CJGM) presentert, som kombinerer deler av løsninger funnet av MPHGA og MPABC i en eksakt løsningsmetode.

Et beregningsstudie er gjennomført for å evaluere de foreslåtte heurestikkene. Det har blitt generert testinstanser basert på reel data fra ASKO. Alle heurestikkene viser lite avvik fra objektverdiene funnet av en eksakt modell på instanser med få kunder. CJGMen rapporterer beste resultater for alle instanser, også de av reell størrelse (med opptil 115 kunder). I tillegg er CJGMen den mest stabile løsningsmetoden, med en lavest variasjonskoeffisient. Effekten av å løse en eksakt metode ved å kombinere løsninger funnet av heurestikker er analysert, og det viser seg at den klarer å forbedre løsninger for alle instansstørrelser, hovedsakelig basert på løsninger funnet fra MPHGAen. For kundestørrelser opp til 50 kunder klarer MPABCen å finne løsninger av høy kvalitet, men lider av en ineffektiv lokalsøksoperator for større instanser. HGAen avviker mye fra løsningene funnet av CJGMen for alle kundestørrelser. Generelt viser det seg at en heuristisk fremgangsmåte som anvender en periodisk dekomponering i kombinasjon med en eksakt løsningsmetode gir kvalitetsløsninger innen rimelig kjøretid, og skalerer bedre til større instanser.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Abbreviations

Table 1: Table of abbreviations used in this thesis.

Abbrevation	Definition	Abbrevation	Definition
ABC	Artificial Bee Colony	MPABC	Multi-Periodic Artificial Bee Colony
ACO	Ant Colony Optimization	MPHGA	Multi-Periodic Hybrid Genetic Algorithm
AFM	Arc-Flow Model	MTVRP	Multi-Trip Vehicle Routing Problem
ALNS	Adaptive Large Neighbourhood Search	ODC	Order Distribution Chromosome
BPU	Bee Position Update	OD-MIP	Order Distribution Mixed Integer Program
CJGM	Combinatoral Journey-Generating Model	PABC	Periodic Artificial Bee Colony
CV	Coefficient of Variation	PHGA	Periodic Hybrid Genetic Algorithm
EC	Evolutionary Computation	PIX	Giant Tour Crossover
GA	Genetic Algorithm	PMTVRPTW	Periodic Multi-Trip Vehicle Routing Problem with Time Windows
GTC	Giant Tour Chromosome	PSO	Particle Swarm Optimization
HGA	Hybrid Genetic Algorithm	PVRP	Periodic Vehicle Routing Problem
HGSADC	Hybrid Genetic Search with Adaptive Diversity Control	SI	Swarm-Intelligence
IRP	Inventory Routing Problem	TS	Taboo Search
JBM	Journey-Based Model	TSP	Traveling Salesman Problem
LES	Local Enhancement Scheme	VNS	Variable Neighbourhood Search
LNS	Large Neighbourhood Search	VRPTW	Vehicle Routing Problem with Time Windows
MIP	Mixed Integer Program		

Chapter 1

Introduction

The elongated and narrow Norwegian landscape makes nationwide distribution networks challenging to operate efficiently. Persistent delivery services are threatened by widespread demand and regional climatic varieties. Distribution of food and beverages is particularly demanding, being fast-moving consumer goods required with a high frequency. Accounting for 60.5% of the Norwegian food and beverage industry, the grocery market is experiencing changes in the competitive environment by emergence of new operators and sales channels. In 2019, online stores, border shops, and specialized stores have exceeded the growth of the traditional grocery market (Norgesgruppen AS., 2019). In order to attract customers and remain competitive, grocery stores are expanding the variety of products they offer to their consumers. Consequentially, food and beverage distributors must deliver a wider range of products through their transportation network. To remain profitable while adapting to market changes, efficiency in planning and route scheduling is significant.

This thesis is written in collaboration with ASKO Norway AS. By operating warehouses which serve grocery retailers spread across the entire country, ASKO is the leading foodservice distributor in Norway. ASKO is the wholesaler branch of Norgesgruppen, which is the largest company within the Norwegian grocery retail market. Norgesgruppen consists of 1800 grocery stores which belong to various grocery chains. As for 2019, their stores are located in 88% of the Norwegian municipalities (Norgesgruppen AS., 2019).

ASKO is in control of a fleet of trucks used to connect one central warehouse to 13 regional warehouses and their corresponding customers. With more than 600 vehicles of various types in use every day, ASKO is one of the largest transportation companies in Norway (ASKO Norge AS). Their customers are essentially all the grocery store chains

of Norgesgruppen, e.g. Kiwi, Meny, Spar, and Joker. Also, ASKO supplies Norwegian convenience stores and catering sectors, e.g. hotels, canteens, gas stations, and kiosks. In subsequent parts of this thesis, grocery stores and trucks are referred to as *customers* and *vehicles*, respectively.

ASKO schedules deliveries ahead of a given planning horizon, which is composed of multiple days, denoted as *periods*. In advance of the planning horizon, each customer requests a set of *orders* to fulfil the demand for all types of goods. In the set of orders, each order is a request for a specific *commodity*. A commodity is defined as a group of goods with common properties. Goods which belong to the same commodity are located in the same area at the customer. Such an area is referred to as a *zone*, and each customer is uniquely divided into multiple zones. With this system, each zone maintains and displays goods of one commodity type to the store visitors. The number of zones and corresponding commodities for each customer varies, depending on the size of the store.

A commodity can be of two types: non-dividable or dividable. The former corresponds to dry-goods, while the latter are frozen goods, refrigerated goods, and fruits and vegetables. Dividable commodities can be split over multiple deliveries. In contrast, non-dividable commodities must be delivered in one bulk, i.e. transported to the requesting customer by the same vehicle in one period during the planning horizon. The rationale behind forcing some commodities to be non-dividable is to simplify the internal distribution of goods after deliveries at each customer. By receiving goods which belong to only a limited zone in the store, less workforce is needed.

Over the past decade, ASKO has experienced a significant increase in the amount of volume they distribute to their customers. Their annual turnover has almost doubled during this period, where operating income in 2019 amounted to approximately 90 billion NOK. The customers of ASKO now receive 80% of the volume sold in their stores from ASKO rather than from other external suppliers, which corresponds to a 20-30% growth. Increased volume is in part caused by the growing number of different products being offered in the grocery stores. Consequentially, a wider product range must be delivered on the same transport vehicle. Also, ASKO's regional warehouses are exposed to an increased flow of goods, and the task of vehicle routing and scheduling in order to serve all customers is growing in complexity. This complexity is associated with a higher cost level. In particular, costs accrue when an increased amount of different products are assembled at each warehouses before packed orders are loaded on to vehicles and delivered to customers. Also, an increase in distributed volume might cause a need for extra vehicles, which incurs additional costs for ASKO. If a vehicle is used in the planning period, it is assigned one or several *trips*. Each trip starts and ends in the same warehouse, as shown in Figure 1.1. A set of consecutive trips in the same period for a given vehicle type is referred to as a *journey*, illustrated in Figure 1.2.

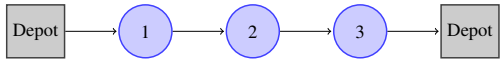


Figure 1.1: A trip is a consecutive sequence of customers visited by a specific vehicle type in a specific period.

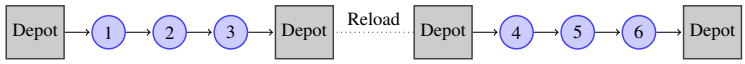


Figure 1.2: A journey consists of a set of trips which can be assigned to a specific vehicle type in a specific period.

In addition to the costs associated with vehicle usage, warehouse assembling is a labour-intensive activity. Overtime is incurred due to deviations between tactical planning and demands. Overall, cost reduction act as an incentive for suppliers to invest in improved route schedule optimization.

General routing problems are commonly modelled as vehicle routing problems (VRP) in literature. Ever since Dantzig and Ramser (1959) introduced the VRP, it has been a ubiquitous topic in operations research, logistics, communications, transportation, distribution, and manufacturing (Elshaer and Awad, 2020). For several companies that are engaged in the collection of goods or people, the VRP has been, and still is, of paramount importance. There are also examples in the literature of real-world routing problems in the food distribution industry which have been modelled as VRPs, e.g. Vidal et al. (2012) and Cattaruzza et al. (2014a).

To broaden the usage area of the VRP in practical applications, various problem extensions are considered in literature. As an example, the problem setting faced by ASKO allows vehicles to conduct multiple trips each day, as opposed to the classical VRP. The resulting multi-trip vehicle routing problem (MTVRP) has, according to Paradiso et al. (2020), been poorly covered in literature until recent years. However, research attention is encouraged by the possibility of obtaining improved city logistics in problems where vehicles are restricted to shorter driving distances and reduced capacities (Cattaruzza et al., 2016a). By modelling such routing problems as MTVRPs, better exploitation of the entire planning horizon, and improved utilization of the fleet can be obtained.

As new needs and usage areas frequently appear in real-life applications, the VRP remains an interesting and challenging field of research. Crainic (2008) emphasize that problem settings are more often subject to limited time available to make decisions. This requires solution methods which are computationally efficient, while maintaining solution quality. Also, the need for simpler, but robust and efficient methods arise, as they provide added flexibility in practical use without the need for complex calibration procedures. For these reasons, among others, heuristics have become the methodologies of choice in recent

literature on VRP solution methods (Elshaer and Awad, 2020).

In Bakken et al. (2019), we proposed three exact methods to solve the problem studied in this thesis. They struggled to find solutions for instances with more than 10 customers. The purpose of this thesis is to develop and study different heuristic solution approaches in order to solve the periodic multi-trip VRP with time windows (PMTVRPTW), incompatible commodities, and a heterogeneous fleet of vehicles, for real-sized instances.

The problem arises in tactical and operational route scheduling at ASKO, and our work aims at serving as decision support in their planning operations. As for today, journey schedules are optimized and determined before, i.e. independent of, the decision of which commodities and their associated quantities the vehicles should transport on each journey. However, compatibility dependencies between commodities can lead to poor utilization of the fleet with this sequential planning scheme. A measure of journey quality cannot be determined without knowing the optimal way of allocating quantities to each journey. On the contrary, the best order allocation cannot be found without having the optimal journey scheme. This thesis is therefore dedicated to investigate methods which simultaneously optimize journey scheduling and order allocation.

Solution methods are developed and studied as follows. First, we implement the problem as a mixed integer program based on prior work (Bakken et al., 2019) to obtain exact solutions on small benchmark instances. All instances are generated based on real-life data, as there exist no benchmarks used in literature for the PMTVRPTW with incompatible commodities and a heterogeneous fleet. Then, to solve the problem for real-size instances, we propose four different heuristic solution methods: two population-based genetic algorithms (GA), one swarm-inspired artificial bee colony (ABC) algorithm, and one hybrid matheuristic which combines GAs, ABCs, and an exact solution method to efficiently solve for instances of a practical size. To the extent of our knowledge, solution methods for this particular VRP variant have not been proposed in previous literature. However, we know that heuristic solution approaches to solve similar VRPs have performed superior to exact methods in terms of solution quality for real-sized instances (Cattaruzza et al., 2014a).

The outline of this thesis is presented in the following. In Chapter 2, we provide an overview of relevant literature within the field of operations research and vehicle routing. A detailed problem description is given in Chapter 3, and a mathematical formulation of the problem is presented in Chapter 4. The next four chapters are dedicated to describe the proposed solution methods. Chapter 5 presents a heuristic based on the hybrid genetic search framework proposed by Vidal et al. (2012). This heuristic is extended with a periodic decomposition in Chapter 6. In Chapter 7, we describe an artificial bee colony algorithm based on the concept of swarm-intelligence. Finally, a matheuristic is described in Chapter 8, which combines solutions obtained by previously described heuristics us-

ing an exact solver. In Chapter 9, test instances are described. A computational study is conducted in Chapter 10 in order to evaluate and compare the behaviour of the proposed methods. Chapter 11 concludes the thesis, whereas Chapter 12 presents suggestions for future work.

Chapter 2

Literature Review

In this chapter, we present a brief overview of relevant literature within the field of operations research and vehicle routing. Section 2.1 introduces the vehicle routing problem (VRP) with extensions relevant for the problem studied in this thesis, i.e. time windows, multiple periods, and multiple trips. In Section 2.2, we review metaheuristic solution methods proposed to solve relevant VRPs in previous literature. Finally, our contribution to the field is summarized in Section 2.3.

2.1 The Vehicle Routing Problem and Relevant Extensions

The VRP was first introduced as the ”Truck Dispatching Problem”, which is a generalization of the traveling salesman problem (TSP), by Dantzig and Ramser in 1959. Given set of cities to visit, the TSP searches for the shortest route that starts and ends in the same city. The VRP is an extension of the TSP, which has been subject to thorough research in subsequent years. In the classical VRP, cities are considered to be customers with given quantity demands, and the following assumptions are made: there is a single depot which a set of vehicles with limited capacities must depart from and return to, and each customer must be served exactly once by a vehicle. As opposed to the TSP, the need for multiple vehicles in the VRP arise when customers request certain quantities which must be served by vehicles with limited capacities. Since the TSP is NP-hard, all extensions of it are also NP-hard, including the VRP (Desaulniers et al., 2014).

In order to study and develop methods to solve the VRP in real-world applications,

various problem extensions have been proposed in the literature. For a thorough overview of the research development, the reader is referred to Laporte (2009) and Mor and Speranza (2020). In the following, VRP extensions which are present in the problem studied in this thesis are described. Note that in subsequent parts of this thesis, a *trip* is defined as a sequence of consecutive node visits, starting and ending at the depot. A *journey* refers to a set of consecutive trips taken by the same vehicle.

Time Windows

The problem is defined as a VRP with time windows (VRPTW) whenever nodes reject visits outside a given time window. A visit must start within a given time window, but can be finished after the end of the time window. Solomon (1984) was among the first to study the VRPTW. Two ways of modelling the problem have been commonly studied in subsequent literature: the VRPTW with hard or soft constraints (Desaulniers et al., 2014). In the former, only solutions where no customer is visited outside its time window are accepted. If time windows are modelled as soft constraints, however, solutions where visits occur outside time windows are accepted, but penalized with a cost that is non-decreasing with either late or early arrival time. A mix of soft and hard constraints has also been proposed (e.g. Mouthuy et al., 2015). Several penalty functions have been considered in the literature. In example, Cordeau et al. (2001) suggest to penalize late services, while Ibaraki et al. (2008) instead penalize early services. We recommend the survey provided by El-Sherbeny (2010) for a review of exact and heuristic solution methods for the VRPTW.

Multiple Periods

In a periodic VRP (PVRP), first introduced by Beltrami and Bodin (1974), the planning horizon is composed of multiple periods where node demand can be served. Each node can be visited in all periods, but often require to be visited in a certain number of periods. Further restrictions on visits have also been proposed in the literature. Cordeau et al. (1997) suggest that customers have their own visit frequencies that must be included in the routing schedules. Gaudioso and Paletta (1992) instead propose to enforce a minimum and maximum spacing between each visit.

Solving the PVRP differs from solving one VRP for each period separately, as each node requires a set of orders or a total quantity which must be delivered during the entire planning horizon rather than in specific periods. The PVRP is thus more flexible in the process of constructing routes, as quantities can be interchanged between periods in order to improve the objective value. An overview of the development of both exact and heuristic solution methods for the PVRP is provided by Campbell and Wilson (2014).

Multiple Trips

An extension of the standard VRP arises whenever each vehicle is allowed to conduct multiple trips within a period (Cattaruzza et al., 2016b). In spite of the rather growing relevance of MTVRPs in real-life applications, the literature is still scarce relative to other VRP extensions (Cattaruzza et al., 2014a). Early MTVRP research was mostly concentrated on the modelling side of the problem, whereas recent studies have focused on developing efficient solution methods (Wassan et al., 2017). We recommend Mingozzi et al. (2013) and Cattaruzza et al. (2014a) for further reference on exact and heuristic solution methods, respectively.

Urban city logistics is among the fields where MTVRP research is emerging, as MTVRPs often arise when customer demand exceeds vehicle capacity, or when distances between customers are short (François et al., 2016). In urban cities, road restrictions often favor the use of vehicles with smaller capacities, and the customers are physically closer to each other. In practice, the duration of trips in urban cities are short relative to a regular working day, and the possibility to reload at the depot between trips is desirable. If multiple trips are prohibited in such situations, an oversized fleet and poor exploration of the search space can be costly consequences (Cattaruzza et al., 2014a).

2.2 Heuristic Solution Methods for the Vehicle Routing Problem

Several papers have discussed different solution methods, both exact and heuristic, for the VRP with the extensions presented in Section 2.1. However, exact methods are usually limited in the size of instances they are able to solve for (Cattaruzza et al., 2014a). As real-life applications of VRPs tend to be large in scale, heuristics and metaheuristics have been the methodologies of choice in recent years (Elshaer and Awad, 2020). In addition, so-called *matheuristics*, which combines exact and heuristic solution methods, have received attention in VRP research. For a recent review of literature which propose matheuristics to solve various VRPs, we recommend the survey provided by Archetti and Speranza (2014).

In order to reveal usage trends of different metaheuristics applied to VRPs, Elshaer and Awad (2020) provide a taxonomic survey of metaheuristics developed and studied in literature between 2009 and 2017. They suggest a classification scheme which divides the methods into *single-solution* and *population-based* metaheuristics. The former generally proceeds by modifying and improving a single incumbent solution, while the latter evolve a set of solutions by recombining existing ones. Elshaer and Awad (2020) found that among the 299 analyzed reviews, 386 different metaheuristics were suggested. 63.7% of

the algorithms were single-solution based, while the remaining 36.3% were population-based. In this section, we review previous research on single-solution methods (2.2.1) and population-based methods (2.2.2) applied to VRPs with relevant extensions. We finish the section with a discussion of the impact a sufficient diversity-management have on algorithmic performance in the reviewed literature (2.2.3).

2.2.1 Single-Solution Metaheuristics

Several single-solution metaheuristics have been suggested for different variants of the VRP, where tabu search (TS), variable neighbourhood search (VNS), and large neighbourhood search (LNS), in the stated order, have appeared most frequent in literature (Elshaer and Awad, 2020). Single-solution metaheuristics are in general based upon the concept of maintaining an incumbent solution, use a neighbourhood operator to define a set of similar neighbourhood solutions, and repeatedly transform the current solution into one of its neighbours.

TS-algorithms are based upon exploring the search space by moving to the best neighbour of the current solution, allowing for deterioration of the objective value. In order to avoid cycling, certain *tabu* criteria and *aspiration* criteria, in combination with a memory structure, are implemented. Solutions which fulfill the tabu criteria are prohibited from being explored. However, exploration is still allowed if the aspiration criteria are fulfilled.

Shortly after the TS-framework was formalized by Glover in 1989, methods exploiting the TS-concept were proposed in VRP-literature. In example, Gendreau et al. (1994) suggested a TS-algorithm called *Taburoute* for the VRP with capacity and route length restrictions. Taburoute proceeds by considering sequences of adjacent solutions that are obtained by repeatedly remove nodes from one route and insert them into another route in the current solution. The algorithm outperformed the current best heuristics in terms of solution quality. Taillard et al. (1996) later proposed a multi-phase algorithm to solve the MTVRP, where the first phase exploits tabu search to create initial routes. Second, VRP solutions are generated, and a bin packing heuristic is finally applied to select routes and assign them to vehicles to form a MTVRP solution. The study established a set of MTVRP instances, which have been widely used as benchmarks in subsequent literature.

Variable neighbourhood search (VNS), first proposed by Mladenović and Hansen (1997) in 1997, is another single-solution metaheuristic based upon assuming there exists a local optimum in the neighbourhood of the incumbent solution. Subsequent to the local neighbourhood search, a perturbation phase enables further solution improvement by altering the nature of the neighbourhood or its parameters. This gives rise to exploitation of multiple neighbourhood operators, thereof the term *variable*. Solution acceptance and the order of which neighbours are evaluated can be either deterministic or probabilistic. Re-

cently, Wassan et al. (2017) proposed a two-level VNS algorithm to solve the MTVRP with backhauls. The two levels, referred to as the *outer* and the *inner* layer, ensure search diversification and intensification, respectively. At each cycle of the algorithm, the outer layer utilize local search procedures to identify the local optimum in the neighbourhood of the current candidate solution. The resulting best solution is carried further to the inner level, where a variable neighbourhood decent phase is applied. Computational study reports that where CPLEX has provided optimal solutions for small and medium-sized instances, the algorithm solves to optimality at lower computational costs.

According to the survey by Elshaer and Awad (2020), large neighbourhood search (LNS) is the third most frequently applied metaheuristic to solve VRPs in recent literature. The concept was first introduced by Shaw (1998) in 1998. In LNS methods, solutions are implicitly defined by *destroy* and *repair* phases. The former breaks down the current solution with a certain element of randomness, while the latter re-builds it in a different way. Typically, a probabilistic element in the destroy phase is implemented, which enables different parts of the current solution to be explored. Note that multiple destroy-repair operators can be proposed, but only one operator is randomly selected and applied in each cycle. When applied to VRPs, new solutions are typically obtained by removing a number customers from the current solution, and later reinsert them in another location. Among others, Prescott-Gagnon et al. (2009) propose a LNS method to solve the VRPTW. The algorithm succeeds to find several new best solutions to the benchmark instances provided by Solomon (1987).

A popular extension of the LNS is the adaptive large neighbourhood search (ALNS). It differs from the LNS by not randomly selecting which destroy-repair operator to apply in each iteration, but rather assigns each operator weights that are adapted through performance feedback between iterations (Azi et al., 2014). Recently, Mancini (2016) propose an ALNS algorithm to solve the multi-depot PVRP. Computational study is conducted with instances of up to 75 customers, an reveals a stable capability of exploring a large neighbourhood with low computational costs. With an average runtime of 400s, the ALNS shows an average improvement with respect to initial solution of 55%, compared to 22% for an exact solver. François et al. (2016) is another example from VRP literature where the the ALNS framework is applied. In order to solve a multi-trip VRP, they develop two ALNS algorithms which provide best known solutions for several benchmark instances from Taillard et al. (1996).

2.2.2 Population-Based Metaheuristics

Population-based metaheuristics are based upon maintaining and improving a set of candidate solutions. In recent years, several population-based methods have become state-of the

art for different VRP extensions (Vidal et al., 2012, Vidal et al., 2013a). Population-based algorithms are particularly suitable when a diversified exploration of the search space is desired, as they can simultaneously maintain a large set of solutions (Cattaruzza et al., 2014a). In the following, we present two classes of population-based methods: *evolutionary computation* (EC) algorithms (2.2.2) and *swarm-intelligence* (SI) algorithms (2.2.2).

Evolutionary Computation Algorithms

According to Elshaer and Awad (2020), genetic algorithms (GA) represent the class of EC algorithms which have received most attention in VRP literature. GAs are inspired by the concept of natural evolution of biological organisms (Cattaruzza et al., 2014a). The class of algorithms was first introduced late in the 1950s, but studied and developed towards their present shape by Holland in 1975. The GA framework is based upon representing solutions as individuals, and evolve a set of individuals (i.e. a *population*) until a sufficient solution quality, a given time limit, or a maximum number of iterations is reached. Improved solutions are obtained by recombining individuals, referred to as a *crossover* procedure.

Vidal et al. (2012) suggest a hybrid metaheuristic for the PVRP based on the GA framework, proposing several advanced features in solution evaluation, population management, and crossover operations. Solutions are represented as a set of *chromosomes*, where a *giant tour* chromosome is a sequence of customer nodes without delimiters. This particular way of representing solutions has been efficiently applied in several GAs for standard VRPs (Vidal et al., 2013a). The resulting algorithm - a Hybrid Genetic Search with Adaptive Diversity Control (HGSADC) - performs superior to previous attempts in terms of computational efficiency and solution quality on benchmark instances from Cordeau et al. (1997).

In Vidal et al. (2013b), the HGSADC is further extended to solve for PVRPs with time windows. Their work is inspired by Nagata et al. (2010), who introduce the idea of temporarily allowing time window infeasible solutions in the population during the search. By modelling time windows as soft constraints, a wider exploration of structurally different feasible solutions is enabled. The HGSADC extension proposed by Vidal et al. (2013b) assumes that upon late arrival, time is paused until the time window is reached, and a penalization is incurred accordingly. This relaxation scheme contributes to an efficient evaluation of solution changes, where several classical neighbourhood search moves can be evaluated in amortized $O(1)$. The algorithm performs impressively in terms of solution quality and computational efficiency compared to previous approaches on benchmark instances for any combination of multi-depot, periodic, site-dependent, and duration-constrained VRPs with time windows.

In recent years, population-based methods have gained attention in the context of MTVRP literature. As one of the first authors to consider solution methods for the MTVRP, Fleischmann (1990) established the mindset of separating route creation and route assignment. This approach remained popular in MTVRP literature for several years (François et al., 2016). In recent years, heuristics which simultaneously tackle both creation and assignment of routes have been proposed. This also encouraged the emergence of population-based methods for MTVRPs.

Cattaruzza et al. (2014a) propose a population-based approach for the MTVRP which outperform previous methods in the literature with respect to average solution quality. Their study is motivated by a real-world application, where a set of supermarkets place orders that must be delivered by a fleet of vehicles within a time horizon. Orders are incompatible and must be delivered with separate vehicles. Their proposed algorithm is largely based on the HGSADC introduced by Vidal et al. (2012). To include the multi-trip aspect, a new local search operator that includes exploitation of potential improving trip swaps is proposed. In the original HGSADC, the swaps considered are limited to exploration of the order of which customers are visited inside each single trip. In addition to an improved local search operator, Cattaruzza et al. (2014a) modifies the split-procedure that constructs VRP solutions from giant tours in the original HGSADC. The new split-procedure incorporates a labeling algorithm subsequent to the creation of trips, where trips are selected and assigned to vehicles to form MTVRP solutions. In order to solve the MTVRP with time windows (MTVRPTW), Cattaruzza et al. (2014b) extend the labeling algorithm developed in Cattaruzza et al. (2014a) by incurring a penalty for time window infeasibility. The proposed method is based upon an iterated local search method. Each time a solution is created by the modified split-procedure, a local search operator is applied to possibly improve solution quality. Computational study of the proposed method shows that by enabling multiple trips as opposed to solving a single-trip VRP, the fleet size can be reduced with as much as 50%.

In order to address large-sized instances of VRPTWs with various extensions more efficiently, several decomposition approaches have been proposed in literature (Vidal et al., 2013b). Usually, problem decomposition is based on geometry (Ostertag, 2008, Bent and Van Hentenryck, 2010), problem structure (Crainic et al., 2009), or temporal aspects (Bent and Van Hentenryck, 2010). For population-based methods, Vidal et al. (2013b) propose a decomposition framework which takes advantage of the maintained pool of solutions. They develop a solution method to solve a PVRPTW, which implements a decomposition phase solve the problem as multiple *subproblems*. In the decomposition phase, subproblems accounting for problem dependencies which arise from both time window constraints and the multi-period extension, are created. To incorporate the former when creating subproblems, a trip-based geometrical decomposition is applied. As for the latter, a structural decomposition is applied by fixing orders to periods, and thus create one subproblem for

each period. Vidal et al. (2013b) conduct a sensitivity analysis to measure the impact of the decomposition phase on algorithmic performance, which confirms a significant contribution, particularly in terms of solution quality and computing efficiency for large-sized instances.

Swarm-inspired Algorithms

Swarm-inspired algorithms are in general based upon exploiting the collective, decentralized intelligent behaviour of various types of organisms, e.g. ants, bees, and birds, in order to find problem solutions. In general, SI-algorithms are based on movement and evaluation in a solution space, where a position in the space represents a solution to the given problem. SI-algorithms were first applied to continuous optimization problems, but were later adapted to handle discrete domain problems such as the job shop scheduling problem, and the VRP (Krause et al., 2013). In discrete domain problems, a solution is often represented as a position-vector in the solution space, which is updated when the candidate solution is changed. However, for discrete combinatorial problems, such position updates can lead to invalid solutions. In order to prevent invalid position updates, two different ways of representing solutions have been proposed: *direct* and *indirect* representation (Krause et al., 2013).

With a direct representation, the position vector of a solution in the solution space explicitly holds the solution. With an indirect representation, an encoding scheme must be applied in order to map the position vector from a continuous space to a discrete solution. Various encoding schemes have been proposed. The most popular schemes are the sigmoid function, which maps the continuous space into a binary vector based on the sigmoid function (Banati and Bajaj, 2011), and the random-key encoding scheme proposed by Chen et al. (2011). The random-key encoding scheme decodes the continuous position in each dimension to an integer position. As a simple example, the candidate solution vector $\vec{x}_i = (0.90, 0.35, 0.03, 0.21, 0.17)$ can be decoded as $\vec{x}_i = (5, 4, 1, 3, 2)$. Other encoding schemes are described in Krause et al. (2013).

Particle swarm optimization (PSO) is the most frequent applied SI-algorithm in recent VRP literature (Elshaer and Awad, 2020). PSO (Kennedy and Eberhart, 1995) is based upon representing candidate solutions as particle positions. Particles move around in the search space, and their positions are guided by the continuously updated local and global best known positions. The swarm is thus expected to move towards the best solutions. Zhen et al. (2020) propose a PSO algorithm, which exploits the split-algorithm in Cattaruzza et al. (2014b) to transform the sequence of customer visits of a particle based on a random key encoding scheme, into a MTVRPTW solution. Their study is motivated by a real-world application within last-mile distribution in online shopping. The suggested

algorithm is able to efficiently solve instances with up to 200 customers and 40 vehicles.

Ant colony optimization (ACO) is another SI-method which have gained recent attention in the context of real-world VRP applications (Rizzoli et al., 2007). ACOs are inspired by the behaviour of ants during their search for food. As in general for SI-algorithms, solutions are positions of ants in the search space. In each iteration, ants move towards better positions by observing pheromone trails which were deposited in the previous iteration. ACOs are commonly proposed to solve VRPs, as the movements of ants are suited to represent vehicles, letting their food sources represent nodes (Yu et al., 2009). Yu et al., 2009 propose an ACO which exploits a direct solution representation, where the amount of pheromone on the edge between two nodes indicates solution quality. Edges are selected based on a probability given by the pheromone density on the edges leaving the customer. Matos and Oliveira (2004) proposed a two-phased ACO algorithm to solve the PVRP for large-sized instances. They address the PVRP as a VRP by duplicating each customer as many times as it has placed an order in the entire planning horizon. Finally, a graph-coloring problem is solved in order to allocate trips to periods.

Artificial bee colony (ABC), first introduced by Karaboga (2005), was recently introduced in VRP literature (Iqbal et al., 2015). Even though few authors have exploited the ABC-framework to solve VRPs, it has proven to obtain quality solutions within a reasonable amount of time for several other hard discrete combinatorial problems in literature (e.g. the traveling salesperson problem (Agrawal et al., 2012), and the general assignment problem (Baykasoglu et al., 2007, Iqbal et al., 2015). The idea is to simulate the foraging behaviour of a honey bee swarm when searching for quality solutions. Solutions are represented by bee positions in the search space. The ABC analogy is based on bees exploring a meadow (search space) with flower patches (positions), where bees collect nectar (solution quality). The algorithm is composed of three phases. In the first phase, random positions in the solution space are found. In the second phase, positions are evaluated to reveal their quality. Finally, the most promising solutions are further explored in the third phase. Exploration is proceeded as a local neighbourhood search around the promising solutions in the solution space. If an area has been explored for a given number of iterations without improvement, it is abandoned.

Szeto et al. (2011) was first to apply an ABC algorithm to the standard VRP problem with limited vehicle capacities. When tested on 20 large-scale benchmark instances (Golden et al., 1998), the proposed algorithm obtained an average deviation of 2.31% from the best known result among previous suggested methods. Iqbal et al. (2015) applied the ABC to a VRP with soft time window constraints. As an extension of the standard ABC framework, they implement an additional step, denoted *global exploration*, dedicated to enhance search diversity. The step is applied whenever the neighbourhood of a current solution is explored, and is composed of two phases:

- (1) Select two different routes and one customer in each of the routes, and swap the two customers
- (2) Select a block of customers, and replace it with a random perturbation of itself

Experimental results show that the *global exploration* step reduces the run time of the method. For the entire algorithm, computational studies reveal that high quality solutions are obtained within a reasonable amount of time on benchmark instances (Solomon, 1987) when compared to previous proposed heuristics.

2.2.3 The Importance of Diversity

A common denominator in the reviewed literature is the importance of a sufficient diversity-management scheme for VRP solution algorithms to be efficient. Various approaches are proposed to obtain the desired level of diversity.

One of the suggested approaches is based upon explicitly including a measure of diversity in the objective function. In example, Vidal et al. (2012) suggest a solution evaluation mechanism which aims at providing broad access to reproduction material during the population-based evolutionary search. The fitness score of an individual, which is used whenever solutions are compared, is based upon both the cost of the solution and its contribution to population diversity. Thus, high-cost solutions can still be used to create new solutions. The proposed evaluation method is adopted by Vidal et al. (2013b), Cattaruzza et al. (2014a), and Zhen et al. (2020).

A more common approach to implement search diversity is to include an algorithmic step, subsequent to evaluation, where solutions are either accepted or rejected. In example, François et al. (2016) suggest an ALNS algorithm which enhance solution diversity whenever a candidate solution is either accepted or rejected with a given probability. Cattaruzza et al. (2014b) apply another diversity-management scheme, which is based upon a frequent use of random solution perturbations. The swarm-inspired ABC proposed by Iqbal et al. (2015) also implements random perturbations, i.e. the *global exploration* step, as en extension of the ABC-framework in order to achieve enhanced search diversity.

The importance of diversity is explicitly tested in computational studies by several authors. Experimental results in Iqbal et al. (2015) reveal that the *global exploration* step improves both solution quality and computation time. In Vidal et al. (2012), they state that the biased fitness evaluation is paramount for the suggested algorithm to obtain quality solutions on literature benchmarks. Similarly, Vidal et al. (2013b) conduct a sensitivity analysis which proves the significant contribution of diversity to the algorithmic performance. Also, they prove the importance of including time window-infeasible solu-

tions in the evolving population. Cattaruzza et al. (2014a) state that sufficient diversity-management enables local search procedures to be efficient while avoiding premature convergence, which is a common challenge in GAs.

2.3 Our Contribution

VRPs are widely studied, and various extensions have been proposed in order to model problems of real-life applicability. However, as emphasized in Vidal et al. (2012), there are significant differences in the amount of attention received by the different problem classes. Most methodological developments tend to target single-attribute VRPs, e.g. VRPs with time windows, or the traditional VRP with vehicle capacity restrictions. In this thesis, we study a periodic VRP with multiple trips and time windows (PMTVRPTW), where commodities are incompatible, and the fleet is heterogeneous. This particular combination of VRP extensions is, to the extent of our knowledge, not covered in previous research.

In order to solve the problem for instances which compare to real-life problem sizes, we propose four different heuristic solution methods. Various solution methods for VRP classes with some of the relevant problem extensions have been studied in literature, particularly in the domain of heuristic and metaheuristic approaches. Two of the proposed methods are based on the framework of genetic algorithms, whereas the third adopts the concept of swarm-intelligence. Finally, the fourth method is a matheuristic, which is a hybrid of the proposed heuristics and an exact method based on prior work (Bakken et al. (2019)).

Genetic algorithms have emerged as promising in VRP research in recent years. Of particular interest is the hybrid genetic algorithm framework proposed by Vidal et al. (2012), which yielded state-of-the-art results on different VRP classes. They suggest extending their framework to new problem classes which include other combinations of VRP extensions as a new area of research. As for algorithms inspired by swarm-intelligence, they are less explored in the VRP context, but have successfully been applied to solve other difficult discrete combinatorial problems (Iqbal et al., 2015).

In recent years, advantages in exact solution methods and hardware technology have improved the ability of mixed integer linear programming (MIP) models to be solved to optimality or close to optimality within a reasonable amount of time. Consequentially, researchers have been encouraged to develop matheuristics, i.e. heuristics which embed phases where MIP-techniques are exploited, to solve various VRPs. We propose a matheuristic which iterates between (1) solving a decomposed problem heuristically to generate a set of potentially quality solutions for each subproblem, and (2) applying a MIP to identify the optimal configuration among the set of potential partial solutions to form a

complete problem solution.

There are only a few papers in previous literature, e.g. Cattaruzza et al. (2014a) and Cattaruzza et al. (2014b), which suggest solution methods to solve VRPs with commodity incompatibilities. However, among these methods, the quantity fractions of each commodity delivered in each period are treated as fixed, rather than as decision variables. Treating quantities as decision variables is an essential characteristic of the problem studied in this thesis. Therefore, the proposed solution methods contribute to the field with different approaches to handle the problem of allocating commodity quantities to periods.

Even though the PMTVRPTW studied in this thesis is poorly covered in existing literature, the mathematical formulation and proposed solution methods can be adapted to describe and solve other problems with similar characteristics. The work in this thesis is motivated by the grocery industry in Norway, but is of general applicability for related practical problems in other industries. Based on the review of similar approaches suggested to solve other VRP classes in this section, we believe that our proposed methods will both serve as decision support for ASKO, and provide valuable insight to future VRP research.

Chapter 3

Problem Description

In this thesis, a periodic multi-trip vehicle routing problem with time windows (PMTVRPTW), incompatible commodities, and a heterogeneous fleet is studied. The one-to-many problem consists of multiple customers and one warehouse, represented by nodes and one depot, respectively. Each planning period is considered to be one day, and every customer has a set of orders that needs to be fulfilled during a planning horizon of six days, i.e. Monday to Saturday.

In a set of orders placed by a customer, each order is a quantity demand for a particular commodity. There are two types of commodities: *dividable* and *non-dividable* commodities. Orders of a non-dividable commodity must be delivered as one unit in a single period. Customers can receive at maximum one non-dividable commodity in each period. Dividable commodities can be split up and delivered in several periods during the planning horizon. However, each customer has individual quantity and frequency restrictions on deliveries of each dividable commodity. Thus, deliveries of a dividable commodity must happen with a frequency within a pre-specified interval, and the quantity delivered must be between a lower and upper bound.

A customer delivery can only be scheduled in periods where the particular customer accepts visits. Potential delivery periods are determined by each customer in advance of the planning horizon. Also, visits must be scheduled inside a customer specific time window, and the unloading at a customer takes a fixed unloading time. Vehicles which arrive in advance of a time window must wait until it opens before unloading. As long as deliveries begin inside the time window, unloading does not need to be completed within it.

The supplier is in possession of a heterogeneous fleet of vehicles which are used to

execute trips in each period. A trips is a sequence of customer visits that must start end end at the depot. Each vehicle can conduct multiple trips, i.e. a journey, during each planning period. A maximum number of trips can be completed within a period for each vehicle. The different vehicle types vary with respect to capacity, the fixed cost of usage in a period, the loading time at the depot, and the driving costs. The supplier has a limited number of vehicles available of each vehicle type during the planning horizon.

Orders are assembled and loaded into vehicles at the depot. Loading time is considered fixed, as orders are assembled prior to loading. The required amount of working hours in a period at the depot is a multiple of the total quantity delivered by the fleet in this particular period. A cost per unit of overtime is incurred in a period whenever the number of working hours exceeds a limit for this period.

The decision to make is, for each customer, in which period to schedule deliveries of which commodities by which vehicle. For the dividable commodities, the distribution of quantities to periods also needs to be determined. The objective is thus to minimize the sum of the following costs:

1. Cost of using vehicles in the planning horizon
2. Cost of the total distance traveled by the fleet
3. Cost of overtime hours incurred at the depot

In summary, the goal is to find a set journeys, and assign journeys to vehicles such that the number of vehicles used, the total distance traveled, and overtime labor costs at the depot are minimized. By the end of the planning horizon, the demand for every commodity requested by all customers must be delivered.

Chapter 4

Mathematical Model

As presented in Chapter 3, this thesis studies a periodic multi-trip vehicle routing problem with time windows (PMTVRPTW), incompatible commodities, and a heterogeneous fleet. In this chapter, a mathematical model for this problem is formulated and described.

4.1 The Arc-Flow Model

The relation between customers, depo,t and arcs in the AFM is represented as a graph $\mathcal{G}$. Let the graph $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be defined by the set of nodes $\mathcal{N}$ connected by the set of arcs $\mathcal{A}$. Depot and customers are represented as unique nodes. The nodes $i = 0$ and $i = |\mathcal{N}| + 1$ are defined as the start and end depot, respectively. The depots have identical locations. Vehicles traverse arcs between nodes to conduct visits.

There are no incompatibilities between vehicles, arcs, and nodes, i.e. the network is fully connected. To simplify notation, constraints are defined using sets of nodes even though some constraints may contain combinations of indices i and j where the corresponding variable, x_{pvrij}, does not exist, e.g. when $i = j$, $(i, j) = (|\mathcal{N}| + 1, 0)$, or $(i, j) = (0, |\mathcal{N}| + 1)$. These variables can be assumed to take the value of zero.

The mathematical model for this problem is based on the 4-index MTVRP formulation described by Cattaruzza et al. (2016a), which is the most common MTVRP formulation. The formulation is extended with an index for each period, $p \in \mathcal{P}$. In addition, an index for commodity $m \in \mathcal{M}_i$ is used for each customer $i \in \mathcal{N}$. To address the heterogeneous fleet, a set of vehicle types $\mathcal{H}$ is introduced, with vehicle type-indexed capacities Q_h, driving costs C_h^T, and fixed usage cost C_h^F. Overtime during a period is incurred whenever

the upper limit for quantity leaving the depot, Q_p^O, is reached. The cost of each unit of overtime is defined as C_p^O.

4.1.1 Definition of Sets, Indices, Parameters, and Variables

Definition of Sets:

$\mathcal{P}$	Set of periods
$\mathcal{R}$	Set of trip indices
$\mathcal{N}$	Set of customers excluding the depot
$\mathcal{N}^0$	Set of customers including the depot, $\mathcal{N}^0 = \mathcal{N} \cup \{0, \lvert\mathcal{N}\rvert + 1\}$
$\mathcal{A}$	Set of all arcs
$\mathcal{H}$	Set of vehicle types
$\mathcal{V}$	Set of vehicles
$\mathcal{V}_h$	Set of vehicles of a given vehicle type, where all vehicles $v \in \mathcal{V}_h$ belong to vehicle type $h \in \mathcal{H}$
$\mathcal{M}_i^{ND}$	Set of non-dividable commodities for customer $i \in \mathcal{N}$
$\mathcal{M}_i^D$	Set of dividable commodities for customer $i \in \mathcal{N}$
$\mathcal{M}_i$	Set of all commodities for customer $i \in \mathcal{N}$, where $\mathcal{M}_i = \mathcal{M}_i^D \cup \mathcal{M}_i^{ND}$

Definition of Indices:

d	Period, where $p \in \mathcal{P}$
r	Trip index, where $r \in \mathcal{R}$
i, j	Customer, where $i, j \in \mathcal{N}$ and $(i, j) \in \mathcal{A}$
v	Vehicle, where $v \in \mathcal{V}$
m	Commodity, where $m \in \mathcal{M}_i$
h	Vehicle type, where all $v \in \mathcal{V}_h$ have a corresponding vehicle type $h \in \mathcal{H}$

Definition of Parameters

$\underline{T}_{pi}$	Start of time window in period $p \in \mathcal{P}$ for customer $i \in \mathcal{N}$
$\overline{T}_{pi}$	End of time window in period $p \in P$ for customer $i \in \mathcal{N}$
T_{ij}	Time spent traversing the arc from customer i to customer j, $(i,j) \in \mathcal{A}$, where $i \neq j$
T^H	Maximum duration of a journey
T_i^U	Fixed unloading time for customer $i \in \mathcal{N}$
T_h^L	Loading time at depot for vehicle $h \in \mathcal{H}$
Q_h	Capacity for vehicle type $h \in \mathcal{H}$
Q_{im}^{ND}	Quantity of non-dividable commodity $m \in \mathcal{M}_i^{ND}$ ordered by customer $i \in \mathcal{N}$
Q_{im}^D	Quantity of dividable commodity $m \in \mathcal{M}_i^D$ ordered by customer $i \in \mathcal{N}$
$\overline{Q}_{im}^D$	Maximum quantity of commodity $m \in \mathcal{M}_i^D$ delivered to customer $i \in \mathcal{N}$ in period $p \in \mathcal{P}$
$\underline{Q}_{im}^D$	Minimum quantity of commodity $m \in \mathcal{M}_i^D$ delivered to customer $i \in \mathcal{N}$ in period $p \in \mathcal{P}$
Q_p^O	Upper limit on the total quantity of commodity leaving the depot before overtime is incurred in period $p \in \mathcal{P}$
C_p^O	Unit cost of overtime at the depot
C_h^T	Traveling cost for vehicle type $h \in \mathcal{H}$
C_h^F	Fixed cost of using a vehicle of type $h \in \mathcal{H}$ in one period
I_{pi}	Binary parameter, 1 if a customer $i \in \mathcal{N}$ must be visited in period $p \in \mathcal{P}$, 0 otherwise
$\overline{F}_{im}$	Maximum number of deliveries of dividable commodity $m \in \mathcal{M}_i^D$ for customer $i \in \mathcal{N}$ in the planning horizon
$\underline{F}_{im}$	Minimum number of deliveries of dividable commodity $m \in \mathcal{M}_i^D$ for customer $i \in \mathcal{N}$ in the planning horizon
$\lvert\mathcal{N}\rvert$	Number of customers
$\lvert\mathcal{R}\rvert$	Maximum number of trips a vehicle can take in one period
M_X	A large number, where the value of x differentiates different large numbers

Definition of Variables

$$x_{pvrij} = \begin{cases} 1 & \text{if vehicle } v \in \mathcal{V} \text{ on trip } r \in \mathcal{R} \text{ traverse arc } (i,j) \in \mathcal{A} \\ & \text{in period } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

$$y_{pvri} = \begin{cases} 1 & \text{if vehicle } v \in V \text{ on trip } r \in \mathcal{R} \text{ visits customer } i \in \mathcal{N} \\ & \text{in period } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

$$z_{pvr} = \begin{cases} 1 & \text{if vehicle } v \in \mathcal{V} \text{ uses trip } r \in \mathcal{R} \text{ in period } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

$$u_{pim} = \begin{cases} 1 & \text{if commodity } m \in \mathcal{M}_i \text{ is delivered to customer } i \in \mathcal{N} \\ & \text{in period } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

q_{pvrim} Quantity of commodity $m \in \mathcal{M}_i$ delivered to customer $i \in \mathcal{N}$ with vehicle $v \in \mathcal{V}$ on trip $r \in \mathcal{R}$ in period $p \in \mathcal{P}$

t_{pvri} Time of visit if node $i \in \mathcal{N}^0$ by vehicle $v \in \mathcal{V}$ on trip $r \in \mathcal{R}$ in period $p \in \mathcal{P}$

q_p^O Quantity of commodity distributed using overtime in period $p \in \mathcal{P}$

4.1.2 Mathematical Formulation of the Ark-Flow Model

Objective Function:

$$Minimize \sum_{p\in\mathcal{P}} \sum_{h\in\mathcal{H}} \sum_{v\in\mathcal{V}_h} \sum_{r\in\mathcal{R}} \sum_{(i,j)\in\mathcal{A}} C_h^T T_{ij} x_{pvrij} + \sum_{p\in\mathcal{P}} \sum_{h\in\mathcal{H}} \sum_{v\in\mathcal{V}_h} C_h^F z_{pv(1)} + \sum_{p\in\mathcal{P}} C_p^O q_p^O \tag{4.1}$$

The objective (4.1) minimizes total costs, which is composed of three terms. The first term represents the cost of traversing an arc between two nodes, where time used is converted to a cost which depends on the vehicle type used. The second term is the fixed cost of using a vehicle in a period, where a vehicle is considered used if it is assigned at least one trip in the period. Third, a cost is incurred per unit of overtime at the depot in each period.

In the following, constraints are presented and explained in groups based on their ap-

plication.

Time Constraints:

$$\underline{T}_{pi} \leqslant t_{pvri} \leqslant \overline{T}_{pi}, \quad i \in \mathcal{N}, h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, p \in \mathcal{P} \tag{4.2}$$

$$\begin{aligned} t^{pvri} + T_{ij} + T_i^U - t_{pvrj} \leqslant M_1(1 - x_{pvrij}), \\ h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, i \in \mathcal{N}, j \in \mathcal{N}^0 \setminus \{0\}, p \in \mathcal{P} \end{aligned} \tag{4.3}$$

$$\begin{aligned} t_0^{pvri} + T_{0j} - t_{pvrj} \leqslant M_2(1 - x_{pvr0j}), \\ h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, j \in \mathcal{N}^0 \setminus \{0\}, p \in \mathcal{P} \end{aligned} \tag{4.4}$$

$$t_{pvr(|\mathcal{N}|+1)} + T_h^L z_{pv(r+1)} \quad \leqslant t_{pv(r+1)(0)}, \qquad r \in \mathcal{R} \setminus \{|\mathcal{R}|\}, h \in \mathcal{H}, v \in \mathcal{V}_h, p \in \mathcal{P} \tag{4.5}$$

$$t_{pvr(|\mathcal{N}|+1)} \leqslant T^H, \quad h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, p \in \mathcal{P} \tag{4.6}$$

Constraints (4.2) ensure that all visiting times are within the time window for each customer. Constraints (4.3) and (4.4) handle the time spent traversing an arc between nodes. The difference between (4.3) and (4.4) is that unloading time should not be incurred when traveling from the depot. These constraints are non-linear, and a linear formulation will be presented in Section 4.1.3. Constraints (4.5) ensure that the vehicle depending loading time are included in the time dependencies between two trips. Loading time is not incurred if vehicle v does not use the next trip as a part of its journey. Constraints (4.6) enforce the duration of a journey to be less than the maximum duration of a journey T^H.

Capacity Constraints:

$$\sum_{i \in \mathcal{N}} \sum_{m \in \mathcal{M}_i} q_{pvrim} \leqslant Q_h, h \in \mathcal{H}, v \in \mathcal{V}_h, \quad r \in \mathcal{R}, p \in \mathcal{P} \tag{4.7}$$

$$\sum_{h \in \mathcal{H}} \sum_{v \in \mathcal{V}_h} \sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{N}} \sum_{m \in \mathcal{M}_i} q_{pvrim} \leqslant Q_p^O + q_p^O, \quad p \in \mathcal{P} \tag{4.8}$$

Constraints (4.7) prohibit vehicles from transporting a quantity of commodities that exceeds the capacity of the vehicle Q_h. Constraints (4.8) ensure that overtime is incurred at the depot if the amount of commodities transported during a period is above Q_p^O.

Flow Constraints:

$$\sum_{j \in \mathcal{N}^0 \setminus \{0\}} x_{pvr0j} = z_{pvr}, \quad r \in \mathcal{R}, h \in \mathcal{H}, v \in \mathcal{V}_h, p \in \mathcal{P} \tag{4.9}$$

$$\sum_{i \in \mathcal{N}^0 \setminus \{|\mathcal{N}|+1\}} x_{pvri(|\mathcal{N}|+1)} = z_{pvr}, h \in \mathcal{H}, v \in \mathcal{V}_h, \quad r \in \mathcal{R}, p \in \mathcal{P} \tag{4.10}$$

$$\sum_{j \in \mathcal{N}^0 \setminus \{0\}} x_{pvrij} = y_{vrpj}, \quad j \in \mathcal{N}, h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R} p \in \mathcal{P} \tag{4.11}$$

$$\sum_{i \in \mathcal{N}^0 \setminus \{|\mathcal{N}|+1\}} x_{ij}^{vrp} = y_{pvri}, \quad i \in \mathcal{N}, h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, p \in \mathcal{P} \tag{4.12}$$

$$\sum_{h \in \mathcal{H}} \sum_{v \in \mathcal{V}_h} \sum_{r \in \mathcal{R}} y_{pvri} = I_{pi}, \quad i \in \mathcal{N}, p \in \mathcal{P} \tag{4.13}$$

$$z_{pvr} \geqslant z_{pv(r+1)}, \quad r \in \mathcal{R} \setminus \{|\mathcal{R}|\}, h \in \mathcal{H}, v \in \mathcal{V}_h, p \in \mathcal{P} \tag{4.14}$$

$$\sum_{i \in \mathcal{N}} y_{pvri} \leqslant M_3 z_{pvr}, \quad r \in \mathcal{R}, h \in \mathcal{H}, v \in \mathcal{V}_h, p \in \mathcal{P} \tag{4.15}$$

$$\sum_{m \in \mathcal{M}_i} q_{pvrim} \leqslant M_4 y_{pvri}, \quad i \in \mathcal{N}, r \in \mathcal{R}, h \in \mathcal{H}, v \in \mathcal{V}_h, p \in \mathcal{P} \tag{4.16}$$

Constraints (4.9) and (4.10) ensure that each trip must start and end in the depot. Constraints (4.11) and (4.12) handle the flow into and out of all customer nodes. Constraints (4.13) ensure that a customer is visited in a period p if required, and prohibit visits in other periods. Constraints (4.14) state that vehicle v cannot use trip r if trip $r+1$ unused. Constraints (4.15) prohibit a vehicle from visiting customers in trip r if the trip is unused. Alternatively, it could be formulated by iterating over all customers outside of the con-

straints. However, preliminary studies show that this has a marginal effect on the solution time. Constraints (4.16) ensure that no commodities are delivered to a customer that has not been visited.

Quantity Constraints:

$$\sum_{h\in\mathcal{H}}\sum_{v\in\mathcal{V}_h}\sum_{r\in\mathcal{R}} q_{pvrmi} = Q^{ND}_{mi}u_{pmi}, \quad p\in\mathcal{P}, i\in\mathcal{N}, m\in\mathcal{M}^{ND}_i \tag{4.17}$$

$$\underline{Q}^{D}_{im}u_{pim} \leqslant \sum_{h\in\mathcal{H}}\sum_{v\in\mathcal{V}_h}\sum_{r\in\mathcal{R}} q_{pvrim} \leqslant \overline{Q}^{D}_{im}u_{pim}, \quad p\in\mathcal{P}, i\in\mathcal{N}, m\in\mathcal{M}^{D}_i \tag{4.18}$$

$$\sum_{p\in\mathcal{P}}\sum_{h\in\mathcal{H}}\sum_{v\in\mathcal{V}_h}\sum_{r\in\mathcal{R}} q_{pvrim} = Q^{D}_{im}, \quad i\in\mathcal{N}, m\in\mathcal{M}_i \tag{4.19}$$

$$\sum_{m\in\mathcal{M}^{ND}_i} u_{pim} \leqslant I_{pi}, \quad p\in\mathcal{P}, i\in\mathcal{N} \tag{4.20}$$

$$\sum_{p\in\mathcal{P}} u_{pim} = 1, \quad i\in\mathcal{N}, m\in\mathcal{M}^{ND}_i \tag{4.21}$$

$$\sum_{p\in\mathcal{P}} u_{pim} \geqslant \underline{F}_{im}, \quad i\in\mathcal{N}, m\in\mathcal{M}^{D}_i \tag{4.22}$$

$$\sum_{p\in\mathcal{P}} u_{pim} \leqslant \overline{F}_{im}, \quad i\in\mathcal{N}, m\in\mathcal{M}^{D}_i \tag{4.23}$$

Constraints (4.17) ensure that if a non-dividable commodity is delivered to customer i in period p, the total quantity ordered of this commodity is delivered. Constraints (4.18) state that if a dividable commodity is delivered in one period p, then the quantity of that commodity must be within upper and lower bounds. Constraints (4.19) ensure that customers receive all their requested orders during the planning horizon. Constraints (4.20) allow at most one non-dividable commodity is to be delivered to customer i in period p. Constraints (4.21) state that all non-dividable commodities are delivered within the planning horizon. Finally, Constraints (4.22) and (4.23) provide upper and lower bounds for how many times a dividable commodity can be delivered during the planning horizon.

Constraints on Variables:

$$x_{pvrij} \in \{0,1\}, \qquad p \in \mathcal{P}, h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, (i.j) \in \mathcal{A} \tag{4.24}$$

$$y_{pvri} \in \{0,1\}, \qquad p \in \mathcal{P}h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, i \in \mathcal{N} \tag{4.25}$$

$$z_{pvr} \in \{0,1\}, \qquad p \in \mathcal{P}, h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R} \tag{4.26}$$

$$u_{pim} \in \{0,1\}, \qquad p \in \mathcal{P}, i \in \mathcal{N}, m \in \mathcal{M}_i \tag{4.27}$$

$$q_{pvrim} \geqslant 0, \quad p \in \mathcal{P}, h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, i \in \mathcal{N}, m \in \mathcal{M}_i \tag{4.28}$$

$$q_p^O \geqslant 0, \qquad p \in \mathcal{P} \tag{4.29}$$

$$t_{pvri} \geqslant 0, \qquad p \in \mathcal{P}, h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, i \in \mathcal{N}^0 \tag{4.30}$$

4.1.3 Improvements to the Arc-Flow Model

A linearization of constraints (4.3) and (4.4) to obtain constraints (4.31) and (4.32) is done according to Toth and Vigo (2002):

$$t_{pvri} + T_{ij} + T_i^U - t_{vrpj} \leqslant M_1(1 - x_{pvrij}), \qquad h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, i \in \mathcal{N}, j \in \mathcal{N}^0 \setminus \{0\}, p \in \mathcal{P} \tag{4.31}$$

$$t_{pvr(0)} + T_{0j} - t_{pvrj} \leqslant M_2(1 - x_{pvr(0)j}), \qquad h \in \mathcal{H}, v \in \mathcal{V}_h, r \in \mathcal{R}, j \in \mathcal{N}^0 \setminus \{0\}, p \in \mathcal{P} \tag{4.32}$$

In general, these constraints provide weak bounds when the formulation is linearly relaxed, which is one of the main weaknesses of the 4-index MTVRP formulation. The tightest possible bounds are:

$$M_1 = \overline{T}_{pi} + T_{ij} + T_i^U - \underline{T}_{pj} \tag{4.33}$$

$$M_2 = \overline{T}_{p(0)} + T_{(0)j} - \underline{T}_{pj} \tag{4.34}$$

This implies that $\underline{T}_{p0} = \underline{T}_{p(|N|+1)} = 0$ and $\overline{T}_{p0} = \overline{T}_{p(|N|+1)} = T^H$, corresponding to the time window at the depot. It should also be mentioned that in the case where $j = |N| + 1$, i.e. the end-depot, $T_{(0)(|N|+1)}$ is set to 0 in order to ensure a non-decreasing start and end time at the depot for all trip indices in Constraints (4.32).

The formulation of the AFM can further be strengthened by introducing suitable values

of M for Constraints (4.15) and (4.16). The left hand side of Constraints (4.15) cannot be larger than $|N|$. The sum of quantities delivered by vehicle v in period d for customer i can at most be equal to the capacity of the vehicle. Thus, the Constraints (4.15) and (4.16) are changed to:

$$\sum_{i \in N} y_{pvri} \leqslant |N| z_{pvr}, \quad r \in \mathcal{R}, h \in \mathcal{H}, v \in \mathcal{V}, p \in \mathcal{P} \tag{4.35}$$

$$\sum_{m \in \mathcal{M}_i} q_{pvrim} \leqslant Q_h y_{pvri}, \quad i \in \mathcal{N}, r \in \mathcal{R}, h \in \mathcal{H}, v \in \mathcal{V}_h, p \in \mathcal{P} \tag{4.36}$$

Since all vehicles of one vehicle type are identical, the AFM formulation introduces a lot of symmetry in the solution space, which is according to Cattaruzza et al. (2016a) another weakness of the 4-index formulation. Each vehicle is modelled as unique in terms of costs and capacity. It is worth mentioning that the time windows property by nature remove some symmetry in the solution space since the order of customer visits are not arbitrary compared to other VRP formulations. Symmetry breaking constraints can be applied to reduce symmetry in the model formulation. Some suggestions are given below:

$$\sum_{r \in \mathcal{R}} z_{pvr} \geqslant \sum_{r \in \mathcal{R}} z_{p(v+1)r}, \quad h \in \mathcal{H}, v \in \mathcal{V}_h \setminus \{|\mathcal{V}_h|\}, p \in \mathcal{P} \tag{4.37}$$

$$\sum_{r \in \mathcal{R}} \sum_{(i,j) \in \mathcal{A}} C_h^T T_{ij} x_{pvrij} \geqslant \sum_{r \in \mathcal{R}} \sum_{(i,j) \in \mathcal{A}} C_h^T T_{ij} x_{p(v+1)rij}, \quad h \in \mathcal{H}, v \in \mathcal{V}_h \setminus \{|\mathcal{V}_h|\}, p \in \mathcal{P} \tag{4.38}$$

$$\sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{N}} y_{pvri} \geqslant \sum_{r \in \mathcal{R}} \sum_{i \in \mathcal{N}} y_{p(v+1)ri}, \quad h \in \mathcal{H}, v \in \mathcal{V}_h \setminus \{|\mathcal{V}_h|\}, p \in \mathcal{P} \tag{4.39}$$

$$z_{pv(1)} \geqslant z_{p(v+1)(1)}, \quad h \in \mathcal{H}, v \in \mathcal{V}_h, \setminus \{|\mathcal{V}_h|\}, p \in \mathcal{P} \tag{4.40}$$

Constraints (4.37) ensure that the vehicle with the lowest index of its type is assigned the most trips. Constraints (4.38) order vehicles based on travel cost, meaning that vehicles with the lowest indices must incur higher travel costs than higher indexed vehicles of the same type. Constraints (4.39) force vehicles with lower indices to use equally many or more trips than higher indexed vehicles. Constraints (4.37) - (4.39) cannot be applied at the same time, as this may remove optimal solutions. The final symmetry breaking

constraints proposed are (4.40), where vehicle $v + 1$ cannot be used in the planning period if vehicle v is unused. Constraints (4.40) are compatible with the other symmetry breaking constraints proposed. As all symmetry-breaking constraints are based on vehicle type indices, combining constraints will result in the same vehicles being used.

In Bakken et al. (2019), combinations of these symmetry breaking constraints where tested, and (4.37) and (4.40) in combination outperformed the other combinations. These are therefore applied for all instances tested with the AFM in the computational study (Chapter 10).

Chapter 5

A Hybrid Genetic Algorithm

Exact solution methods are in literature proven to perform poor in terms of finding solutions for real-size VRP instances (Cattaruzza et al., 2014a). Therefore, we propose a heuristic solution method - the hybrid genetic algorithm (HGA) - inspired by recently developed VRP literature. The suggested method adopts the concept of genetic algorithms (GA), which was first introduced by Holland (1975). In general, GAs are based upon representing solutions as individuals, and evolve a set of individuals (population) by generating improved solutions (offsprings) through a recombination operator (crossover). The hybrid genetic search algorithm with adaptive diversity control (HGSADC) metaheuristic, first introduced by Vidal et al. (2012), has proven to be efficient for various VRP classes (Vidal et al., 2013b, Cattaruzza et al., 2014a, and Cattaruzza et al., 2016a). The proposed HGA is inspired by the HGSADC framework, and adapted to the periodic, multi-trip VRP with time windows, incompatible commodities, and a heterogeneous fleet. This chapter is initiated with a brief overview of the HGA in Section 5.1, followed by a description of its components in Section 5.2 - 5.11.

5.1 Overview of the Algorithmic Framework

The general scheme of the metaheuristic is described in Algorithm 1. Overall, individuals evolve through generations, where both feasible and infeasible individuals r are maintained in population R. Subsequent to initialization of R (line 1), the population is successively expanded and reduced until a solution of sufficient quality is generated. The population R is initialized with 4μ individuals, and expanded by applying a crossover-procedure to yield λ new child individuals r_c (*offsprings*) from parents r_{p1} and r_{p2} (line

4). With a given probability p^{ed}, each offspring is in line 6 subject to *education*, where different local search operators are applied to possibly improve solution quality. In line 7, an order distribution mixed integer problem (OD-MIP) solver is applied with probability p^{mip} to improve the way orders are distributed to periods in each offspring. The offspring is then, with probability p^{trip}, subject to a trip optimizer to further improve the solution quality (line 8). In line 13, all infeasible individuals in the population are repaired with probability p^{rep}.

When λ new individuals are created, a selection procedure is applied (line 15). Of particular interest is the proposed solution evaluation procedure used during selection, which takes in to consideration both the cost of the solution, and its contribution to the population diversity (Section 5.11). The latter ensures a diverse population, which in previous literature has proven to enhance algorithmic performance (Section 2.2.3).

When survivors are selected, each individual in R is given a possibility to adopt the way orders are distributed to periods from other individuals to improve solution quality. Evaluation of possible changes are based on calculations of a *vehicle filling level fitness* (line 17), which is described in Section 5.10. If the best solution in the population has not improved in N^{div} iterations, a *diversification* (line 20) phase, described in Section 5.11, is applied.

Before the algorithm proceeds with a new iteration, the termination criteria are checked. Termination occurs if either a maximum number of iterations is reached, or the number of iterations without improvement has reached a given maximum, N^{it}. The returned solution is the best individual r in the current population R (line 20).

In subsequent sections, components of Algorithm 1 are described in detail. Initially, the solution representation is presented in Section 5.2. The split-algorithm used to derive multi-trip VRP solutions from the representation of an individual is described in Section 5.3. Solution evaluation is explained in Section 5.4, followed by a description of the crossover-procedure used to generate new offsprings in Section 5.5. Each offspring is subsequently subject to a local search procedure, i.e. education, which is presented in Section 5.6. Then, Section 5.7 describes a formulation of the OD-MIP applied to an offspring to improve they way orders are allocated to periods and trips. The trip optimization operator is presented in Section 5.8, followed by a description of the repair phase in Section 5.9. The vehicle filling level fitness is described in Section 5.10. Finally, we present different population management mechanisms in Section 5.11.

Algorithm 1: HGA

Input: Problem instance
Initialize population R: $|R| = 4\mu$
while number of iterations without improvement $< N^{it}$ and $time < T^{max}$ **do**
  for ($i = 1...\lambda$) {
    Create offsprings r_c from parents (r_{p_1} and r_{p_2}) (CROSSOVER)
    With probability p^{ed}: EDUCATE r_c
    With probability p^{mip}: apply the OD-MIP to r_c
    With probability p^{trip}: apply the TRIP-OPTIMIZER to r_c
    Add r_c to population R
  }
  foreach $r \in R^{infeasible}$ **do**
    With probability p^{rep}: REPAIR r
  end
  Select μ individuals to survive to the next generation
  foreach $r \in$ R **do**
    Change order distribution for r based on VEHICLE FILLING LEVEL FITNESS
  end
  if best solution is not improved for N^{div} iterations **then**
    Diversify population
  endif
end
RETURN the best feasible individual in R

5.2 Solution Representation

Solutions $r \in R$ are characterized by how customer visits are allocated to periods and trips, assignment of trips to vehicles, and the amount of each commodity that is delivered on each trip. In the context of genetic algorithms, *chromosomes* are typically used to represent solutions. The analogy is inspired by the field of biology, where chromosomes are unique organism identifiers. Solutions r are represented as a couplet of the following two chromosomes: a *giant tour chromosome* $GTC(r)$, and an *order distribution chromosome* $ODC(r)$. In subsequent parts of this chapter, *individual* and *solution* will be used interchangeably when referring to a ($GTC(r)$, $ODC(r)$)-couplet. The $ODC(r)$ and $GTC(r)$ are further described in Section 5.2.1 and 5.2.2, respectively.

Note that a solutions is considered infeasible if either its $GTC(r)$ or $ODC(r)$ is invalid. A $GTC(r)$ is considered infeasible if either customer time windows are neglected, or the quantity delivered on any of the trips exceeds the vehicle capacity. An $ODC(r)$ is infeasible if commodity deliveries are scheduled to invalid delivery days, or delivery frequency constraints are neglected.

Let $\mathcal{P}$ be the set of periods and $\mathcal{H}$ be the set of vehicle types, where $p \in \mathcal{P}$ and $h \in \mathcal{H}$. A simple illustration of how individuals are represented for a problem with 2 periods, 5 customers, and 2 vehicle types is given in 5.1. Observe that individual r is represented by a $GTC(r)$ in the 4 uppermost squares, and an $ODC(r)$ in the 2 rectangles below. Nodes represent customers, where the blue color indicates that a customer has an order which must be delivered in the particular period. In opposite, grey nodes represent customers with no orders in the given period, and must therefore not be visited. As for the $ODC(r)$, green and red rectangles represent dividable and non-dividable commodities, respectively. Note that multiple dividable commodities can be delivered in the same period, while each period is restricted to deliver a maximum of one non-dividable commodity.

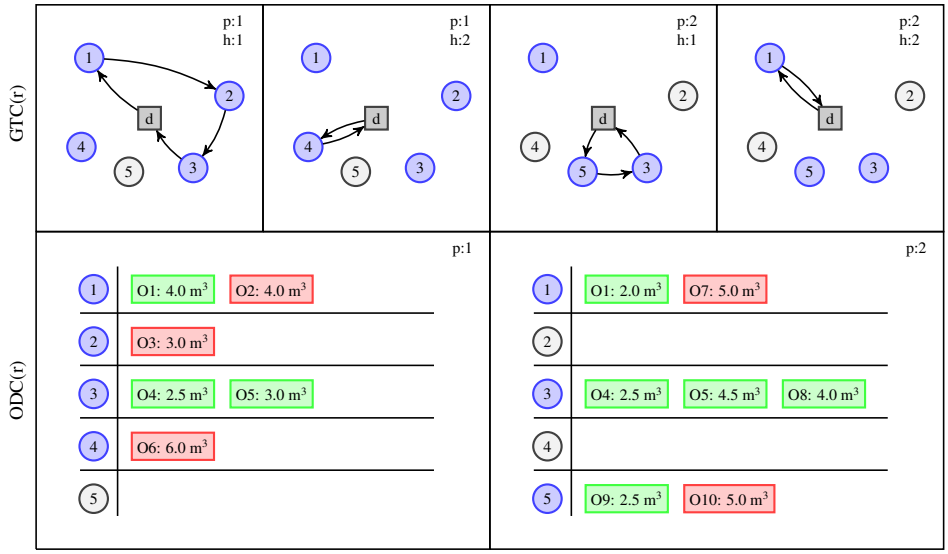


Figure 5.1: Illustration of an individual, including a GTC at the top and an ODC at the bottom, for two periods, five customers and two vehicle types. The green rectangles in the ODC represents dividable orders, while the red rectangles represents non-dividable orders. Grey nodes are customer which cannot be visited in the current period, while blue nodes are customers that must be visited in the current period.

5.2.1 The Order Distribution Chromosome

The $ODC(r)$ holds a particular allocation of commodity quantities to periods such that all orders are delivered during the planning horizon. In contrast to the problems studied in previous research which have adapted the HGSADC-structure (Vidal et al., 2012, Cattaruzza et al., 2014a, and Zhen et al., 2020), customer orders are not fixed to periods in the problem studied in this thesis. Thus, in order to enable the desired flexibility in quantity

assignment, individuals must have the possibility to change the ODCs they are mapped to during the search. The mechanisms which ensure this flexibility, i.e. the *OD-MIP* and the *vehicle filling level fitness*, are described in Section 5.7 and 5.10, respectively.

5.2.2 The Giant Tour Chromosome

The $GTC(r)$ contains a sequence of customers without trip delimiters. The sequence can be interpreted as the order of which customers are visited if they were to be served by the same vehicle. The $GTC(r)$ can be decomposed into an array-structure, where a customer sequence without delimiters is defined for each (period, vehicle type)-couplet. To ease the understanding of the reader, we define the following terminology in subsequent parts of this chapter: $GTC(r)$ refers to the concatenated sequence of customers for all (period, vehicle type)-couplets for individual r, whereas $\mathcal{S}_{p,h}(r)$ refers to the sequence of customers for one particular (period, vehicle type)-couplet in the $GTC(r)$ for individual r. Recall that P is the set of periods and $\mathcal{H}$ is the set of vehicle types, where $p \in P$ and $h \in \mathcal{H}$. The structure of a $GTC(r)$ is illustrated in the following:

$$GTC(r) = \begin{bmatrix} \mathcal{S}_{p=0,h=0}(r) & \cdots & \mathcal{S}_{p=0,h=|\mathcal{H}|}(r) \\ \vdots & \ddots & \vdots \\ \mathcal{S}_{p=|\mathcal{P}|,h=0}(r) & \cdots & \mathcal{S}_{p=|\mathcal{P}|,h=|\mathcal{H}|}(r) \end{bmatrix}$$

Avoidance of delimiters in combination with the array-structure of the $GTC(r)$ enables improved efficiency during transmission of information between iterations (Prins, 2004). Recombination operators, i.e. crossover (Section 5.5), can operate on sequences rather than explicitly accounting for individual trips (Vidal et al., 2013b). However, it requires a separate procedure to divide customers in the $GTC(r)$ into trips and allocate those to vehicles. For this, we propose an *adSplit*-algorithm, which is explained in detail in Section 5.3.

5.3 A Split-Algorithm for the Giant Tour

In order to turn a $GTC(r)$ into a set of trips and assign those to vehicles, we propose a procedure called *adSplit*. For an individual r, the algorithm takes both its $GTC(r)$ and $ODC(r)$ as input. AdSplit is an adaption of the *split*-algorithm suggested by Prins (2004), and later modified by Cattaruzza et al. (2014a) to include the multi-trip property, and altered in Cattaruzza et al. (2014b) to solve for MTVRPs with time windows. However, further modification is needed in order to incorporate a heterogeneous fleet of vehicles and

include the objective of minimizing the number of vehicles used, i.e. incorporate the cost of using a vehicle. The resulting adSplit-procedure is applied each time a new individual is created, or when an existing individual is altered by changing its $GTC(r)$ or $ODC(r)$. Recall that a $GTC(r)$ is composed of $|\mathcal{P}| \times |\mathcal{H}|$ number of customer sequences, $\mathcal{S}_{p,h}$. The adSplit-procedure is applied to all $\mathcal{S}_{p,h}$ in the $GTC(r)$ independently to obtain a complete trip assignment for the individual r.

The AdSplit-procedure is composed of two phases: trip creation (5.3.1), where customers are allocated to trips, and trip assignment (5.3.2), where trips are assigned to vehicles. In the following, the two phases are described. A simple example is introduced in Section 5.3.1, and used throughout the remaining parts of the section.

5.3.1 Trip Creation

The trip creation phase implicitly generates an auxiliary graph, where nodes represent customers with indices based on their order in sequence $\mathcal{S}_{p,h}(r)$. The arc between node i and j represents a trip serving all customers with index from $i+1$ to j in $\mathcal{S}_{p,h}(r)$. Arc costs, i.e. trip costs, are calculated by Equation (5.1), where the following three costs are included: (1) driving cost, $C_h^T t_{i,j}$, where $t_{i,j}$ denotes the driving time and C_h^T the driving cost per time unit, (2) capacity overload, $\omega^Q \overline{q}_{i,j}$, where $\overline{q}_{i,j} = (q_{i,j} - Q_h)^+$ is the actual overload, Q_h is the capacity for vehicles of type h and $q_{i,j}$ is the quantity on the trip from i to j, and (3) *time warp* cost $\omega^T t_{i,j}^W$, where $t_{i,j}^W$ is the incurred time warp. Capacity overload and time warp are adjusted with penalty parameter ω^Q and ω^T, respectively. Time warp is defined as the amount of time exceeding the end of the time window of a customer in the trip, and is adopted from Vidal et al. (2013b). One pays for time warp upon a late arrival to a customer. Whenever time warp is incurred at a customer in a trip, the current time is reduced to the end of the time window before time warp at subsequent customers is evaluated. Note that the unloading time at the previous customer in a trip will also affect the time warp incurred at the subsequent customer, as it delays the time of departure.

$$c_{i,j} = C_h^T t_{i,j} + \omega^Q \overline{q}_{i,j} + \omega^T t_{i,j}^W \tag{5.1}$$

The trip creation phase targets the least costly set of trips that visits all nodes in the auxiliary graph, which is equivalent to the task of finding a shortest path solution. Thus, all possible trips are iteratively considered to search for an improved way of splitting the customer sequence into trips. As the auxiliary graph is acyclic, Bellman's algorithm can be applied to find the shortest path in $O(n^2)$.

A simple example with 4 customers for $\mathcal{S}_{p,h}(r) = [1, 2, 3, 4]$ and depot d is given in Figure 5.2 and Figure 5.3. Time windows are shown in square brackets, demands are given

in round brackets, and edge labels represent driving cost between customers in Figure 5.2. Note that values are selected in order to provide understanding of the procedure, and do not represent realistic instances.

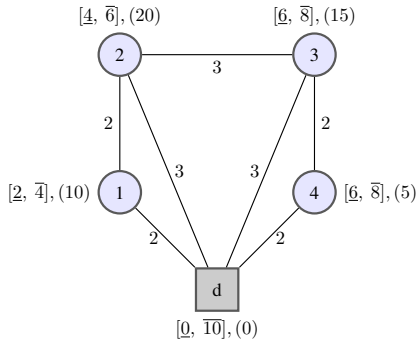


Figure 5.2: A simple example with 4 customers. Time windows are given in square brackets, and demand is given in round brackets. Edge labels represent driving times between customers.

Figure 5.3 shows the auxiliary graph, where each arc represents a possible trip. Arc costs are calculated according to Equation 5.1, where $Q_h = 30$, $\omega^Q = 2$, and $\omega^T = 5$. Unloading time at customers and depot are set to 1 and 0, respectively. For instance, the arc between depot and node 2 represents the trip which starts in the depot, visits customer 1 and 2, and returns back to the depot. Arc cost $c_{d,2} = (2+2+3)+2\times(30-(10+20))^+ +5\times 0 = 7$, where time warp is calculated as follows: $W_{d,2} = (2-4)^+ +((2+2+1)-6)^+ = 0$. The resulting shortest path is illustrated in Figure 5.4.

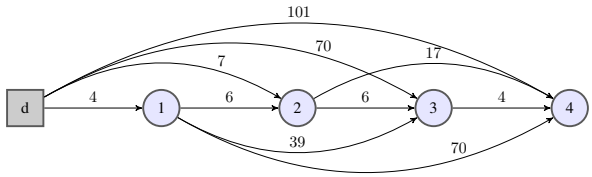


Figure 5.3: Illustration of the auxiliary graph based on the example in 5.2. Arc costs $c_{i,j}$ are calculated according to Equation 5.1.

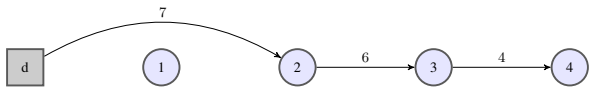


Figure 5.4: Illustration of the shortest path solution based on the example in Figure 5.2.

5.3.2 Trip Assignment

When trips are generated as described in Section 5.3.1, a subsequent phase where they are combined to *journeys* and allocated to vehicles must be applied in order to create a complete problem solution. A journey is a set of consecutive trips that can be assigned to vehicles of a given vehicle type in a given period. For this assignment task, we propose the labeling procedure displayed in Algorithm 2. The algorithm is inspired by the work of Cattaruzza et al. (2016a), where a single-period, multi-trip VRP with time windows and release dates is studied. We propose a modified version of the algorithm, which includes the objective of minimizing the number of vehicles used. In concrete, the label structure is altered, where additional cost terms are incorporated.

The Label Structure

The procedure constructs labels to represent assignment solutions. Each label ι has $|\mathcal{V}_h|+1$ entries, where $\mathcal{V}_h$ is the set of vehicles of type h. Figure 5.5 gives an illustration of the label structure.

$$
\begin{aligned}
\iota = [\, v_1 : \quad & [\, t_1, \quad t_1^W, \quad t_1^A, \quad \overline{q}_1, \quad \mathcal{T}_1 \,] \\
v_2 : \quad & [\, t_2, \quad t_2^W, \quad t_2^A, \quad \overline{q}_2, \quad \mathcal{T}_2 \,] \\
& \ldots \\
v_{|\mathcal{V}_h|} : & [\, t_{|\mathcal{V}_h|}, \, t^W_{|\mathcal{V}_h|}, \, t^A_{|\mathcal{V}_h|}, \, \overline{q}_{|\mathcal{V}_h|}, \, \mathcal{T}_{|\mathcal{V}_h|} \,] \,]
\end{aligned}
$$

Figure 5.5: Illustration of the label structure. All entries are values except $\mathcal{T}_v$, which is a set of trip-references.

The first $|\mathcal{V}_h|$ entries hold values for each vehicle $v \in V_h$, i.e. the total driving time t_v, time warp t_v^W, overload $\overline{q}_\tau = (q_\tau - Q_h)^+$, and the earliest possible time of return to depot t_v^A. Each vehicle entry also includes $\mathcal{T}_v$, which is a set of trips containing the trips allocated to the vehicle. Each time vehicle v is assigned a new trip τ, the values in the associated entry are updated. Values are updated according to Equation (5.2) - (5.7), where $t(\tau)$, $t^W(\tau)$, t_v^L, $t^U(\tau)$, and $t^S(\tau)$ represent driving time, time warp, unloading time, loading time, and latest possible start time respectively for vehicle v and trip τ. Note that $t_v^U(\tau)$ includes waiting time at a customer if time of arrival is before the start of the time window. Thus, value updates for vehicle entries depend on whether or not the vehicle is already assigned other trips, i.e. if $\mathcal{T}_v \neq \emptyset$ or $\mathcal{T}_v = \emptyset$. All vehicles are initialized with $\mathcal{T}_v = \emptyset$ and T^H represents the latest arrival time at the depot. The assignment of values are in the order of the equations given below.

$$t_v \leftarrow \begin{cases} t_v(\tau), & \mathcal{T}_v = \emptyset \\ t_v(\tau) + t_v, & \mathcal{T}_v \neq \emptyset \end{cases} \tag{5.2}$$

$$t_v^W \leftarrow \begin{cases} t^W(\tau), & \mathcal{T}_v = \emptyset \\ t^W(\tau) + (t_v^A - t^S(\tau))^+ + t_v^W, & \mathcal{T}_v \neq \emptyset \end{cases} \tag{5.3}$$

$$t_v^A \leftarrow \begin{cases} t_v^A + t_v(\tau) + t^U(\tau), & \mathcal{T}_v = \emptyset \\ t_v^A + t_v(\tau) + t^U(\tau) + t_v^L, & \mathcal{T}_v \neq \emptyset \end{cases} \tag{5.4}$$

$$t_v^A > T^H \Rightarrow t_v^W \leftarrow t_v^A - T^H + t_v^W \;\&\; t_v^A \leftarrow T^H \tag{5.5}$$

$$\overline{q}_v \leftarrow \begin{cases} (q_\tau - Q_h)^+, & \mathcal{T}_v = \emptyset \\ (q_\tau - Q_h)^+ + \overline{q}_v, & \mathcal{T}_v \neq \emptyset \end{cases} \tag{5.6}$$

$$\mathcal{T}_v \leftarrow \mathcal{T}_v \cup \tau \tag{5.7}$$

The cost of a label, $c(\iota)$, is derived from the values stored in the vehicle entries of ι by applying Equation (5.8), where C_v^T is the cost of one time unit, C_h^F is the fixed cost of using a vehicle of type h in one period, and ω^Q and ω^T are the same penalty parameters as used in Equation (5.1). Note that as the labeling algorithm operates separately on (period, vehicle type)-couplets, C_h^F is similar for all vehicles $v \in \mathcal{V}_h$. If a vehicle v has no trips assigned in label ι, i.e. $\mathcal{T}_v = \emptyset$, the usage cost C_h^F is not included in $c(\iota)$.

$$c(\iota) = \sum_{v \in \mathcal{V}_h} C_v^T t_v + \omega^Q \sum_{v \in \mathcal{V}_h} \overline{q}_v + \omega^T \sum_{v \in \mathcal{V}_h} t_v^W + \sum_{v \in \mathcal{V}_h | \mathcal{T}_v \neq \emptyset} C_h^F \tag{5.8}$$

The Labeling Algorithm

Algorithm 2 displays the labeling procedure. The algorithm takes in a list $\mathcal{T}_{ph}$ of generated trips τ for a specific p and h, and returns a solution where each trip is assigned to a vehicle $v \in \mathcal{V}_h$.

The labeling-algorithm proceeds as follows. Sequentially, each trip $\tau \in \mathcal{T}_{ph}$ is allocated to a vehicle v. Labels are constructed to represent partial assignment solutions. Two sets are maintained to temporarily store labels which are not dominated during the search: $\mathcal{L}^{toExtend}$ and $\mathcal{L}^{extended}$. The former holds all labels that will be extended when a new trip τ is allocated, while the latter is filled during the assignment of trip τ with all labels that are not dominated. Thus, $\mathcal{L}^{extended}$ stores the labels that will be extended when the

successor of τ is to be allocated to a vehicle.

Initially, label ι_0 is created by assigning the first trip τ in $\mathcal{T}_{ph}$ to the first vehicle in ι (line 2). Then, ι_0 is added to $\mathcal{L}^{toExtend}$, whereas $\mathcal{L}^{extended}$ remains empty (line 3). Then, labels are progressively extended from ι_0 by assigning the remaining trips $\tau \in \mathcal{T}$ to vehicles in ι_0 (line 5). Note that as a trip can be assigned to either used or unused vehicles, a label can be extended in at most $\min\{\#vehiclesInUse + 1, |\mathcal{V}_h|\}$ ways. Note that the number of trips assigned to a vehicle v can not exceed $|R|$, defined in Section 4.1.1, which represents the maximum number of trips a vehicle can take in one period.

An extension of a label ι produces a new label ι^{new}. The vehicle entries in ι^{new} inherit those in ι, updating only the entry for the vehicle which is assigned the new trip. In addition, the value of $c(\iota^{new})$ is updated. When the values in ι^{new} are updated, the vehicles v in ι^{new} are sorted in decreasing order based on their earliest possible arrival time at the depot. Next, ι^{new} is added to $\mathcal{L}^{extended}$ if it is not dominated by any other label in the set. Dominance criteria are described later in this section. Similarly, labels in $\mathcal{L}^{extended}$ which are dominated by ι^{new} are removed.

Before a new trip is extracted from $\mathcal{T}_{ph}$ in order to be allocated to a vehicle, the content of $\mathcal{L}^{extended}$ is copied into $\mathcal{L}^{toExtend}$, and $\mathcal{L}^{extended}$ is cleared (line 6). Then, labels in $\mathcal{L}^{toExtend}$ are successively extended, and each resulting label that is not dominated is stored in $\mathcal{L}^{extended}$ (line 13). Finally, when every trip in $\mathcal{T}_{ph}$ have been assigned to a vehicle, the algorithm returns the label $\iota^{best} \in \mathcal{L}^{extended}$ with the lowest label cost $c(\iota)$ (line 18). The final assignment of trips is extracted from $\mathcal{T}_v$ in label ι^{best}.

Label Dominance

In order to speed up the labeling procedure in Algorithm 2, a dominance evaluation step (line 11) is applied to reduce the total number of labels extended. Label dominance is evaluated according to a dominance criteria, which is adopted from the work by Cattaruzza et al. (2016a). Overall, a label ι_i dominates ι_j if and only if Inequality (5.9) holds. $c(\iota_i)$ is the label cost calculated by Equation (5.8). The δ-function calculates the difference between earliest possible arrival time for vehicle v in label ι_i and ι_j, where ω^T is the time warp penalty parameter introduced in Equation 5.1. Note that the δ-function only accounts for time differences for vehicle v if it has a later arrival time in ι_i than in ι_j.

$$c(\iota_i) + \omega^T \sum_{v \in V_h} \delta_v(\iota_i, \iota_j) \leqslant c(\iota_j) \tag{5.9}$$

$$\delta_v(\iota_i, \iota_j) = (t_v^A(\iota_i) - t_v^A(\iota_j))^+ \tag{5.10}$$

Algorithm 2: labelingAlgorithm

Input: List of trips from (p, h)-couplet, $\mathcal{T}_{ph}$
Result: Assignment of every trip to a vehicle, $(\mathcal{T}_v,\ v \in \mathcal{V}_h)$

$\mathcal{L}^{extended} \leftarrow \emptyset$, $\mathcal{L}^{toExtend} \leftarrow \emptyset$
Generate first label ι_0 by assigning the first trip τ_0 in $\mathcal{T}_{ph}$ to the first vehicle v in ι_0
Add ι_0 to $\mathcal{L}^{toExtend}$
Remove τ_0 from $\mathcal{T}_{ph}$
foreach $\tau \in \mathcal{T}_{ph}$ **do**
    Make $\mathcal{L}^{toExtend} \leftarrow \mathcal{L}^{extended}$ and clear $\mathcal{L}^{extended}$
    foreach $\iota \in \mathcal{L}^{toExtend}$ **do**
        foreach $v \in \mathcal{V}_h$ **do**
            Generate new label ι^{new} by extending ι and assigning τ to v in ι^{new}, and updating entries for v
            Sort the vehicles in ι^{new} in decreasing order based on earliest possible arrival time, t_v^A
            if ι^{new} *is not dominated by any* $\iota \in \mathcal{L}^{extended}$ **then**
                Eliminate any $\iota \in \mathcal{L}^{extended}$ dominated by ι^{new}
                Add ι^{new} to $\mathcal{L}^{extended}$
            endif
        end
    end
end
RETURN the assignment scheme $(\mathcal{T}_v,\ v \in \mathcal{V}_h)$ for ι with lowest $c(\iota)$ in $\mathcal{L}^{extended}$

The dominance criteria can be interpreted as follows. For the vehicles in ι_i that arrives later than the corresponding vehicles in ι_j, a time warp cost is incurred. If $c(\iota_i)$ with the additional time warp, calculated by the δ-function (Equation 5.10) is less than $c(\iota_j)$, any extension of ι_i will dominate ι_j, and ι_j is considered dominated. Sorting the vehicles based on earliest possible arrival time at the depot will reduce the total difference in earliest possible arrival time when comparing the vehicles of two labels, and hence reduce the contribution of additional costs from the δ-function in Equation (5.9). This will improve the efficiency of the dominance criteria, as emphasized by Cattaruzza et al. (2014a).

In order to illustrate the labeling procedure, we continue with the example presented in Section 5.3.1 and illustrated in Figure 5.2. We assume that at least three vehicles are available, and that remaining parameter values are unchanged. The labeling algorithm is displayed in Table 5.1, where one label extension is incurred each time a new trip is assigned to a new vehicle. When all trips are assigned, label 6, which uses all three vehicles available, is found to be the best allocation of trips. The complete solution obtained by the adSplit-procedure, i.e. from both trip creation and the labeling algorithm, is illustrated in Figure 5.6.

Table 5.1: Simulation of the labeling algorithm. In total, three trips are assigned, where only label 8 is dominated by label 5 and removed by domination. The best label with the lowest $c(\iota)$ is shown at the bottom of the table. Note that the cost of using a vehicle of type h, C_h^F, is included in $c(\iota)$. No overload is incurred in any label, and max journey duration $T^H = 10$, making the earliest arrival time t_v^A no later than 10. In the dominated column, the number in parenthesis represents which label it is dominated by.

Label Number	Vehicle 1 $[t_1, t_1^W, t_1^A, \overline{q}_1]$	Vehicle 2 $[t_2, t_2^W, t_2^A, \overline{q}_2]$	Vehicle 3 $[t_3, t_3^W, t_3^A, \overline{q}_3]$	Extended Label	Cost $c(\iota)$	Dominated
1	$[7, 0, 9, 0]$	$[0, 0, 0, 0]$	$[0, 0, 0, 0]$	-	27	NO
2	$[6, 0, 10, 0]$	$[7, 0, 9, 0]$	$[0, 0, 0, 0]$	1	53	NO
3	$[13, 7, 10, 0]$	$[0, 0, 0, 0]$	$[0, 0, 0, 0]$	1	103	NO
4	$[10, 6, 10, 0]$	$[7, 0, 9, 0]$	$[0, 0, 0, 0]$	2	117	NO
5	$[13, 5, 10, 0]$	$[6, 0, 10, 0]$	$[0, 0, 0, 0]$	2	109	NO
6	$[6, 0, 10, 0]$	$[7, 0, 9, 0]$	$[4, 0, 9, 0]$	2	77	NO
7	$[17, 13, 10, 0]$	$[0, 0, 0, 0]$	$[0, 0, 0, 0]$	3	167	NO
8	$[13, 7, 10, 0]$	$[4, 0, 9, 0]$	$[0, 0, 0, 0]$	3	127	$YES(5)$
6	$[6, 0, 10, 0]$	$[7, 0, 9, 0]$	$[4, 0, 9, 0]$	2	77	NO

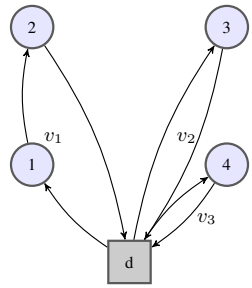


Figure 5.6: Illustration of the trips from the shortest path solution in Figure 5.4 and the assignment to vehicles by the labeling-procedure.

A Heuristic Dominance Criteria

The example in Table 5.1 shows that the number of labels may grow rapidly with each additional trip assigned. In practice, several labels will be a lot worse than the current best label, but not dominated according to the exact dominance criteria (Inequality 5.9). Label 7 in Table 5.1 serves as an example, where all of the three trips are assigned to the same vehicle, which incurs a large time warp cost. The total cost of label 7 is more than twice as large as label 6, and will most likely not be the optimal trip assignment.

Cattaruzza et al. (2016a) suggest a heuristic dominance criteria in order to improve computational efficiency of the labeling algorithm. By introducing an indifference factor $\gamma \geqslant 1$, the number of dominated labels will increase. Label ι_i is said to weakly dominate

label ι_j if and only if Inequality (5.11) and Inequality (5.12) hold.

$$c(\iota_i) + \omega^T \sum_{v \in V_h} \delta_v(\iota_i, \iota_j) \leqslant \gamma c(\iota_j) \tag{5.11}$$

$$c(\iota_i) \leqslant c(\iota_j) \tag{5.12}$$

Inequality (5.12) ensures that no label ι_j is dominated by another label ι_i with an initially larger label cost. If $\gamma = 1$, the heuristic dominance rule is equivalent to the exact dominance rule in Inequality (5.9). A larger value of γ will result in a higher frequency of dominance and hence improve computational efficiency. However, the risk of removing potentially good solutions is increased, as labels with initially high cost can be removed, even though they might be better than the dominant label when more trips have been assigned. Table 5.2 displays the steps in the labeling algorithm when the heuristic dominance rule with $\gamma = 3$ is applied to the simple example in Figure 5.2. Observe that the number of labels generated is reduced from 8 to 6, while the best label identified in Table 5.1 persists.

Table 5.2: Simulation of the labeling algorithm with heuristic dominance. The number of labels dominated are increased, and the number of generated labels are reduced by 2 without removing the best solution. In the dominated column, the number in parenthesis represents which label it is dominated by.

Label Number	**Vehicle 1** $[t_1, t_1^W, t_1^A, \overline{q}_1]$	**Vehicle 2** $[t_2, t_2^W, t_2^A, \overline{q}_2]$	**Vehicle 3** $[t_3, t_3^W, t_3^A, \overline{q}_3]$	**Extended Label**	**Cost** $c(\iota)$	**H.Cost** $\gamma c(\iota)$	**Dominated**
1	$[7, 0, 9, 0]$	$[0, 0, 0, 0]$	$[0, 0, 0, 0]$	–	27	81	NO
2	$[6, 0, 10, 0]$	$[7, 0, 9, 0]$	$[0, 0, 0, 0]$	1	53	159	NO
3	$[13, 7, 10, 0]$	$[0, 0, 0, 0]$	$[0, 0, 0, 0]$	1	103	309	$YES(2)$
4	$[10, 6, 10, 0]$	$[7, 0, 9, 0]$	$[0, 0, 0, 0]$	2	117	342	$YES(6)$
5	$[13, 5, 10, 0]$	$[6, 0, 10, 0]$	$[0, 0, 0, 0]$	2	109	327	$YES(6)$
6	$[6, 0, 10, 0]$	$[7, 0, 9, 0]$	$[4, 0, 9, 0]$	2	77	231	NO
6	$[6, 0, 10, 0]$	$[7, 0, 9, 0]$	$[4, 0, 9, 0]$	2	77	231	NO

5.3.3 Computational Complexity of the AdSplit-Algorithm

The adSplit algorithm is a prominent component of the HGA (Algorithm 1), as it is applied each time an individual is created or altered. As run time analysis have affected design choices in the HGA, a brief comment on the computational complexity of the adSplit-procedure is given in the following. In the first phase of adSplit, the trip creation procedure (Section 5.3.1) is implemented in polynomial time $o(n^2)$ (Vidal et al., 2012), where n is the number of customers in the $GTC(r)$ sequence. The computational time of the subsequent

labeling-procedure in Algorithm 2 can in theory grow exponentially, whereas the number of generated labels can more than double in each iteration. However, dominance criteria will in practice reduce the number of label extensions performed. In any case, there are incentives to limit the use of the adSplit procedure to a minimum, as it is computationally expensive.

5.4 Solution Evaluation

In order to compare individuals when performing various selections, e.g. select parents used to generate offsprings during crossover (Section 5.5) or select individuals to advance to the next generation (Section 5.11), a method to evaluate individuals is needed. Such a *fitness score* is often based upon the objective of the problem at hand (Vidal et al., 2012).

Thus, let $fit(r)$ denote the fitness score of solution r, calculated by Equation 5.13 as the sum of the costs associated with each journey in all (period, vehicle type)-couplets in the solution. Recall that a *journey* j is a set of consecutive trips in the same period p for a given vehicle type h, and that each solution is composed of a set of journeys, $\mathcal{J}_{ph}$, for each (p, h)-couplet. As shown in Equation 5.13, each journey is associated with both its objective cost, and its infeasibility costs. The objective cost is composed of the following three terms: (1) the total driving time in the journey t_j multiplied with the hourly driving cost C_h^T, (2) the cost of using the assigned vehicle C_h^F, and (3) overtime cost at the depot in each period $p \in P$, which sums the quantity delivered to all customers $i \in \mathcal{N}$ in period p, $\sum_{i \in \mathcal{N}} q_{pi}$, and subtracts the the overtime limit $\overline{Q}_p^O$. Note that $q_{pi} = 0$ for customers which are not visited in the period. The second part of $fit(r)$ are the infeasibility costs, which are composed of the following two terms: (1) the total vehicle overload $\overline{q}_j$, and (2) the time warp incurred on the journey, t_j^w. Overload and time warp are adjusted with penalty parameters ω^Q and ω^T, respectively.

$$fit(r) = \sum_{p \in \mathcal{P}} \sum_{h \in \mathcal{H}} \sum_{j \in \mathcal{J}_{ph}} [C_h^T t_j + C_h^F + \omega^T t_j^W + \omega^Q \overline{q}_j] + \sum_{p \in \mathcal{P}} C_p^O (\sum_{i \in \mathcal{N}} q_{pi} - \overline{Q}_p^O)^+ \tag{5.13}$$

As discussed in Section 2.2.3, a sufficient diversity-management in order to avoid premature convergence has proven to be a common denominator in successful heuristic solution methods for various VRP classes. Thus, inspired by Vidal et al. (2012), we propose a second way of evaluating individuals, a *biased fitness score*, which is based upon (1) the fitness score of r, $fit(r)$, and (2) the contribution of r to the population diversity. The biased fitness score is used as basis for selection of individuals to survive from one

generation to the next, which is described in Section 5.11.

The biased fitness for individual r is calculated by Equation 5.14, where $rank^{fit}(r)$ is the rank, i.e. the value relative to other individuals in R, of individual r with respect to $fit(r)$. The number of individuals in R that are guaranteed to advance to the next generation is denoted by n^{elite}, where $n^{elite} = el \times \mu$. Let el be the proportion of the total population that are considered as elite individuals, and μ be the number of offspring generated each generation.

$$bfit(r) = rank^{fit}(r) + (1 - \frac{n^{elite}}{|R|}) \times rank^{div}(r) \tag{5.14}$$

$$div(r) = \frac{1}{n^{close}} \sum_{r_i \in \mathcal{N}^{close}} \mathcal{D}(r, r_i) \tag{5.15}$$

The diversity contribution of an individual r to the population R is denoted as $div(r)$, and $rank^{div}(r)$ is the rank of $div(r)$ among the remaining individuals in R. Equation 5.15 shows how $div(r)$ is calculated, where $n^{close} = nc \times \mu$, where nc is a parameter which represents the neighbourhood size as a proportion of the total population size, $\mathcal{N}^{close}$ is the set of the closest neighbours of r, and $\mathcal{D}(r, r_i)$ is the distance between r and a neighbour solution, r_i. As suggested by Cattaruzza et al. (2014a), the measure of distance is the *broken pair* distance, $\mathcal{D}(\cdot, \cdot)$, which sums the number of pairs of adjacent customers in the $GTC(r)$ of r which are different in the neighbour solution r_i. Thus, $\mathcal{D}(\cdot, \cdot)$ provides a quantification on the number of possible common arcs used for two solutions.

In subsequent parts of this thesis, solution *fitness* refers to Equation (5.13), while the *biased fitness* of a solution refers to Equation (5.14).

5.5 Crossover

In order to generate offsprings from the population, a crossover procedure is applied. Crossover is composed of two sequential phases: an order distribution crossover (5.5.1), and a giant tour crossover (5.5.2). Both phases are based upon inheritance from the same parent individuals, r_{p_1} and r_{p_2}, which are selected through a tournament procedure (Section 5.11). In the following, the two crossover phases are described.

5.5.1 Order Distribution Crossover

No offspring r_c can be generated without having selected an $ODC(r_c)$ that is used whenever the journeys are created and assigned determined by the adSplit-procedure. We therefore propose a separate order distribution crossover in order to determine the $ODC(r_c)$ in advance of the giant tour crossover.

The order distribution crossover takes the parent individuals r_{p1} and r_{p2}, extracts their order distributions $ODC(r_{p1})$ and $ODC(r_{p2})$, and creates a child order distribution $ODC(r_c)$ which inherits from its parent. Iteratively, the child distribution is constructed by inheritance of so-called *order deliveries*, each representing a commodity and one particular realization of periods it is delivered in. If the commodity is dividable, the order delivery includes multiple periods, each with a specified quantity of the particular commodity. If the commodity is non-dividable, only one period is included in the order delivery. A complete order distribution consists of order deliveries for all existing commodities.

The order distribution crossover initially divides all order deliveries into two equally sized sets, O_1 and O_2. First, the child inherits all order deliveries in O_1 from the first parent. Second, it attempts to inherit all order deliveries in O_2 from the second parent. As a customer can only receive one non-dividable commodity in each period, some order deliveries in O_2 might result in an infeasible child distribution, i.e. $ODC(r_c)$, if they are inherited. This situation occurs if parent one delivers a non-dividable commodity m_1 to customer i in period p, and parent two delivers another non-dividable commodity m_2 to the same customer i in the same period p. The order delivery from parent two thus makes the resulting $ODC(r_c)$ infeasible, as customer i already receives a non-dividable commodity in period p. In order to prevent creation of an infeasible $ODC(r_c)$, all order deliveries which generate infeasibility are stored in a separate memory structure while the remaining order deliveries are inherited. When all order deliveries are evaluated, commodities which are not inherited from any parents are sequentially added to the available and valid period with the least total delivery volume.

5.5.2 Giant Tour Crossover

The giant tour crossover takes parent individuals r_{p_1} and r_{p_2}, and the order distribution for the offspring $ODC(r_c)$ generated according to Section 5.5.1, and completes the child individual r_c by generating $GTC(r_c)$.

The proposed algorithm is a *periodic crossover with insertions* (PIX), which is inspired by Vidal et al. (2013b). PIX operates on (period, vehicle type)-couplets, where Ψ is the set of all customer sequences, $\mathcal{S}_{p,h}(r)$. Algorithm 3 displays a pseudocode for the PIX

procedure, which is composed of the following 4 steps: initialization is conducted in Step 0, Step 1 and 2 apply inheritance from parents, and Step 3 ensures that all customers are visited in the child individual.

In Step 0 (line 1-6), all couplets in Ψ are randomly divided into three disjoint subsets, Ψ_1, Ψ_2, and Ψ_{mix}, which correspond to the sets that will inherit from r_{p_1}, r_{p_2}, or a mix of both, respectively. Generating random numbers n_1 and n_2 ensures that the crossover is diversified, as the sizes of the sets will vary, and couplets are randomly selected.

In Step 1, (line 7-14), the new child fully inherits the customer sequences of the (period, vehicle type)-couplets belonging to Ψ_1 from parent 1. In addition, a random subsequence is extracted from each (period, vehicle type)-couplet belonging to the set Ψ_{mix} from parent 1. This sequence is determined by the random numbers α and β, representing two indices in the customer sequence. The offspring inherits all customers from α to β. Note that if $\alpha > \beta$, it will inherit from two intervals, $[\alpha, |\mathcal{S}_{p,h}(P_1)|]$ and $[0, \beta]$. If $\alpha = \beta$, no customers are inherited. Alternatively, whenever $\alpha = \beta$, the offspring could have inherited all customers from parent 1. However, in order to reduce the probability of the crossover being biased by the order of inheritance where parent 1 always comes first, no customers are inherited whenever $\alpha = \beta$.

Figure 5.7: Two different examples on how the α and β will determine which customers that will be inherited from parent 1.

In Step 2 (line 15-22), inheritance from parent 2 is proceeded by inheriting from the combined set ($\Psi_2 \cup \Psi_{mix}$). Each couplet is considered in a random order. When considering a customer sequence $\mathcal{S}_{p,h}(P_2)$, it is important to check whether the customer has already been inherited from parent 1, meaning that the customer is already visited in that period. Figure 5.7 illustrates how different values for α and β affect which customers that are inherited from the different parents.

When Step 1 and Step 2 are completed, some customers might not have been inherited from any of the parents. Step 3 (line 23-29) takes these customers and inserts them at the current optimal position, i.e. where the cost of insertion is minimized. For each missing customer, the optimal insertion is identified during the following three steps: (1) apply the adSplit-procedure (Section 5.3) to update the way r_c is divided into trips, (2) insert the current missing customer in every position in every trip and calculate the associated altered solution cost, and (3) select the position with the least cost deterioration from the original solution.

In order to evaluate insertions whenever a missing customer is re-positioned and a trip

Algorithm 3: PIX

Input: Individual r_{p_1}, Individual r_{p_2}, Order Distribution o
Output: Individual r_c

STEP 0: INITIALIZE SETS:
Initialize empty individual r_c
Select two random numbers, n_1 and n_2, in interval $[0, td]$, where $td = |p| \times |vt|$. Let n_1 be the smallest of these two numbers.
Randomly select n_1 (period, vehicle type)-couplets to form set Ψ_1
Randomly select $n_2 - n_1$ (period, vehicle type)-couplets among the remaining couplets to form set Ψ_2
The $td - n_2$ Remaining (period, vehicle type) couplets form set Ψ_{mix}
STEP 1: INHERIT FROM P_1:
foreach *(period, vehicle type) (p, h)* $\in \Psi_1$ **do**
    Copy all customer visits from $\mathcal{S}_{p,h}(r_{p_1})$ to $\mathcal{S}_{p,h}(r_c)$
end
foreach *(period, vehicle type) (p, h)* $\in \Psi_{mix}$ **do**
    Select two random numbers, α and β, in interval $[0, |\mathcal{S}_{p,h}(r_{p_1})|]$
    Copy all customer visits from α to β from $\mathcal{S}_{p,h}(r_{p_1})$ to $\mathcal{S}_{p,h}(r_c)$
end
STEP 2: INHERIT FROM P_2:
foreach *(period, vehicle type) ((p, h))* $\in \Psi_2 \cup \Psi_{mix}$ **do**
    for *customer c* $\in S_{p,h}(r_{p_2})$ **do**
        if *c* $\notin S_{p,h'}(r_c)$ *for all vehicle types* **then**
            Add c to the end of $\mathcal{S}_{p,h'}(r_c)$
        endif
    end
end
STEP 3: FILL REMAINING CUSTOMERS
foreach *period p* **do**
    foreach *missing customer c in p for Offspring* r_c **do**
        adSplit(r_c, o) for period p
        Insert c at the position in $\mathcal{S}_{p,h'}(r_c)$ which has lowest cost increase at insertion for any vehicle type h'
    end
end
RETURN: r_c

is altered, the deterioration of the fitness for the whole journey, rather than the altered trip only, is evaluated. Thus, if a missing customer is inserted into a trip which, if isolated from its associated journey, is improved with respect to its fitness score (Section 5.4), the insertion can be rejected if it deteriorates the overall solution fitness of the affected journey. Insertions based on single trip fitness, i.e. by first selecting optimal trips and then concatenate them into journeys, journeys with much idle time, i.e. waiting time between customers due to large gaps between time windows, can be generated. This is caused by the time warp between trips, which is not incorporated when trips are evaluated isolated from their associated journeys. Therefore, the adSplit procedure might split the GTC into suboptimal trips, as it does take time warp between trips into consideration. For this reason, the fitness of an insertion is evaluated based on the resulting journey that the modified trip is included in, rather than the single trip which is altered.

Note that Step 3 operates separately for each period, but allows the insertion phase to change the vehicle type that serves each customer. Periods cannot be swapped due to the customer time windows, which are given for a specific period. Thus, if a customer has a time window in a period, it must be served in this particular period for the individual to remain feasible.

Figure 5.8 shows an example of a giant tour crossover for a single period with three vehicle types. Note that as the figure illustrates the crossover for a single period, a complete crossover would additionally include a similar procedure for all remaining periods. In the example, Ψ_1, Ψ_2 and, Ψ_{mix} are of the same size. The (period, vehicle type)-couplets have been distributed in Step 0 as follows: $(p, h_1) \in \psi_1$, $(p, h_2) \in \psi_{mix}$, and $(p, h_3) \in \psi_2$. The indices of α and β for the mix set couplet is $\alpha = 1$ and $\beta = 3$. In Step 1, all customer from Ψ_1 and customers between α and β in Ψ_{mix} are inherited from parent 1. In Step 2, customers from $\Psi_1 \cup \Psi_{mix}$ are inherited from parent 2. Note that customer 6, 7, and 3 cannot be inherited as they already exist in the offspring chromosome. Finally, in Step 3, the missing customers 9 and 10 are inserted at the position with the least cost deterioration.

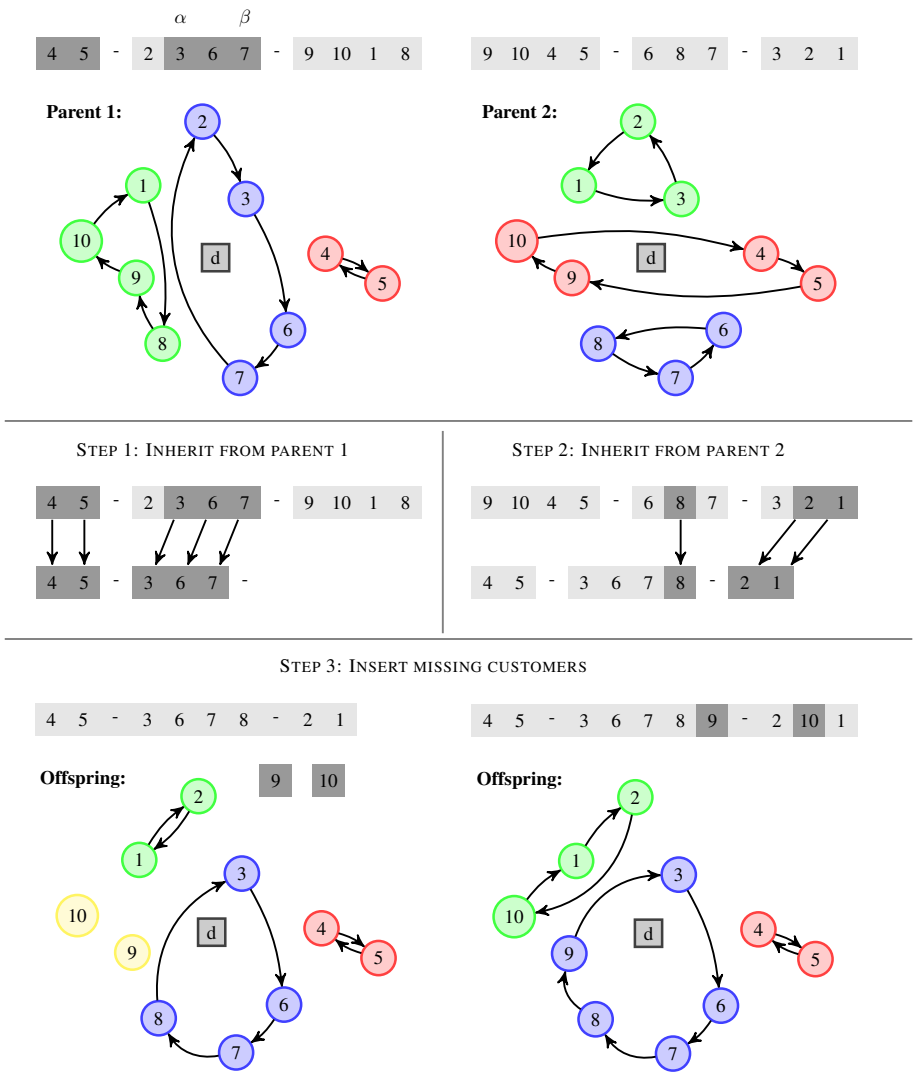


Figure 5.8: All Steps of the PIX crossover on giant tours for a single period with 3 different vehicle types. The colors indicate the different vehicle types. The offspring first inherits from parent 1 before inheriting from parent 2. Afterwards, the customers that has not been inherited are inserted at optimal positions.

5.6 Education

Subsequent to crossover, education is applied for each offspring with probability p^{ed}. The education procedure is composed of a local neighbourhood search with different operators that intend to enhance the fitness score of an individual. The neighbourhood is composed

of the h nearest neighbours, where $h = h^n \times |\mathcal{N}|$, and h^n is in range $h^n \in [0, 1]$. All enhancements are performed on a periodic level, meaning that each customer cannot be interchanged between periods, and journeys are only compared within the same period during the search. Note that the same reasoning does not apply to vehicle types, as two journeys allocated to different vehicle types within the same period can be compared.

For all customers in the giant tour sequence of an offspring, the proposed education procedure is applied for (customer, neighbour)-pairs. Neighbourhoods are determined based on the driving times between each customer and its neighbours. Thus, a (customer, neighbour)-pair will be subject to education within a period p if they are both served in p, and they are close with respect to driving distance. In order to avoid premature convergence, neighbours of the current customer are evaluated in an arbitrary order.

Let $t(u)$ denote the trip which contains customer u in individual r, and let v be a neighbour of u. Note that v does not have to be served in $t(u)$. Let x be the successor of u in $t(u)$, and y be the successor of v in $t(v)$. Based on the suggested education procedure in Vidal et al. (2012), nine different improvement operators are implemented:

- **(M1)**: if u is a customer visit, remove u from $r(u)$ and insert it after v in $r(v)$
- **(M2)**: If u and x are customer visits, remove u and x from $t(u)$, and insert u and x after v in $t(v)$
- **(M3)**:) If u and x are customer visits, remove u and x from $t(u)$, then place x and u after v in $t(v)$
- **(M4)**: If u and v are customer visits, swap u and v
- **(M5)**: If u, x, and v are customer visits, swap u and x with v
- **(M6)**: If u, x, v, and y are customer visits, swap u and x with v and y
- **(M7)**: If $t(u) = t(v)$, replace (u, x) and (v, y) by (u, v) and (x, y)
- **(M8)**: If $t(u) \neq t(v)$, replace (u, x) and (v, y) by (u, v) and (x, y)
- **(M9)**:If $t(u) \neq t(v)$, replace (u, x) and (v, y) by (u, y) and (x, v)

M1-M3 correspond to *insertions*, and M4-M6 correspond to *swap* operations. These moves can be applied to (customer, neighbour)-pairs where customer u and its neighbour v are either within the same trip, or on different trips. Move M7-M9 are 2-opt swaps, where M7 is intra-route, while M8 and M9 are inter-route. Note that any move is accepted only if the fitness of the solution is improved.

When the education procedure is applied, operators M1-M9 are evaluated in an arbitrary order, as proposed in Vidal et al. (2012). Each operator searches for improvements among trips, but the evaluation of an improvement is in performed on the journey that the trip is part of, as described Section 5.4. The rationale behind evaluating journeys rather than single trips subsequent to trip modifications is to avoid a deterioration of journey fitness when single trips are improved locally. In example, such a case might occur whenever two short trips are assigned to the same vehicle, and an insertion of a new customer to one of the trips results in an infeasible journey length such that an extra vehicle is needed. The fitness of each trip is improved, but the need for an extra vehicle results in an overall worsening of the solution fitness.

Note that as the adSplit-algorithm (Section 5.3) is never applied during education, the search for enhancements is done locally among improvement of journeys. Avoidance of adSplit is motivated by its computational complexity.

5.7 An Order Distribution Mixed Integer Program

The ODC-crossover in Section 5.5.1 is applied a priori to both the GTC-crossover (Section 5.5.2) and the education-procedure (Section 5.6). Therefore, subsequent to these mechanisms in the HGA (line 7 in Algorithm 1), the GTC of the offspring has been subject to many changes, whereas its ODC has remained unchanged. In practice, this means that way trips are created and assigned has been altered, but changes have been restricted to deliver the same quantities in each period. In order to find an ODC which fits better to the altered GTC, i.e. the fitness of the individual is improved, we propose an exact mixed integer program (MIP), referred to as the *Order Distribution Mixed Integer Program* (*OD-MIP*). The OD-MIP is applied with probability p^{mip}, and is guaranteed to find the optimal ODC with respect to both vehicle capacities, and overtime incurred at the depot. The OD-MIP takes in the GTC and the associated trip split and journey assignment for the offspring found by adSplit, and identifies the optimal way of allocating customer orders to each period in the planning horizon, accounting for all restrictions on each of the commodity types. The resulting ODC will replace the one generated during ODC-crossover. A formulation of the OD-MIP is provided in the following:

5.7.1 Definition of new Sets, Indices, Variables, and Parameters

The following new sets, indices, variables, and parameters are defined to extend the notation from AFM, provided in Chapter 4.

Definition of Sets

$\mathcal{T}_{ph}$	Set of trips $\tau \in \mathcal{T}_{ph}$ used in period $p \in \mathcal{P}$ with a vehicle of type $h \in \mathcal{H}$
$\mathcal{N}_{ph\tau}$	Set of customers visited in period $p \in \mathcal{P}$ with vehicle type $h \in \mathcal{H}$ in the same trip $\tau \in \mathcal{T}_{ph}$
$\overline{\mathcal{N}_p}$	Set of customers not visited in period $p \in \mathcal{P}$, equal to $\mathcal{N} \setminus \bigcup_{h \in \mathcal{H}} \bigcup_{\tau \in \mathcal{T}_{ph}} \mathcal{N}_{ph\tau}$

Definition of Indices

τ	Trip, where $\tau \in \mathcal{T}_{ph}$

Definition of Variables

u_{pim}	$= \begin{cases} 1 & \text{if commodity } m \in \mathcal{M}_i \text{ is delivered to customer } i \in \mathcal{N} \text{ in period } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$
q_{pim}	Quantity of commodity $m \in \mathcal{M}_i$ delivered to customer $i \in \mathcal{N}$, in period $p \in \mathcal{P}$
q_p^O	Quantity of commodity distributed using overtime in period $p \in \mathcal{P}$

5.7.2 Mathematical Formulation

Objective Function:

$$Minimize \quad \sum_{p \in \mathcal{P}} C_p^O q_p^O \tag{5.16}$$

Constraints:

$$\sum_{i \in \mathcal{N}_{ph\tau}} \sum_{m \in \mathcal{M}_i} q_{pim} \leqslant Q_h, \quad p \in \mathcal{P}, h \in \mathcal{H}, \tau \in \mathcal{T}_{hp} \tag{5.17}$$

$$\sum_{i \in \mathcal{N}} \sum_{m \in \mathcal{M}_i} q_{pim} \leqslant Q_p^O + q_p^O, \quad p \in \mathcal{P} \tag{5.18}$$

$$q_{pim} = Q_{im}^{ND} u_{pim}, \quad p \in \mathcal{P}, i \in \mathcal{N}, m \in \mathcal{M}_i^{ND} \tag{5.19}$$

$$\underline{Q}_{im}^D u_{pim} \leqslant q_{pim} \leqslant \overline{Q}_{im}^D u_{pim}, \quad p \in \mathcal{P}, i \in \mathcal{N}, m \in \mathcal{M}_i^D \tag{5.20}$$

$$\sum_{p \in \mathcal{P}} q_{pim} = Q_{im}^D, \quad i \in \mathcal{N}, m \in \mathcal{M}_i \tag{5.21}$$

$$\sum_{m \in \mathcal{M}_i^{ND}} u_{pim} \leqslant I_{pi}, \quad p \in \mathcal{P}, i \in \mathcal{N} \tag{5.22}$$

$$\sum_{p \in \mathcal{P}} u_{pim} = 1, \quad i \in \mathcal{N}, m \in \mathcal{M}_i^{ND} \tag{5.23}$$

$$\sum_{p \in \mathcal{P}} u_{pim} \geqslant \underline{F}_{im}, \quad i \in \mathcal{N}, m \in \mathcal{M}_i^D \tag{5.24}$$

$$\sum_{p \in \mathcal{P}} u_{pim} \leqslant \overline{F}_{im}, \quad i \in \mathcal{N}, m \in \mathcal{M}_i^D \tag{5.25}$$

$$u_{pim} = 0, \quad p \in \mathcal{P}, \quad i \in \overline{\mathcal{N}}_p, m \in \mathcal{M}_i^D \tag{5.26}$$

Constraints on Variables:

$$u_{pim} \in \{0, 1\}, \quad p \in \mathcal{P}, i \in \mathcal{N}, m \in \mathcal{M}_i \tag{5.27}$$

$$q_{pim} \geqslant 0, \quad p \in \mathcal{P}, i \in \mathcal{N}, m \in \mathcal{M}_i \tag{5.28}$$

$$q_p^O \geqslant 0, \quad p \in \mathcal{P} \tag{5.29}$$

Objective (5.16) aims at minimizing overtime at the depot for all periods, where the

cost per overtime unit is specific for each period. Constraints (5.17) prohibit vehicles from delivering a quantity on a trip that exceeds its capacity. Constraints (5.18) ensure that overtime at the depot is incurred when the total amount of quantity delivered exceeds the overtime limit for the period. Constraints (5.19) force the entire quantity of a non-dividable commodity to be delivered, while Constraints (5.20) force the delivery of a dividable commodity to be within the lower and upper bound. Constraints (5.22) ensure at each customer receives a maximum of one non-dividable commodity each period, whereas Constrains (5.25) ensure that all non-dividable commodities are delivered within the planning horizon. Constraints (5.24) and Constraints (5.25) enforce frequency requirements on dividable commodities. Finally, Constraints (5.26) prohibit delivery of dividable commodities in periods where the customer is not visited.

5.8 Trip Optimization

In order to further improve offsprings generated, we propose a *trip optimization* phase, subsequent to the OD-MIP, which aims at optimizing single trips by changing the order customers are visited in. The work by Zhen et al. (2020) serve as inspiration, where a different variant of the reorder routine is suggested as an attempt to accelerate their proposed hybrid GA. They conclude that the reorder routine significantly improves solution quality and computational performance.

The proposed trip optimization routine works as follows. For an individual r, trip optimization is applied for each trip in its trip split solution found by adSplit, with a given probability p^{trip}. In order to improve computational efficiency, only trips shorter than a threshold value are subject to the trip optimization procedure due to the factorial growth in permutations with additional customers. Thus, the procedure will only be applied to a subset of all the trips that an individual consists of. For each trip where the procedure is applied, every possible permutation of the customer sequence is evaluated, and the resulting permutation which provides the best fitness score is selected. Fitness scores are evaluated for journeys rather than single trips, as a trip might be improved, but still be part of a journey which is inefficient due to time windows.

5.9 Repair

When offsprings are created in crossover and subsequently possibly improved by education, the OD-MIP, and the trip optimizer, they are added to the population R (line 14 Algorithm 1). If enough offsprings are created such that the population has reached a maximum size of $\mu + \lambda$, all infeasible individuals are subject to a repair operator with a

given probability p^{rep}. The repair operator attempts to create incentives for the infeasible individual to become feasible. Repair proceeds by iteratively increasing infeasibility penalties, i.e. penalty parameters for time warp and overload, by means of multiplication with a penalty multiplier. As suggested by Vidal et al. (2012), the multiplier is initialized with a value of 10. In each iteration, both penalty parameters are multiplied with the penalty multiplier. Then, the adSplit-algorithm (5.3) is applied in order to update the way journeys are created and assigned, accounting for the increased penalty parameters. In the end of the iteration, the individual is subject to an education-procedure as an attempt to improve solution quality by means of local search. If infeasibility persists, the penalty multiplier is multiplied with 10. Iterations proceed until the individual becomes feasible, or the multiplier reaches a value of 1000. If the latter occurs, i.e. the individual remains infeasible when the entire repair phase is completed, the modified individual is kept in the population, but has its fitness value re-calculated with penalty parameters which are reset to their initial values. In this manner, fitness values across individuals in the population remain comparable.

5.10 Order Distribution Selection Based on Vehicle Filling Level

The choice of which $ODC(r)$ that is mapped to an individual, i.e. the commodity quantities delivered in each period, will impact the fitness score of the solution. As an attempt to find an $ODC(r)$ which fits better to the $GTC(r)$ individual r is mapped to, i.e. possibly improve its fitness score, we propose a mechanism that gives each individual the opportunity to change the ODC it is mapped to during the search.

Let all ODCs that are being mapped to an individual in the current population R constitute a set of ODCs. If computational time was no concern, one could, for each individual, test its $GTC(r)$ in combination with all ODCs in the set by applying adSplit (Section 5.3), and calculate the resulting fitness scorer. The combination with the lowest fitness would reveal the ODC which is optimal for the individual in combination with its $GTC(r)$. However, applying the adSplit-procedure for each evaluation is computationally demanding (see Section 5.3.3).

In order to evaluate different ODCs for an individual while avoid using the adSplit-procedure, we propose a fitness evaluation method based on the concept of *vehicle filling level*. The underlying motivation for using the filling level as performance measure is based on the intuition of obtaining low-cost solutions whenever vehicle capacities are efficiently exploited. Also, ASKO uses the filling level of a vehicle, i.e. the percentage of each vehicle which is filled with actual goods on a trip, as one of their performance mea-

sures to characterize quality solutions. Filling level is measured as the ratio between the volume in the vehicle which consists of products, and the total volume of the vehicle. The calculation is conducted when a vehicle leaves the depot in advance of a trip.

As shown in Equation 5.30, the filling level for an individual r mapped to a given ODC is calculated as the sum of the filling level on all trips assigned to each vehicle of all vehicle types in every period. Let ω^O and ω^U denote penalty parameters for overload and idle capacity, respectively. As overload generates solution infeasibility, idle capacity is preferred. Thus, the ratio between the parameters ω^O/ω^U is set to 2.

$$fill_r(ODC) = \sum_{p\in\mathcal{P}} \sum_{h\in\mathcal{H}} \sum_{v\in\mathcal{V}_h} \sum_{\tau\in\mathcal{T}_v} \Delta_{fill}(\tau, ODC) \tag{5.30}$$

$$\Delta_{fill}(\tau, ODC) = \omega^O(\sum_{i\in\mathcal{N}_\tau} q_{pi}(ODC) - Q_h)^+ + \omega^U(Q_h - \sum_{i\in\mathcal{N}_\tau} q_{pi}(ODC))^+ \tag{5.31}$$

The filling level $fill_r$ is calculated for all combination of r and each ODC in the current pool of ODCs. Finally, $argmin_{ODC}\{fill_r\}$ then determines the optimal ODC, and the adSplit-algorithm is applied for r with the selected order distribution to update the individual and its fitness score. If the resulting new fitness score deteriorates the original fitness score of r, the changes made to the $GTC(r)$ as a consequence of the new $ODC(r)$ are reversed, and the original solution is re-stored. Similarly, if an originally feasible individual becomes infeasible due to the new $ODC(r)$, no changes are made.

Note that the filling level is a subjective performance measure, and provides no guarantee that quality solutions are obtained. Optimal filling levels does not necessary correspond to optimal journey creation and assignment. For instance, whenever two vehicles operate with low filling levels in a particular solution, the probability of reducing the fleet size in subsequent iterations is increased as the changes needed to get rid of a vehicle and thus reduce the fleet cost are smaller. Such a solution can be more cost efficient. For this reason, the procedure only accepts changes which improve the fitness of an individual. The suggested fitness measure based on filling level will therefore not deteriorate individuals.

5.11 Population Management

The population is composed of two sub-populations: feasible and infeasible individuals. The giant tour crossover (Algorithm 3) generates λ new individuals, and adds them to the corresponding sub-population based on their feasibility status. When all operators which

modify the offsprings generated have been completed, both sub-populations are trimmed down to size μ.

The rationale behind maintaining two sub-population, as proposed by Vidal et al. (2012), is based on enhancement of population diversity. By letting infeasible and feasible individuals be subject to the crossover-procedure together, i.e. they can be selected as reproduction material for the same child offspring, new individuals can be created with a larger range of new characteristics. Quality solutions often tend to be on the border of feasibility, which an infeasible subpopulation can draw solutions toward (Vidal et al., 2012). In addition, by operating with a divided population, it is straight forward to draw candidate solutions from the feasible sub-population without the need for a separate evaluation phase.

5.11.1 Initialization

An initial population is generated by randomly creating multiple GTCs, and map each to separate order distributions which are generated according to the explanation below. Ad-Split is then applied to extract the trips for each solution, and the resulting individuals are added to the subpopulation which matches its feasibility status. The procedure is repeated until 4μ individuals have been created. Note that the initial giant tour chromosomes can be either feasible or infeasible, but initial ODCs are generated in a way that guarantees feasibility.

Generation of Feasible Order Distribution Chromosomes

For an ODC to be feasible, the following must hold: orders must be delivered in customer-specific viable periods, and frequency and volume requirements for every order must be obeyed. These requirements only exists for the dividable commodities, as the non-dividable commodities must be delivered as a whole in one single delivery.

A feasible ODC is generated as follows: for each customer, all orders are extracted. Next, as many empty *delivery packages* are created as there are viable delivery periods for the customer. A delivery package represents all commodities a customer receives on a single customer visit. These delivery packages are then filled with dividable commodity orders before they are assigned to a particular valid period for its corresponding customer. Finally, non-dividable commodity orders are allocated to the delivery periods.

When dividable commodities are allocated to delivery packages, each order is initially assigned a random frequency between upper and lower bounds. Based on this frequency, delivery quantities are set to their minimum volume bound, which is allocated to every

delivery package. The remaining volume for each commodity is then distributed randomly between the delivery packages.

When delivery packages are filled with dividable commodities, they are assigned to viable delivery periods. This is proceeded sequentially, where the package with the highest volume is selected and assigned to the period with the lowest total volume until all packages are allocated to a period. In this manner, the total volume delivered in each period is balanced, which seek to minimize overtime costs at the depot.

Finally, the non-dividable orders are allocated to periods. For each order, the viable delivery period with the lowest total volume is selected in order to minimize overtime. Also, note that a non-dividable commodity can only be assigned to a period if the customer has a time window in the period, and no other non-dividable commodity is delivered in this period.

5.11.2 Penalty Adjustment

Recall from Section 5.3 that when the adSplit-procedure splits customer sequences into trips, it calculates the cost of a trip by applying Equation 5.1. Costs are incurred with trip overload and time warp, and adjusted with penalty parameters ω^Q and ω^T, respectively. These parameters are initially set to large values in order to obtain feasible solutions in the beginning of the search. However, a desirable population contains both feasible and infeasible individuals, as near-feasible individuals enhance population diversity and consequentially reduce the probability of premature convergence (see Section 2.2.3). Therefore, as proposed in Vidal et al. (2012), the penalty parameter values are dynamically adjusted whenever 100 individuals have been altered by the adSplit-procedure (5.3) in order to increase exploration.

Let ξ^{REF} be the target proportion of feasible individuals in population R, and let ξ^T and ξ^Q be the proportion of feasible individuals among the 100 last individuals altered by the adSplit-procedure (5.3) with respect to time warp and overload, respectively. Then, adjustment of ω^Q and ω^T are performed according to Equation 5.32 and Equation 5.33, where $PAR \in \{T, Q\}$. Note that the fixed values are directly adopted from Vidal et al. (2012).

$$\text{if } \xi^{PAR} \leqslant \xi^{REF} - 0.05, \text{ then } \omega^{PAR} \leftarrow \omega^{PAR} \times 1.2 \tag{5.32}$$

$$\text{if } \xi^{PAR} \geqslant \xi^{REF} + 0.05, \text{ then } \omega^{PAR} \leftarrow \omega^{PAR} \times 0.85 \tag{5.33}$$

5.11.3 Population Diversity and Elitism

Vidal et al. (2012) suggest several population management mechanism, in which complement the remainder of the proposed algorithm in preserving both *elitism* and *population diversity* during the search for optimality. The former is the principle of memorizing characteristics of quality solutions, and ensures that the best individuals advance to the next generation. The latter is the concept of expanding the search space, i.e. generate new genetic material. It is motivated by the desire to avoid premature convergence, which is a common challenge in population-based approaches. In the following, we describe three mechanisms implemented to improve the population management: a *diversification phase*, a *survival selection*, and a *tournament selection* of parents during crossover.

Diversification phase

A diversification phase (line 18 in Algorithm 1) is adopted from Vidal et al. (2012), and applied when N^{div} iterations are proceeded without improving the best individual in the population. The diversification phase first eliminates all but the best $\mu/3$ individuals for both subpopulations. Then, 4μ new individuals are generated and added to the suitable subpopulation, proceeded similarly as the initialization procedure described in Section 5.11.1. The diversification phase introduces a significant amount of new genetic material, restoring the lost diversity of the population.

Survival selection/biased fitness

Whenever λ individuals are created by crossover, the μ individuals which will be carried to the next generations must be selected. For this, we propose the survivor selection procedure displayed in Algorithm 4. The procedure applies the biased fitness measure described in Section 5.4. Individuals which are inferior in terms of biased fitness are iteratively discarded, subsequently updating distances and biased fitness measures. The elitism property of the survival selection procedure states that elite individuals, R^{elite}, will advance to the next generation (Vidal et al., 2012).

Algorithm 4: Survival Selection

Input: Subpopulation R
1 **while** $|R| > \mu$ **do**
2 | Remove $r \in R$ with the maximum biased fitness
3 | Update distance and biased fitness measures
4 **end**
5 RETURN reduced subpopulation R

Tournament selection

The parent selection procedure during crossover (Section 5.5) is also subject to a mechanism which enhances population diversity: *tournament selection*. The idea is to draw k random individuals with uniform probability from the entire population, including both feasible and infeasible individuals. The individual with the best biased fitness score (Section 5.4) is selected. This procedure is repeated twice, once for each parent. If the same parent is selected twice, a new selection procedure is conducted for one of the parents. Tournament selection gives a preference to select individuals with the best biased fitness scores, i.e. low-cost individuals which also contribute to population diversity. However, all individuals can in practice be used as genetic material when creating new offsprings, independent of their feasibility status and biased fitness scores. In this manner, solutions are drawn towards the border of feasibility, where high quality solutions are expected to be found (Vidal et al., 2012). Note that the larger value of k, the more greedy the selection procedure will be, as the probability of drawing individuals with good fitness values increases. The selection procedure is a generalization of the binary tournament selection, which is used in Vidal et al. (2012).

Chapter 6

A Multi-Periodic Hybrid Genetic Algorithm

In this chapter, we propose another heuristic solution method, the multi-periodic hybrid genetic algorithm (MPHGA), in order to more efficiently solve practical-sized instances of the problem studied in this thesis. The MPHGA is composed of multiple *periodic* HGAs (PHGA). The PHGA adopts several mechanisms developed for the HGA in Chapter 5. In addition, motivated by the work in Vidal et al. (2013b), the PHGA applies a structural problem decomposition, where the problem of creating journeys and assign them to vehicles are solved separately for each period. Solutions for each periodic problem are combined to form a complete problem solution.

The underlying motivation for applying a periodic decomposition is twofold. First, literature indicate that when solving a complex problem as a set of smaller problems, large-sized instances can be solved more efficiency (Vidal et al., 2013b). Second, the means of solution evaluation is altered when each period is treated as a separate problem, which might improve solution quality. For a solution found by the HGA to be improved between two generations, the sum of changes across all periods must be positive. This means that if an impacting improvement in the solution journeys for one period is discovered while journeys in other periods are deteriorated, the improved journeys are possibly discarded as the solution is evaluated for all periods together. In contrast, when journeys are evaluated for each period separately in the PHGA, periodic improvements can be detected and carried to future generations.

The remainder of this chapter is structured as follows. In Section 6.1, the means of

a periodic problem decomposition is discussed. Section 6.2 provides an overview of the MPHGA, followed by a description of the PHGA in Section 6.3. Finally, Section 6.4 describes how the problem of assigning orders to periods is handled in the MPHGA framework.

6.1 Introducing a Periodic Decomposition

A periodic problem decomposition is motivated by the work in Vidal et al. (2013b). They suggest a population-based solution method to solve a periodic VRP with time windows, including a decomposition phase, which is shown to improve method performance in terms of solution quality and computational efficiency on large-sized instances. They assume that order assignments are fixed to periods, and solve one smaller routing problem for each period. As the assignment of orders to periods are treated as decision variables rather than fixed in the problem studied in this thesis, the same procedure cannot be directly applied.

However, observe that these variables, ensuring that each customer receives its total demand throughout the planning horizon, are the only variables which connect periods. If a fixed order assignment would be assumed, journeys in each period would be independent, and the periodic problem decomposition suggested by Vidal et al. (2013b) could be applied. In practice, it means that one first determines the orders which are delivered in each period, and subsequently, for each period separately, creates journeys and assigns them to vehicles. However, if the fixed order assignment is poor, solution quality might suffer, as journeys for each period are constructed based on the orders which are assigned to this period. Characteristics of quality order assignments cannot be defined without knowing the characteristics of quality journey schedules for each period, and vice versa. In order to apply a periodic decomposition while reducing the risk of obtaining poor solution quality if initial fixed order assignments are poor, the problem of constructing journeys for each period, and the problem of allocating orders to periods, should be solved in an iterative manner.

The proposed MPHGA iterates between finding journeys which fit to fixed order assignments, and updating these order assignments in order to avoid premature convergence. The former is handled by solving multiple PHGAs, each being mapped to its own fixed order assignment. Each PHGA can find journeys for every period separately, as periods are independent when orders are fixed across the entire planning horizon. Section 6.4 is dedicated to describe mechanisms in the MPHGA which alter the order assignments mapped to the PHGAs.

6.2 The Multi-Periodic Hybrid Genetic Algorithm

In this section, the MPHGA is described. One iteration of the MPHGA is displayed in Figure 6.1. In step 0, N^{PHGA} PHGAs are initialized. Note that in contrast to the HGA (Algorithm 1), each individual in population R_n in $PHGA_n$ is mapped to the same, global ODC_n^g. The initial ODC_n^g is constructed as described in Section 5.11. Then, step 1-2 are allocated a fixed amount of time, T^{it}. Step 1 proceeds by solving each PHGA for a given time. The PHGA is described in Section 6.3. In step 2, each PHGA passes its best found solution, i.e. a set of journeys for each period, to the OD-MIP described in 5.7. The OD-MIP finds the optimal way of assigning orders to periods, given the set of input journeys. The new ODC takes place as the global order assignment, ODC^g, for this particular PHGA. Step 1 is repeated with the new ODC^g for the remaining time for the iteration, before the fitness score of the obtained solution is calculated according to Section 5.4. Step 1 and 2 takes a total time of T^{it}. Time is equally divided to solve the routing problem before and after the ODC^g update.

In step 3, the fitness score for each PHGA is used as basis to determine whether its ODC^g should be re-initialized or not in advance of the next iteration. Re-initialization is enforced if one of the following two scenarios occur:

- (1) The fitness value of its obtained solution is not improved from the previous iteration.
- (2) The fitness value of its obtained solution is improved from the previous iteration, but the fitness value is not within a range of 1.4 times the current best found solution by all PHGAs across all previous iterations.

These criteria ensure that solutions which do not appear as promising are discarded. Note that all PHGAs which are selected for re-initialization are assigned different ODCs. Section 6.4 is dedicated to describe how the new ODCs are selected. Step 1-3 complete one iteration of the MPHGA. Successive iterations proceed until a maximum number of iterations without improvements, N^{it}, or a given time limit, T^{max}, is reached.

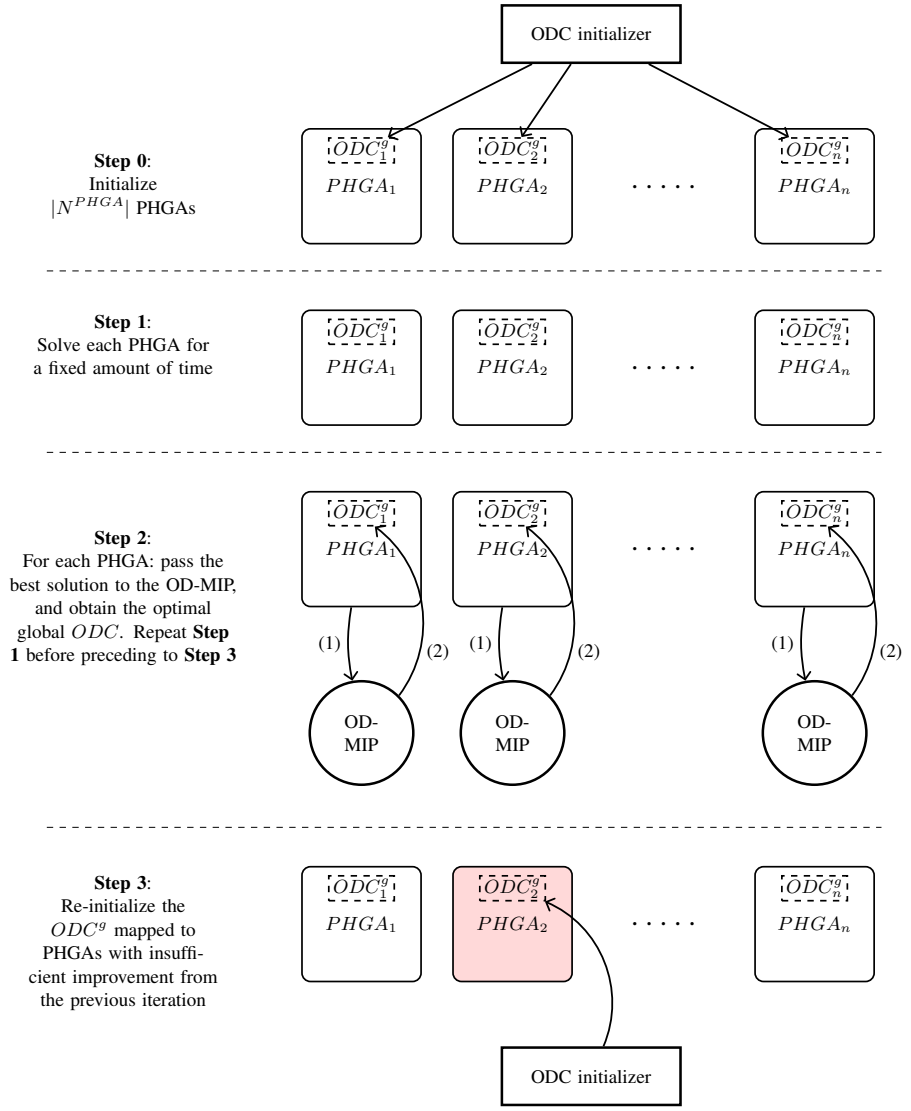


Figure 6.1: Illustration of the MPHGA, applying n PHGAs. Each PHGA is initialized in step 0, and solved in step 1 for a $T^{it}/2$ time. In step 2, journeys found in step 1 are passed to the OD-MIP, which finds an optimal order allocation given the journey solutions found by each PHGA. Step 1 is then re-run for $T^{it}/2$ time. PHGAs which do not appear as promising are re-initialized with new order distributions in step 3. Step 1-3 completes one iteration o the MPHGA.

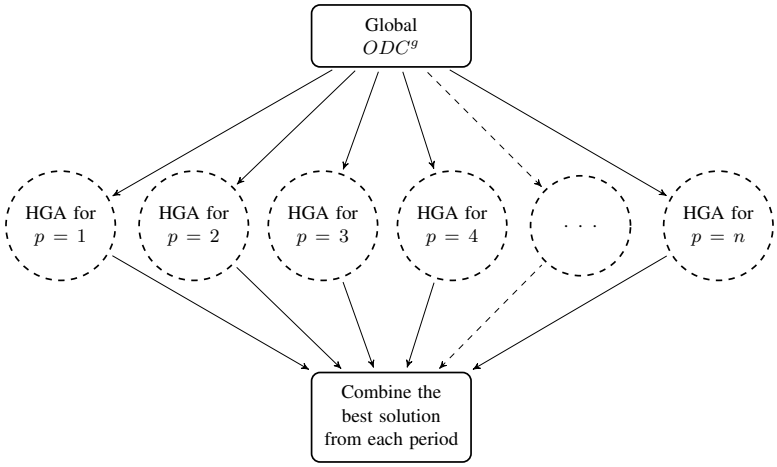


Figure 6.2: Illustration of the PHGA for n periods. The global order distribution is uses as basis for journey creation in each periodic problem, and journey solutions from each period are combined to form a complete solution.

6.3 The Periodic Hybrid Genetic Algorithm

In this section, we provide a description of the PHGA. It adopts several mechanisms developed for the HGA, which are described in Chapter 5. However, rather than solving the problem of creating and allocating journeys for all periods combined, the PHGA solves the problem for each period independently. Figure 6.2 provides a simple illustration of the PHGA. It shows that a global order assignment is assumed to be fixed, and that the problem is decomposed to one problem for each period. The periodic solutions are gathered to form a complete problem solution.

The PHGA is displayed in Algorithm 5. In general, it proceeds by assuming a global ODC^g as input, and solves the problem of creating and assigning journeys independently for each period. The best solution found in each period are gathered and combined to form a complete solution for the problem. For each period p, giant tours for individuals in population R^p are initialized according to the procedure described in Section 5.11 (line 2). Note that in contrast to the HGA (Algorithm 1), the population of individuals is specific for each period p, denoted R^p. All individuals $r^p \in R^p$ are mapped to the same global ODC^g. However, they are mapped to different giant tours $GTC(r^p)$, which hold customer sequences for period p only. Subsequent to initialization, the population is expanded with μ offsprings by means of crossover (line 5). With given probabilities, each offspring is subject to education (line 7) and trip optimization (line 8). Offsprings are added to the population in line 9. Subsequently, repair is applied to all infeasible individuals with probability p_{rep} in line 12. Crossover, education, trip optimization, and

repair are mechanisms adopted from the HGA and described in Section 5.5, 5.6, 5.8, and 5.9, respectively. Recall from Chapter 5 that these mechanisms are structured such that they operate on each (period, vehicle type)-couplet independently. Therefore, they can be adopted by the PHGA without modifications. When each periodic problem is solved for a given amount of time, solution from each period with the best fitness score (Section 5.4) are gathered (line 16) and combined to form a complete solution.

Note that the adaptive penalty adjustment described in Section 5.11 is adopted from the HGA. It it is applied for each periodic problem independently, as the search may benefit from the flexibility of having different penalty parameters in each period. However, when complete solutions obtained by a PHGA are compared across generations, they are all evaluated based on initial penalty parameters to ensure a fair comparison.

Algorithm 5: PHGA

Input: A global order distribution ODC^g

1 **do for all** $p \in P$
2 Initialize population R^p: $|R^p|$ =4μ
3 **while** time limit not reached **do**
4 **for** ($i = 1...\lambda$) $\{$
5 Create offsprings (r_c^p) from parents ($r_{p_1}^p$ and $r_{p_2}^p$) (**GIANT TOUR CROSSOVER**)
6 With probability p^{ed}: **EDUCATE** R_c^p
7 With probability p^{trip}: apply the TRIP-OPTIMIZER to r_c^p
8 Add r_c^p to population R^p
9 $\}$
10 With probability p^{rep}: **REPAIR** any infeasible individuals in R^p
11 Select μ individuals to survive to the next generation
12 **end**
13 **end**
14 Gather the best solution from each period to construct a complete problem solution: $r^{complete}$
15 **Return:** solution $r^{complete}$

6.4 Updating the Global Order Distribution Chromosomes

As discussed in Section 6.1, if the ODC^g which is used as basis for creation of journeys in a PHGA remains fixed during the search, a poor initial ODC^g might impose restrictions on the journey creation such that poor solutions are obtained. Therefore, the MPHGA in Figure 6.1 is composed of two mechanisms which enable the possibility of a PHGA to

change the ODC^g it is mapped to during the search.

One of the mechanism is implemented in step 2 of Figure 6.1. When each PHGA is solved for a given amount of time, the OD-MIP described in Section 5.7 is applied to generate a new ODC which is optimal for the best journey solution found in step 1. The new ODC takes the place of the ODC^g which was used in step 1, and the PHGA is re-solved to obtain a new set of journeys. The second mechanism which enables changes of ODC^gs is found in step 3 in Figure 6.1, where some PHGAs are selected to have their ODC^g re-initialized based on evaluations of solutions obtained in step 2. The selected PHGAs are re-initialized in an arbitrarily order. For each selected PHGA, re-initalization of new ODCs is proceeded as follows:

1. GENERATE A SET OF NEW $ODCs$:
   A set of new ODCs, denoted ODC^{new}, is generated according to the initialization process described in Section 5.11. A total of $|ODC^{new}|$ new ODCs are generated ad subject to be selected for re-initialization. Recall that only feasible ODCs are generated.

2. EVALUATE EACH ODC IN ODC^{new}:
   Let ODC^{used} be the set of all ODC^gs generated by the OD-MIP in step 2 in Figure 6.1, i.e. the set of all ODC^gs which are mapped to any PHGA. Each $ODC \in ODC^{new}$ is assigned a diversity score, which is calculated as a sum of how much it differs from each ODC in ODC^{used}. The difference between two arbitrary ODCs (e.g. ODC_a and ODC_b) is calculated according to Equation (6.1), where $q_{pi}(ODC_a)$ is the quantity delivered to customer i in period p for ODC_a.

$$\Delta(ODC_a, ODC_b) = \sum_{p \in \mathcal{P}} \sum_{i \in \mathcal{N}} |q_{pi}(ODC_a) - q_{pi}(ODC_b)| \tag{6.1}$$

3. SELECT ONE ODC:
   Finally, the most diverse ODC in ODC^{new} is assigned to the PHGA, and takes the place as its ODC^g in the next iteration.

Chapter 7

A Multi-Periodic Artificial Bee Colony Algorithm

In this chapter, we propose a multi-periodic artificial bee colony optimization algorithm (MPABC) to solve the problem studied in this thesis. Artificial bee colony (ABC) optimization is a swarm-inspired metaheurstic, first introduced by Karaboga (2005), inspired by simulation of a honey bee swarm in the search for problem solutions. The ABC is built on the concept of allowing brief exploration and quick abandonment of parts of the solution space which do not seem promising. Literature has shown that when applied to other hard discrete combinatorial problems, the ABC framework obtains quality solutions within a reasonable amount of time (Iqbal et al., 2015).

This chapter is structured as follows. In Section 7.1, we describe how the MPABC adopts the structure from the MPHGA described in Chapter 6. Finally, an overview of the *periodic* ABC (PABC) algorithm, which is an important component of the proposed MPABC, is provided in Section 7.2.

7.1 Adopting the Structure of the Mulit-Periodic Hybrid Genetic Algorithm

The proposed MPABC adopts most of its core structure from the MPHGA described in Section 6.2. Recall that the MPHGA is composed of 4 steps, as illustrated in Figure 6.1, where step 0 and 3 are adopted by the MPABC. The ODC-initializer in step 0 and

step 3 is applied similarly. However, the MPHGA and MPABC are distinguished by two elements: first, where the MPHGA is composed of solving multiple PHGAs (see Section 6.3) in parallel to find a set of problem solutions in step 1, the MPABC instead solves multiple *periodic* ABCs (PABC). Similar as in the PHGA, the PABC exploits a periodic decomposition to solve the problem of creating and allocating journeys independently for each period. Secondly, where each PHGA applies the OD-MIP (5.7) once in one iteration of the MPHGA (step 2), the OD-MIP is applied several times for each PABC during one iteration of the MPABC. Thus, in the MPABC, step 1 and step 2 in Figure 6.1 are repeated several times in one iteration, before the algorithm proceeds to step 3.

Note that one iteration of the PABC and PHGA are both limited to T^{it} time, and both the MPHGA and the MPABC are terminated if run time exceeds T^{max}. The remainder of this chapter is dedicated to describe the PABC, and assumes that the MPABC framework is familiar to the reader.

7.2 The Periodic Artificial Bee Colony Algorithm

The PABC solves the problem of creating and allocating journeys separately for each period in the planning period. Recall that similar as in the PHGA (Section 6.3), the periodic problems are independent due to the assumption of a predetermined global ODC^g. In advance of the solution process, the problem for each period extracts the order distribution from the ODC^g for this particular period, and creates and allocates journeys such that these quantities are delivered.

An overview of the PABC is displayed in Algorithm 6. It is composed of four main phases: (0) initialization, (1) employee phase, (2) onlooker phase, and (3) scout phase. Phase 1, 2, and 3 constitute one iteration, which is repeated N^{ODC} times before the OD-MIP is applied by combining the best journey solutions found in each period. This is repeated until run time exceeds T^{it}. The algorithm applies two local search mechanisms: the bee position update (BPU), and the local enhancement scheme (LES). In the remainder of this section, we first describe how solutions are represented in the PABC, followed by how mechanisms are adopted from the HGA proposed in Chapter 5. Then, a detailed description of the four algorithmic phases is provided. Finally, the local search mechanisms exploited in the PABC, i.e. the BPU and the LES, are described.

7.2.1 Solution Representation

A complete problem solution must, for each period in the planning horizon, contain the following information: the selected set of journeys, the customers that are visited in each

Algorithm 6: PABC Algorithm

Input: ODC^g

do for all $p \in P$
  PHASE 0: INITIALIZATION
  $E \leftarrow$ Set of N^{employees} employees
  $O \leftarrow$ Set of N^{onlookers} onlookers
  foreach $e \in E$ **do**
    initialize e's position to random a position in solution space
  end
  for (N^{ODC} iterations **)** {
    PHASE 1: EMPLOYEE PHASE
    foreach $e \in E$ **do**
      Update its current position by the *Bee Position Update* (Equation 7.4) and the *Local Enhancement Scheme* (Algorithm 7)
    end
    PHASE 2: ONLOOKER PHASE
    foreach $o \in O$ **do**
      Inherit employee position using a *roulette wheel* (Equation 7.1) selection based on employees fitness scores
      Perform local search by a *random perturbation* (Equation 7.2) and the *Bee Position Update* (Equation 7.4)
    end
    foreach $e \in E$ **do**
      Update e position to best found position in the local search update trials for e (Function 7.3)
    end
    PHASE 3: SCOUT PHASE
    foreach $e \in E$ **do**
      if number of trials for e exceeds limit **then**
        Find N^{scouts} new positions in the solution space
        Assign e the best position found among N^{scouts} new positions
      endif
    end
  }
end
Combine best solutions found in each period P **Return** Best found solution

journey, which vehicles that are assigned to each journey, and the quantity that is delivered in each journey.

In the PABC, a solution is represented by a position in a continuous solution space, as described in Section 2.2.2. Similar as Zhen et al. (2020), the PABC uses a random key encoding scheme of the solution space, and a mapping procedure is applied to obtain discrete solutions. Discrete solutions are represented as giant tours for each vehicle type, i.e. customer sequences, as used in the HGA and described in Section 5.2. The continuous search space has one dimension for each customer. The range of each dimension in the position corresponds to the number of vehicle types. The range is therefore $[0, |\mathcal{H}|]$, where $\mathcal{H}$ is the set of vehicle types, and the value in the dimension representing a customer determines which vehicle type it is assigned to.

Figure 7.1 illustrates the representation of a solution for one period with 10 customers and two vehicle types. The upper part of the figure shows the encoded continuous solution representation, and the lower part shows the decoded discrete solution. The continuous solution is composed of two arrays, where the upper array corresponds to the IDs of the customers to visit. The customer IDs corresponds to the customers visited in this period. The lower array represents a position in the search space for the corresponding customer. Blue and purple colors are used to differentiate between the two vehicle types. The decoded discrete solution is represented as one giant tour per vehicle type. Each giant tour is composed of two arrays, where the upper array holds the customer ID. The lower array contains, for the corresponding customer, a fractional number. The integer part corresponds to which vehicle type the customer is assigned to, and the fractional part corresponds to the order which the customer appears in the giant tour in the continuous representation, i.e. the position in the search space.

In summary, the encoded solution represents a position, where the upper array holds its dimension, and the position in the solution space is found in the lower array. When decoded to a discreet solution, the upper array corresponds to giant tours, and the lower array determines the order of which customers appear in the giant tour.

7.2.2 Mechanisms Adopted from the Hybrid Genetic Algorithm

When a position in the continuous search space is decoded into a discrete giant tour representation, mechanisms that are developed to operate on discrete represented solutions for the HGA (Chapter 5) and the PHGA (Chapter 6) can be adopted by the PABC. Solutions are evaluated according to the fitness calculation described in Section 5.4.

The adSplit-procedure (Section 5.3) developed to transform the giant tour representation for a given vehicle type and period into a set of journeys in the HGA and PHGA, is

Encoded continuous solution representation:

Customer ID	1	3	5	6	7	10	13	15	16	18
Position	0.23	1.45	1.02	0.19	0.68	1.22	0.91	0.57	0.72	1.55

Decoded discrete solution representation:

6	1	15	7	16	13
0.19	0.23	0.57	0.68	0.72	0.91

Giant for vehicle type 0

5	10	3	18
1.02	1.22	1.45	1.55

Giant tour for vehicle type 1

Figure 7.1: An illustration of a solution representation for one period with 10 customers and 2 vehicle types. The upper part shows the representation in the continuous space. The lower part consists of the discrete solution representation for both vehicle types. Colors are used to highlight that the giant tours in the discrete representation are representing one vehicle type each, where purple represents vehicle type 0, and blue represents vehicle type 1.

adopted to create solutions from the discrete representation of bee positions in the PABC. Recall that adSplit allows creation of solutions which are infeasible with respect to time warp and overload, but penalizes infeasible solutions with costs that are multiplied with penalization parameters ω^T and ω^Q, respectively. These parameters can, as explained in Section 5.11, be adaptively adjusted according to initial target values for the proportion of feasible solutions created, ξ^{REF}.

However, the efficiency of adaptive penalty parameter adjustment in the PABC is questionable, as there are less incentives to create infeasible solutions than in the HGA. Recall that the motivation for creating infeasible solutions in the HGA is that quality solutions often lie on the edge of feasibility. As new solutions are created by using existing ones, having infeasible solutions in the population might be beneficial when creating new individuals during crossover (Section 5.5). In contrast, previous solutions obtained in the PABC are not memorized, except from the global best solution found. Selection of new positions to explore in the search space are therefore less affected by search history. We dedicate the parameter tuning in Section 10.1 to investigate whether the adaptive penalty adjustment should be implemented for the PABC, or if the penalty parameters ω^T and ω^T should be fixed to large values in order to prevent creation of infeasible solutions.

7.2.3 The Algorithmic Steps Described in Detail

The following section describes the four phases presented in Algorithm 6.

Phase 0: Initialization

The PABC is initialized with a set of randomly generated solutions, where each employee bee is assigned a random position in the solution space. The current global best found position is initialized as the position of a random employee in the first iteration.

Phase 1: Employee Phase

Each employee performs a local search near its current position in the solution space. The local search consists of a bee position update (BPU) and a local enhancement scheme (LES), both described in Section 7.2.4. If the local search yields an improved solution, the position is updated. Finally, the employed bees recruit onlookers, as described in Section 7.2.3.

Phase 2: Onlooker Phase

The onlookers are dedicated to a more thorough exploration of the space near the positions of the employed bees. During the onlooker phase, each onlooker updates its position, using a *roulette wheel selection*, to a position among the positions found in phase 1. Roulette wheel selection is based on the fitness of the decoded solutions. The probability of selecting an employee position is given by Equation 7.1, where $p^{e'}$ denotes the probability of selecting the position of employee e', and $employees$ is the set of all employees in the swarm. Note that the probability of being selected is larger for positions with lower fitness values, meaning that good positions in the solution space will be explored more thoroughly. This is motivated by the idea that better solutions can be found near already good solutions.

$$p^{e'} = \frac{\frac{1}{fitness(e)}}{\sum_{e \in employees} \frac{1}{fitness(e)}} \tag{7.1}$$

When an onlooker has selected which location to follow, its position is updated in two steps. First, its position in each dimension i, x_i^o, is updated according to Equation 7.2. This update is based on both the position of the employee it has selected to follow, x_i^e, and a *random perturbation* ρ^o. Let ρ^o be randomly selected in the range $[-\rho^o, \rho^o]$, and R^i be a subspace of the solution space. The size of the subspace determines how many dimensions that will be changed, and is tuned to balance the size of the local search. Second, the BPU equation, described in Section 7.2.4, is applied in order to adjust the position according to the globally best found solution, and a randomly selected neighbour.

$$x_i^o = x_i^e + \rho^o, i \in R^i \tag{7.2}$$

When all onlookers have updated their positions, their decoded solutions are evaluated based on the fitness calculation described in Section 5.4. Then, for each employee, the fitness value of its position is compared with the best position found among the onlookers that followed this particular employee. If the onlooker position improves the employee position, the employee inherits the position found by the onlooker.

Phase 3: Scout Phase

The scout phase executes abandonment of poor positions and positions which have been explored thoroughly enough to assume that no better solution can be found, and ensures exploration of new positions in the search space. These mechanisms are enabled by assigning each employee a variable which keeps track of the number of *trials* that are used to explore its position. In each trial, the employee registers a new fitness score for its position (employee phase), and the onlookers which select to follow this employee (onlooker phase) search for further improved positions in the area of the new employee position. The resulting employee fitness is denoted fit^{new}. Equation 7.3 shows how the registered number of trials for an employee is updated each time a new trial is incurred. If the fitness score of the position found in a new trial, fit^{new}, is worse than the fitness of the best solution found among all previous trials, fit^{old}, the number of trials is incremented with 1. A similar increment is incurred if fit^{new} is less than fit^{old}, but still worse than the globally current best found solution, fit^{best}, multiplied with a parameter λ^{gb}, which is a number larger than 1. Note that if this is the case, the fitness is still updated to fit^{new} and the position is adopted by the employee. The reason for imposing a comparison requirement with the fit^{best} is to abandon less promising solutions faster. If fit^{new} both improves fit^{old}, and is less than $\lambda^{gb} \times fit^{best}$, the number of trials is reset to 0 and more trials will be used to explore the position.

$$trials^{t+1} = \begin{cases} trials^t + 1, & \text{if } fit^{old} < fit^{new} \\ trials^t + 1, & \text{if } fit^{old} > fit^{new} \text{ and } fit^{new} > \lambda^{gb} \times fit^{best} \\ 0, & \text{if } fit^{old} > fit^{new} \text{ and } fit^{new} < \lambda^{gb} \times fit^{best} \end{cases} \tag{7.3}$$

If the number of trials for an employee position reaches a maximum limit, the employee and the onlookers followed the employee have been searching in an area near the initial position for too long without finding sufficient improvements. This location is therefore abandoned, and the employee is assigned a new position. To find a new position, a number

of S scouts are sent out to each find a random position in the solution space. The best found position is selected as the new position for the employee.

7.2.4 Local Search Mechanisms

The PABC Algorithm is based on the assumption that quality solutions can be found in the neighbourhood of other quality solutions. The benefits of having a continuous solution space makes the local search procedure simpler, as no positions within the dimension bounds are invalid. We propose two local search mechanisms which are used to find new positions and update the ones that are assigned to bees in the swarm during the search. The mechanisms are described in the following sections.

The Bee Position Update Equation

When a position is updated by the bee position update (BPU) equation in Algorithm 6, the adjustment is based on two positions: one being the globally best found position so far, the other being the position of a random neighbour bee. The use of the globally best found solution is inspired by the common way to update positions in particle swarm optimization algorithms, as described in Section 2.2.2. By letting a random neighbour influence the position update, search diversity is enhanced. The BPU is given by Equation 7.4.

$$x_i^b = \omega_n \times (x_i^n - x_i^b) + \omega_{gb} \times (x_i^{gb} - x_i^b), i \in R^i \tag{7.4}$$

where x_i^b is the position of a bee in the i'th dimension. Let x^n be the position of a random neighbour, and x^{gb} the globally best found position so far. Also, ω_n and ω_{gb} correspond to random weights in a given range, where $\omega_n \in [-1/2k^n, k^n]$, which means that a bee can possibly move away from its neighbour. Let ω_{gb} be in the range $\omega_{gb} \in [0, k^{gb}]$, meaning that the bee can only move towards the globally best position. R^i is a subspace of the solution space, containing the dimensions which are being updated. The weights, as well as the size of the subspace, control the size of the neighbourhood of the bee. Figure 7.2 shows a simple example of the BPU in two dimensions.

A Local Enhancement Scheme

In addition to the position update, a local enhancement scheme (LES) is applied in Algorithm 7.2. The LES is only applied to the employee bees, as it is a computationally expensive operation since the adSplit procedure in Section 5.3 is applied. The LES is a greedy operation, which only accepts changes which improves the solution. The number

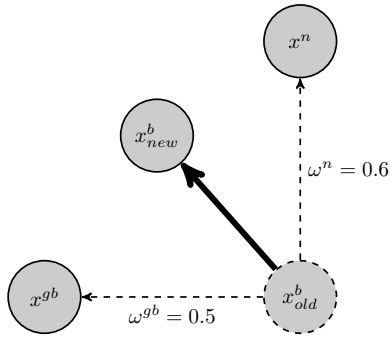


Figure 7.2: An example of how a position for x^b is updated by the BPU in two dimensions. Weights ω^{gb} and ω^n shows how much x^b is affected by the global best position x^{gb} and a random neighbour x^n.

of operations used, k, is updated according to Algorithm 7.

The enhancement scheme consists of three operators: *swap, reinsertion*, and *reverse*. For each operator, two random indices are selected, and the operation is applied for those indices, as shown in Figure 7.3. The operators are only applied within the same vehicle type. As in the *education* scheme proposed for the HGA (Section 5.6), only moves which improves solution fitness are applied. In order to evaluate the fitness change when a solution is altered by any of the operators, the adSplit-procedure (see Section 5.3)is applied for the given giant tour and vehicle type. Algorithm 7 displays the LES scheme for the PABC. Note that the number of enhancements conducted depends on how many improvements that are found.

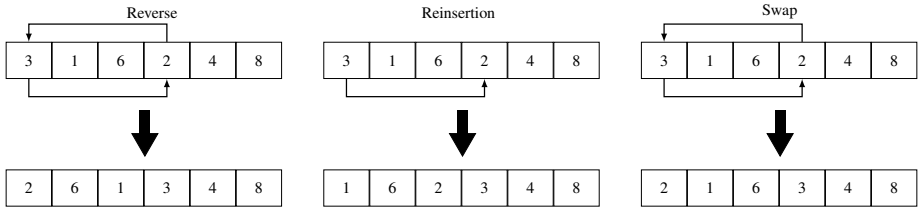


Figure 7.3: Three local operators applied to a sequence of customers.

When LES is applied, the operators make changes to the solutions in their discrete representation. More concretely, they alter the sequence of customers in the giant tours. When the LES is finished, the altered discrete solutions, i.e. the updated bee positions, are transformed back to the continuous space. Figure 7.4 illustrates the LES, where a solution for 10 customers and 2 vehicle types is subject to a local enhancement, and subsequently transformed back its representation in the continuous space. The colors are used to highlight that for customer 4, its position is changed from .42 to .79 by the LES. Note that it

Algorithm 7: Local enhancement scheme procedure for the PABC algorithm.

```
   Input: Bee position arrays
 1 for vehicle type h ∈ H do
 2 |   S_0 ← initial giant tour for vehicle type h
 3 |   k ← 0
 4 |   while k < n^LES do
 5 |   |   operator ← random(swap, reverse, reinsertion)
 6 |   |   S ← operator(S_0)
 7 |   |   if fitness(S) < fitness(S_0) then
 8 |   |   |   S_0 ← S
 9 |   |   |   k ← 0
10 |   |   endif
11 |   |   else
12 |   |   |   k ← k + 1
13 |   |   endif
14 |   end
15 end
```

remains served by vehicle type 0, as seen by the integer part of the position number.

Recall from Section 7.2.1 that a discrete solution is represented by two connected arrays for each vehicle type. The upper array contains the customer sequence, and the lower array contains fractional numbers where the integer part represents the vehicle type the customer is visited by, whereas the fractional part holds the position of the customer in the continuous space. Before the LES is applied to the discrete solution representation in Figure 7.4, the upper array for vehicle type 0 contains the following customer sequence: $[1, 4, 3, 10, 2, 6]$. When the LES operators have finished their enhancement, the new array is $[10, 3, 2, 1, 4, 6]$. Note that the position array remains unchanged while the customers sequence is changed. This means that when the order of customer visits is changed, each customer inherits the position in the continuous solution space from the customer it replaces in the sequence. Colors are used to illustrate that the position of customers after they have been reordered are the ones who are used when the discrete solution is encoded to the continuous space.

Discrete representation before local enhancement:

1	4	3	10	2	6
0.23	0.42	0.49	0.61	0.79	0.90

7	8	5	9
1.13	1.54	1.66	1.98

Discrete representation after local enhancement:

Vehicle type 0:

10	3	2	1	4	6
0.23	0.42	0.49	0.61	0.79	0.90

Vehicle type 1:

9	5	8	7
1.13	1.54	1.66	1.98

Continuous solution representation after local enhancement:

Customer ID	1	2	3	4	5	6	7	8	9	10
Position	0.61	0.49	0.42	0.79	1.54	0.90	1.98	1.66	1.13	0.23

Figure 7.4: The example illustrates the LES, where a solution for 10 customers and 2 vehicle types is subject to a local enhancement, and subsequently transformed back its representation in the continuous space. Note that within a the solution representation for a vehicle type, the array of customer sequences are changed, while the position arrays remains unchanged. Therefore, the LES changes the positions which customers are mapped to. The colors are used to highlight that for customer 4, its position is changed from .42 to .79, while it remains served by vehicle type 0 as seen by the integer part of the position number.

Chapter 8

The Combinatorial Journey-Generating Model

In order to improve the probability of obtaining quality solutions within a time limit which is applicable to real-life planning, we propose another heuristic approach, the combinatorial journey-generating model (CJGM), to solve the problem studied in this thesis. The CJGM is motivated by the idea of using mathematical programming models to improve performance of heuristic solution methods, commonly referred to as *matheuristics*. Matheuristics are successfully applied to solve VRPs in recent literature (Archetti and Speranza, 2014).

Matheuristics use characteristics of solutions obtained by exact methods to guide the heuristic search. Archetti et al. (2017) propose a matheuristic to solve an inventory routing problem (IRP), which share several properties with the problem studied in this thesis. The IRP differs in that each customer requires a certain inventory level in each period, accepting neither overstock nor stock-outs. Archetti et al. (2017) proceeds by first comparing journeys obtained from solving a relaxed exact journey-based formulation with the frequencies of journeys which are generated by solving a heuristic method, and secondly use the results to fix a set of variables when the journey-based formulation is re-solved.

The CJGM is based on a similar approach, iterating between two steps. First, it solves multiple heuristic methods, i.e. PHGAs (Section 6.3) and PABCs (Section 7.2). Second, the journeys from the best solutions with respect to fitness from each heuristic are extracted and sent to an exact MIP, i.e. the journey-based model (JBM), introduced in Section 8.3. The JBM solves the problem of finding the best journeys to form a complete solution to

optimality. In this manner, the JBM generates a solution which is composed of journeys found by different PHGAs and PABCs. This solution is used to guide the heuristic search in the next iteration in the CJGM.

This chapter is structured as follows: in Section 8.1, the CJGM is described in detail by means of an example of one iteration. We discuss the role of the exact JBM in Section 8.2, whereas a mathematical formulation of the JBM is formulated in Section 8.3.

8.1 Overview of the Combinatorial Journey-Generating Model

The first iteration of the CJGM is illustrated in Figure 8.1, and proceeds as follows. In step 0, a set of heuristics, composed of a number of PABCs (Algorithm 6) and PHGAs (Algorithm 5), are initialized with one global ODC^g each. Next, the solution methods are run for a given amount of time in step 1. At this stage, each heuristic has generated a set of journeys. Recall that both the PHGA and the PABC exploits a periodic problem decomposition, and thus search for ways to create and allocate journeys independently for each period. When a PHGA terminates, it holds one population for each period, where individuals in each population represent a solution. Each solution is a set of journeys for the particular period. When a PABC terminates, it holds a set of positions, where each position represents a solution, which is a set of journeys for a particular period. Subsequent to completion of the first phase in step 1, they each apply the OD-MIP (Section 5.7) which finds the optimal ODC for the identified best set of journeys. The algorithms are re-solved with the new ODC, generating a new sets of journeys.

In step 2, the set of all feasible journeys in the N^{sol} best solutions for each heuristic are selected based on fitness evaluations. This amounts to a total of $N^{PHGA} \times N^{SOL} \times |\mathcal{P}| + N^{PABC} \times N^{SOL} \times |\mathcal{P}|$ solutions, where $|\mathcal{P}|$ is the number of periods, and N^{PHGA} and N^{PABC} are the number of PHGAs and PABCs, respectively. The total number of journeys depends on the number of journeys in each of the selected solutions. The journeys are then passed to the JBM, which identifies the optimal subset to form a complete problem solution. Note that the JBM is allowed to combine journeys to form new solutions independent of which solution they were part of when generated in step 1. In addition to generating a new combination of journeys, the JBM identifies the optimal ODC for this particular set of journeys. The selected set of journeys and the new ODC are stored.

Selection of the N^{SOL} best solutions is based on the fitness evaluation procedures described in Section 5.4. Recall that the fitness score (Equation 5.13) of a feasible solution is equal to the objective value, whereas the biased fitness score (Equation 5.14) incorporates its contribution to solution diversity. When solutions are selected from PABCs, the regular

fitness measure is used as basis. However, in order to enhance the diversity of the journeys which are passed to the JBM, the biased fitness measure is applied when solutions from PHGAs are evaluated. In addition, the best solution with respect to regular fitness in each PHGA is included in N^{SOL}. By ensuring that the best solution with respect to the regular fitness measure from each PHGA are also sent to the JBM, the JBM is guaranteed to generate at least an equally good solution as any of the heuristics if enough run time is provided.

Finally, step 3 selects heuristics to have their ODC^g re-initialized. They are selected if either of the two scenarios described in Section 6.4 occurs, i.e. when insufficient solution improvement is obtained from the previous iteration. The selected heuristics are re-initialized in an arbitrarily order. One of the PABCs or PHGAs is re-initialized with the ODC generated by the JBM. If more than one heuristic is subject to re-initialization, they are initialized with new diversified ODCs generated according to the procedure described in Section 6.4. Step 1-3 complete one iteration of the CJGM, where each iteration is given T^{it} time. Step 1-3 is repeated until either a maximum number of iterations without improvement is reached, N^{it}, or the run time exceeds a given time limit, T^{max}.

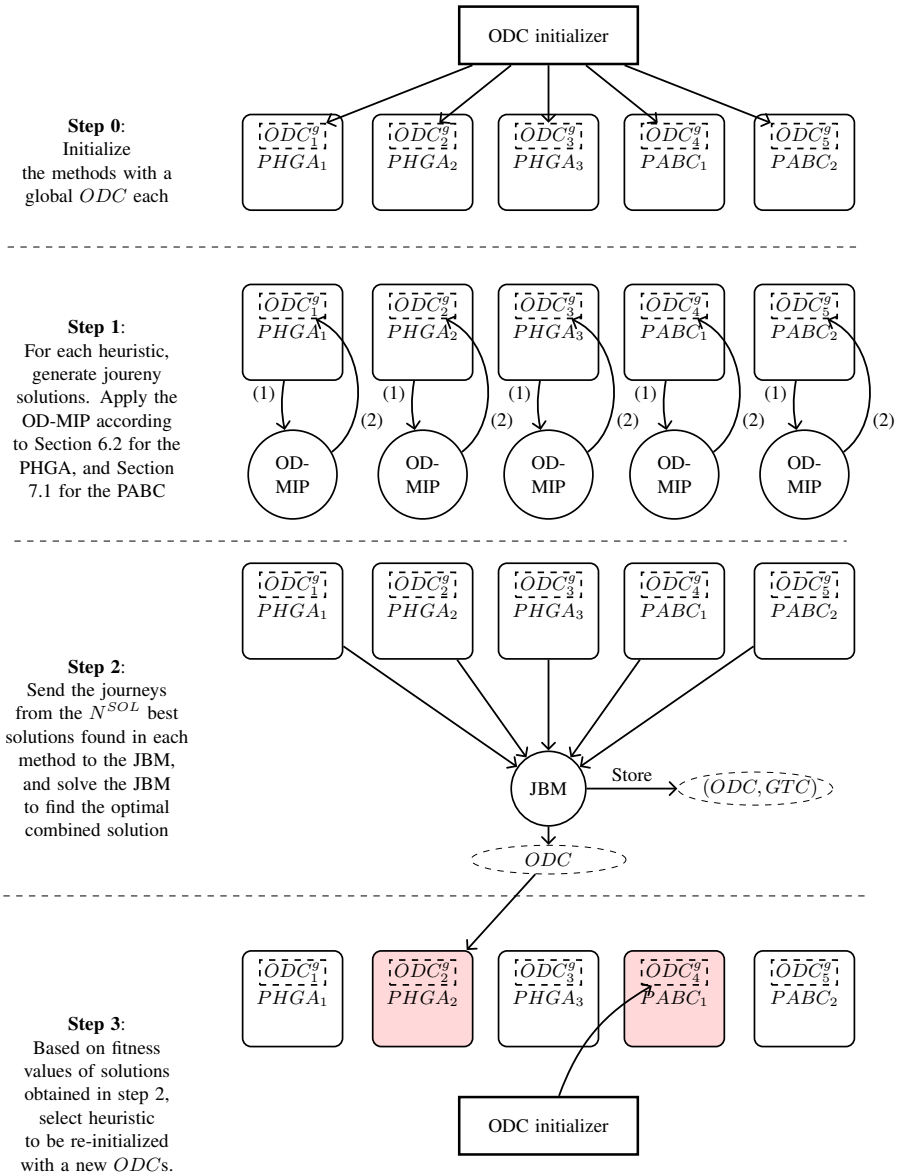


Figure 8.1: An example of one iteration of the CJGM, where the number of PHGAs and PABCs are set to 3 and 2, respectively.

8.2 Improving Heuristic Solutions with an Exact Journey-Based Model

As shown in step 2 in Figure 8.1, the CJGM exploits an exact JBM which finds the best combination of the journeys generated by PHGAs and PABCs in step 1, and simultaneously creates an optimal order distribution for these journeys. The purpose of using the JBM to guide the search in the CJGM is twofold. First, as the best solutions found by each PHGA and PABC are included in the subset of journeys which is passed to the JBM, solutions obtained by the JBM can never deteriorate those generated by any heuristic in step 1. Second, the JBM generates a new ODC which is optimal for the selected journeys. In step 3, this ODC replaces the ODC^g for one of the heuristic methods.

A mathematical formulation of the JBM is provided in Section 8.3. As concluded in Bakken et al. (2019), the JBM scales poorly with the number of journeys. When the JBM is used as an exact method for the entire problem studied in this thesis, as in Bakken et al. (2019), the number of journeys needed to ensure solution optimality grows exponentially with the number of customers. However, by using fitness values of heuristic solutions to extract the journeys which are most likely to be part of a quality solution, the JBM can be solved more efficiently and thus improve the heuristic search in the CJGM.

8.3 Mathematical formulation of the Journey-Based Model

In this section, we present a mathematical formulation of the JBM. A JBM to solve the PMTVRPTW problem was first introduced in Bakken et al. (2019). Modifications are made in order to fit the application to this thesis, as this formulation is more efficient to solve problems with a smaller number of journeys. The description extends the AFM described in Chapter 4, and OD-MIP described in Section 5.7.

8.3.1 Definition of new Sets, Indices, Variables, and Parameters

The following sets, variables, and parameters are defined. Note that in contrast to the AFM formulation, j does not represent a customer. For the JBM, j denotes a journey $j \in \mathcal{J}_{ph}$, where $h \in \mathcal{H}$ represents a vehicle type, and $p \in P$ represents a period.

Definition of Sets:

$\mathcal{J}_{ph}$	Set of journeys that can be completed for vehicle of type $h \in \mathcal{H}$ in period $p \in \mathcal{P}$
$\mathcal{T}_j$	Set of trips $\tau \in \mathcal{T}_j$ for the a journey $j \in \mathcal{J}_{ph}$
$\mathcal{T}_p^S$	Set of unique trips, i.e. sets of customers, for a period $p \in \mathcal{P}$
$\mathcal{N}_{phj\tau}$	Set of customers visited in period $p \in \mathcal{P}$ with vehicle type $h \in \mathcal{H}$, in journey $j \in \mathcal{J}_{ph}$, and trip $\tau \in \mathcal{T}_j$
$\overline{\mathcal{N}_p}$	Set of customers not visited in period $p \in \mathcal{P}$, equal to $\mathcal{N} \setminus \bigcup_{h \in \mathcal{H}} \bigcup_{j \in \mathcal{J}_{ph}} \bigcup_{\tau \in \mathcal{T}_j} \mathcal{N}_{phj\tau}$
$\mathcal{J}_{ph\tau}^S$	Set of journeys j for period $p \in \mathcal{P}$ and vehicle type $h \in \mathcal{H}$, which all include a trip where at least all customers $i \in \mathcal{N}_\tau$ are visited. More formally, all $j \in \mathcal{J}_{ph\tau}^S$ have a trip $\tau' \in \mathcal{T}_j$ where the set of customers $\mathcal{N}_{\tau'} \supseteq \mathcal{N}_\tau$

Definition of Parameters:

A_{iphj}	Binary parameter, 1 if customer $i \in \mathcal{N}$ is visited by vehicle type $h \in \mathcal{H}$ in period $p \in \mathcal{P}$ in journey $j \in \mathcal{J}_{ph}$, 0 otherwise
Q^{max}	Capacity of the larges vehicle type $h \in \mathcal{H}$.

Definition of Variables:

$$\gamma_{phj} = \begin{cases} 1 & \text{if journey } j \in \mathcal{J}_{ph} \text{ for vehicle type } h \in \mathcal{H} \text{ is completed in period } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

$$u_{pim} = \begin{cases} 1 & \text{if commodity } m \in \mathcal{M}_i \text{ is delivered to customer } i \in \mathcal{N} \text{ in period } p \in \mathcal{P} \\ 0 & \text{otherwise} \end{cases}$$

q_{pim}	Quantity of commodity $m \in \mathcal{M}_i$ delivered to customer $i \in \mathcal{N}$, in period $p \in \mathcal{P}$
q_p^O	Quantity of commodity distributed using overtime in period $p \in \mathcal{P}$

8.3.2 Mathematical Formulation

A mathematical formulation of the journey-based model is given below.

Objective Function

$$Minimize \quad \sum_{p\in P}\sum_{h\in H}\sum_{j\in\mathcal{J}_{ph}} C^J_{phj}\gamma_{phj} + \sum_{p\in P} C^O q^O_p \tag{8.1}$$

Compared to the AFM in Chapter 4, the main difference in the objective function is that arc costs and vehicle usage costs are replaced with a journey cost, which is the sum of the cost of the arcs traversed in the journey, and the cost of using a vehicle of type h.

JBM Specific Constraints:

$$\sum_{j\in\mathcal{J}_{ph}} \gamma_{phj} \leqslant |\mathcal{V}_h|, \quad h \in \mathcal{H}, p \in \mathcal{P} \tag{8.2}$$

$$\sum_{h\in\mathcal{H}}\sum_{j\in\mathcal{J}_{ph}} A_{iphj}\gamma_{phj} = I^p_i, \quad i \in \mathcal{N}, p \in \mathcal{P} \tag{8.3}$$

$$\sum_{i\in\mathcal{N}_\tau}\sum_{m\in\mathcal{M}_i} q^p_{im} + \sum_{h\in\mathcal{H}}\sum_{j\in\mathcal{J}^S_{ph\tau}} (\sum_{i\in\mathcal{N}_\tau} Q^{max} - Q_h)\gamma_{phj} \leqslant \sum_{i\in\mathcal{N}_\tau} Q^{max}, \quad p \in \mathcal{P}, \tau \in \mathcal{T}^S_p \tag{8.4}$$

Constraints (8.2) ensure that the number of journeys allocated to vehicles of vehicle type h is less or equal to the number of vehicles of that vehicle type. Constraints (8.3) enforce visits in correct periods for each customer. Constraints (8.4) limit the amount of commodities delivered on trip τ if journey j is selected, i.e. $\gamma_{phj} = 1$. Note that this formulation is efficient when the number of trip configurations, $\sum_{p\in\mathcal{P}} |\mathcal{T}^s_p|$, is relatively small and limited. For larger numbers of $\sum_{p\in\mathcal{P}} |\mathcal{T}^s_p|$, other formulations of the JBM are more suitable, as the number of constraints of type (8.4) grows exponentially with the number of trip configurations.

Constraints Adopted from the Order Distribution Mixed Integer Program:

$$\sum_{i\in\mathcal{N}}\sum_{m\in\mathcal{M}_i} q_{pim} \leqslant \overline{Q^O_p} + q^O_p, \quad p \in \mathcal{P} \tag{8.5}$$

$$q_{pmi} = Q_{im}^{ND} u_{pim}, \quad i \in \mathcal{N}, m \in \mathcal{M}_i^{ND}, p \in \mathcal{P} \tag{8.6}$$

$$\underline{Q_{im}^{D}} u_{pim} \leqslant q_{pim} \leqslant \overline{Q_{im}^{D}} u_{pim}, \quad i \in \mathcal{N}, m \in \mathcal{M}_i^{D}, p \in \mathcal{P} \tag{8.7}$$

$$\sum_{p \in \mathcal{P}} q_{pim} = Q_{im}^{D}, \quad i \in \mathcal{N}, m \in \mathcal{M}_i \tag{8.8}$$

$$\sum_{m \in \mathcal{M}_i^{ND}} u_{pim} \leqslant I_{pi}, \quad i \in \mathcal{N}, p \in \mathcal{P} \tag{8.9}$$

$$\sum_{p \in \mathcal{P}} u_{pim} = 1, \quad i \in \mathcal{N}, m \in \mathcal{M}_i^{ND} \tag{8.10}$$

$$\sum_{p \in \mathcal{P}} u_{pim} \geqslant U_{im}^{-}, \quad i \in \mathcal{N}, m \in \mathcal{M}_i^{D} \tag{8.11}$$

$$\sum_{p \in \mathcal{P}} u_{pim} \leqslant U_{im}^{+}, \quad i \in \mathcal{N}, m \in \mathcal{M}_i^{D} \tag{8.12}$$

$$u_{im}^{p} = 0, \quad p \in \mathcal{P}, i \in \overline{\mathcal{N}}_p, m \in \mathcal{M}_i^{D} \tag{8.13}$$

Constraints on Variables:

$$\gamma_{phj} \in \{0,1\}, \quad p \in \mathcal{P}, h \in \mathcal{H}, j \in \mathcal{J}_{ph} \tag{8.14}$$

$$u_{pim} \in \{0,1\}, \quad p \in \mathcal{P}, i \in \mathcal{N}, m \in \mathcal{M}_i \tag{8.15}$$

$$q_{pim} \geqslant 0, \quad p \in \mathcal{P}, i \in \mathcal{N}, m \in \mathcal{M}_i \tag{8.16}$$

$$q_{p}^{O} \geqslant 0, \quad p \in \mathcal{P} \tag{8.17}$$

$$\tag{8.18}$$

Chapter 9

Generation and Description of Problem Instances

The different solution methods proposed in this thesis are evaluated based on comparisons of test results on various instances in Chapter 10. This chapter gives an overview of the instances used. Section 9.1 describes the real-life data which instances are generated from. The different sets of instances and their applications in this thesis are described in Section 9.2.

9.1 Description of the Data used to Generate Instances

As D_1 and D_2 serve customers in different geographic regions, the data sets have different characteristics which might affect test results in computational study in Chapter 10. Therefore, a small set of descriptive statistics for the two data sets is provided in Table 9.1. Let N_D be the set of customers in data set D, and $d_{i,j}$ be the distance between customer i and j. Further, let $dist_D^{avg}$ represents the average distance between a customer and its closest neighbours in the data set D. The distance is calculated by Equation 9.1, where $\mathcal{N}(i)$ is the neighbourhood of customer i. The neighbourhood consists of its $\mathcal{N}(i) = h \cdot |N_D|$ closest customers in terms of distance between customer i and j, where $h = 0.4$ is the proportion of customers defined as nearest neighbours used in Section 5.6. The average quantity ordered per customer, q_D^{avg}, is calculated by simply adding the total order volume for each customer, and divide the sum with the number of customers in the data set.

$$dist_D^{avg} = \frac{\sum_{i \in N_D} \sum_{i \in \mathcal{N}(i)} d_{i,j}}{|N_D| \cdot |\mathcal{N}(i)|} \tag{9.1}$$

Observe from Table 9.1 that D_1 is composed of customers with almost a doubled average distance to their closest neighbours compared to D_2. In addition to larger distances, the average customer volume in D_1 is approximately 60% of the average total customer volume in D_2. These differences can provide valuable insight into the behaviour of the solution methods in the computational study in Chapter 10.

Table 9.1: Description of the two data sets, D_1 and D_2, which are used to generate instances. The metrics shown for each data set is the size, average distance between each customer and its neighbours, and average volume per costumer

Metric	D_1	D_2
Number of customers	75	118
$dist^{avg}$	0.662	0.404
q^{avg}	33.893	51.747

9.2 Description of the Test Instances

Test instances are divided into 3 test sets, each containing a number of instances which are generated to test different aspects of the methods. Each instance holds a subset of the data provided by ASKO. Instances vary in size and complexity, and are characterized by the data set they are generated from, the number of customers that are served, and the number of vehicles available. The *Instance ID* is named such that "01D1C10V5" is instance number 1 in the instance set, extracted from data set 1, having 10 customers, and 5 vehicles. Note that *instance size* is used when referring to the number of customers in a given instance.

When describing each instance, we also include the following two metrics which are calculated in advance of the instance generation: the average distance from a customer to its nearest neighbours, $dist^{avg}$, and the average volume ordered per customer, q^{avg}. The former is calculated according to an adaption of Equation 9.1, where the size of the number of customers in the data set N_D is replaced with the number of customers in the particular instance. The latter is calculated by taking the sum of the total volume ordered for each customer in the instance, and divide the sum with the number of customers in the instance. The metrics are included as they might provide valuable insight into behaviour

of the solution methods in the computational study in Chapter 10.

All instances are generated with a number of available vehicles which amounts to half of the number of customers. First, this coincides with real-life data provided by ASKO, where the size of the actual fleet used to serve customers is always less than half the number of the customers they serve. In addition, the number of vehicles available is determined based on a discussion of the impact the number of vehicles available has on the behaviour of the solution methods. If the number of available vehicles is too low, one might risk that instances are infeasible, as the total vehicle capacity available is less than the total customer demand. On the contrary, too many vehicles available increases the complexity of the problem. Thus, the final value is a trade-off between the probability of obtaining feasible solutions, and increased problem complexity. Note that this is the number of available vehicles, and does not need to coincide with the number of vehicles which are actually used in a solution.

In the following, we describe the test set of small-sized instances (9.2.1), medium-sized instances (9.2.2), and large-sized instances (9.2.3).

9.2.1 Small-Sized Test Instances

In order to compare the proposed heuristics with exact solution methods, 5 instances are generated and described in Table 9.2. When comparing solutions found by exact and heuristic methods on small instances (i.e. where exact methods find optimal solutions), a basis for performance evaluation of the heuristics is obtained. If a heuristic method is able to find close to optimal solutions for smaller instances, it might serve as an indication, but no guarantee, that it obtains quality solutions for larger instances as well.

In Bakken et al. (2019) it was concluded that exact methods struggle to find solutions for instances with more than 10 customers within a reasonable amount of time. Therefore, the selected instances are all composed of 10 customers. In order to test the method behaviour on instances with different characteristics in terms of average distances between customers and average ordered volume per customer, the instances are selected to have various values of $dist^{avg}$ and q^{avg}.

Table 9.2: Overview of small-sized instances, each consisting of 10 customers and 5 vehicles. Three of the instances are from D_1, and instances are from D_2. The average distance between a customer and its neighbours and average order volume is given.

Instance ID	Data Set	#Customers	#Vehicles	$dist^{avg}$	q^{avg}
01D1S57C10V5	1	10	5	0.714	35.754
02D1S97C10V5	1	10	5	1.223	29.195
03D1S80C10V5	1	10	5	0.647	42.969
04D2S89C10V5	2	10	5	0.420	55.253
05D2S01C10V5	2	10	5	0.412	66.129

9.2.2 Medium-Sized Test Instances

The heuristics proposed in this thesis have several parameters which must be tuned before tests can be conducted to evaluate their performance. A set of medium-sized instances are generated in order to tune parameters for all methods. The tuning process and the resulting parameter values are described in Chapter 10. Note that the instances are, subsequent to parameter tuning, used to evaluate and compare performance of the proposed heuristics.

The instances are described in Table 9.3. They have a diverse selection of values for $dist^{avg}$ and q^{avg} in order to avoid parameters which are limited to be suited for instances with certain characteristics.

Table 9.3: Overview of medium-sized instances, each consisting of 25 customers and 12 vehicles. Two instance set contain two instances from each data set. The average distance between a customer and its neighbours and average order volume is given.

Instance ID	Data Set	#Customers	#Vehicles	$dist^{avg}$	q^{avg}
01D1C25V12	1	25	12	1.002	28.205
02D1C25V12	1	25	12	0.7039	33.497
03D2C25V12	2	25	12	0.390	64.127
04D2C25V12	2	25	12	0.403	50.176

9.2.3 Large-Sized Test Instances to Compare the Heuristic Solution Methods

In order to compare the heuristic methods proposed in this thesis, we generate a set of real-sized instances. A description of the generated instances is provided in Table 9.4. The instances consist of subsets of a data set of either 50, 75, 100, or 115 customers. Note

that all instances of 100 and 115 customers are drawn from D_2, as D_1 only consist of 75 customers. As expected, $dist^{avg}$ and q^{avg} converge towards values for the entire data sets (Table 9.1) when the number of customers approach their full sizes.

Table 9.4: Overview of large-sized instances, containing instances of sizes from 50 to 115 customers, and 25 to 62 vehicles. The majority of the instances originate from D_2 as D_1 has total size of 75 customers. Average distance from a customer to its neighbours and average order volume is given.

Instance ID	Data Set	#Customers	#Vehicles	$dist^{avg}$	q^{avg}
01D1C50V25	1	50	25	0.701	31.911
02D1C50V25	1	50	25	0.625	33.479
03D2C50V25	2	50	25	0.383	48.252
04D2C50V25	2	50	25	0.387	55.572
05D1C75V37	1	75	32	0.662	33.893
06D2C75V37	2	75	32	0.428	53.447
07D2C75V37	2	75	32	0.395	50.480
08D2C75V37	2	75	32	0.395	52.212
09D2C100V50	2	100	50	0.418	52.275
10D2C100V50	2	100	50	0.421	53.569
11D2C100V50	2	100	50	0.417	50.767
12D2C100V50	2	100	50	0.401	49.859
13D2C115V62	2	115	62	0.399	52.591
14D2C115V62	2	115	62	0.410	51.400

Chapter 10

Computational Study

This chapter contains a computational study of the four solution methods proposed in this thesis: the HGA (Chapter 5), the MPHGA (Chapter 6), the MPABC (Chapter 7), and the CJGM (Chapter 8). All methods are implemented in Java 8 with SDK 11.0.6 and Gurobi version 8.1.1 is used as the commercial optimization solver. The computers that have conducted the tests is a rack of Dell PowerEdge R640 running on Linux with a 2 x Intel Xeon Gold 5115 with 20 cores at 3.2 GHz, 96 Gb of RAM and a 120GB SSD installed.

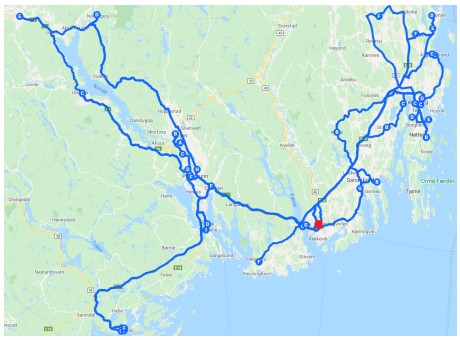
As the heuristics are non-deterministic, meaning that their behaviour can be different with the same input on different runs. Therefore, reported results for all methods during parameter tuning are for each tested parameter value reported as the average of 5 runs, i.e. 5 *samples*. Remaining tests are reported as the average of 10 samples. In total, 2900 samples are solved and used as basis of the analysis presented in this chapter.

In order to enhance the understanding of this chapter, Figure 10.1 illustrates routing solutions obtained by the CJGM for two different problem instances, both with 50 customers. Figure 10.1a and 10.1b show the solution for an instance with customers in Trøndelag (D_1), whereas Figure 10.1c and 10.1d show the solution for an instance with customers in Vestfold&Telemark (D_2). Nodes are customers, and trips are indicated by blue lines. Depot is represented by a red square.

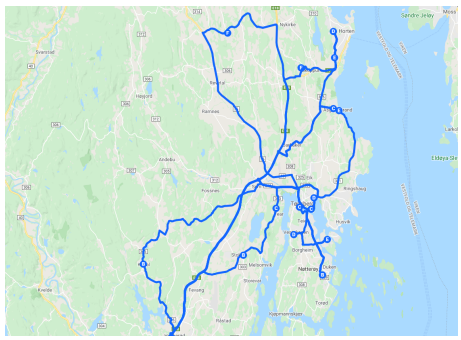
This chapter is structured as follows. Section 10.1 describes parameter values and how they were tuned. In Section 10.2, the heuristics are compared to solutions found by an exact solver on small-sized instances. A discussion and comparison of the behavior of the heuristics for medium and large-sized instances are given in Section 10.3. In Section 10.4, we study how different mechanisms in the CJGM contribute to method performance. Finally, Section 10.5 provides a study of solutions obtained by the CJGM for full-size

(a) Trøndelag

(b) Trondheim



(c) Vestfold&Telemark



(d) Tønsberg area

Figure 10.1: Two routing solutions obtained by the CJGM for Trøndelag (D_1) and Vestfold&Telemark (D_2) for a single period. Figure 10.1b and Figure 10.1d is zoomed in for the same solutions given in Figure 10.1a and Figure 10.1c respectively. Blue nodes represent customers and lines show trips. The location of the depot is marked by the red square. The example consists of 50 customers for each of the data sets.

instances, which discusses solution characteristics, and how they relate to the real-life problem faced by ASKO.

10.1 Parameter Tuning

As performance of metaheuristics in general varies with the value of its parameters, we have conducted a parameter tuning procedure in advance of the testing. Parameters should in general be tuned in order to balance complementary properties of the search algorithms, e.g. exploration and exploitation, and computational time and solution quality. In each of the four proposed solution methods, there are several parameters which must be adjusted in order to find the configuration which maximize their ability to find quality solutions

within a reasonable run time for various instances.

The Tuning Procedure

In general, for each of the four solution methods, parameter tuning is proceeded as follows. Initially, parameters which are calibrated in relevant literature are imported and assumed to be fixed in advance of the tuning process. Next, the remaining parameters are tuned in a sequential order. The test value range for a parameter is determined by computational limits and reasonable bounds. The order of which parameters are tuned is selected based on preliminary testing of their impact on method behaviour, i.e. how solutions are altered when parameter values are changes. Dependencies between parameters have also affected the order of which they have been tuned. For instance, parameters in the MPABC which affect how a bee updates its position are determined before the number of bees of each type are tuned.

Each parameter is tuned on the 4 instances dedicated to parameter testing described in Section 9.2. Within the value range, a set of values are selected as potential parameter values. All parameter values are based on 20 samples, 5 for each instance, and objective values and run times are reported. The average value for all parameter values are calculated for each instance, and are normalized across the parameter values tested. A complete overview of all results, i.e. objective values and run times, given as normalized values, is provided in Appendix A. Final values are selected based on the resulting objective values and run times.

In the following, we describe the parameters for the genetic algorithms (i.e. the HGA and the MPHGA), the MPABC, and the CJGM. Time limits are set to 10 minutes (600 seconds). All methods are subject to an early stopping criteria, forcing termination if the number of consecutive iterations without improvement N^{it} reaches 20. This number is determined based on preliminary testing.

10.1.1 Parameters in the Genetic Algorithms

The performance of genetic algorithms rely on the configuration of input parameter values (Vidal et al., 2012). As an attempt to identify a good configuration, Vidal et al. (2012) applied a *meta-calibration* approach, which has proven to perform well in calibration for other genetic algorithms. Their approach proceeds by applying an independent calibration phase for each of the VRP problem classes they studied, and generate a final set of parameters which can be applied independent of problem class. We assume that the parameter values adopt to both the HGA and the MPHGA, and consider these values as fixed when the remaining parameters are tuned. An overview of the imported parameters, their fixed

Table 10.1: Parameter values adopted from and calibrated by Vidal et al. (2012) used in the HGA, MPHGA, and CJGM.

Parameter	Selected Value	Description
μ	25	Population size (Algorithm 1)
el	0.4	Proportion of elite individuals (Section 5.4)
nc	0.2	Proportion of close individuals (Section 5.4)
p^{ed}	1.0	Probability for education (Algorithm 1)
p^{rep}	0.5	Probability for repair (Algorithm 1)
h^n	0.4	Neighbourhood size (Section 5.6)
ξ^{REF}	0.2	Reference proportion of feasible individuals (Section 5.11)

values, and a brief description of their purpose, is presented in Table 10.1. A detailed description of their usage areas are found in Vidal et al. (2012).

Table 10.2 provides an overview of the parameters which have been subject to a tuning process, with a brief description of the parameters, selected values, a list of the tested values, and which solution method they appear in. These values are subject to a tuning process based on two considerations: (1) preliminary testing showed that values given in literature are not applicable, or (2) no values are proposed in literature. We assume that the same configuration of shared parameters applies to both the HGA and the MPHGA, and therefore conduct tests for their common parameters with the HGA only. In the following, we provide a brief justification for how test results, i.e. run times and objective values, are used to select the final parameter values.

Vidal et al. (2012) emphasize that the best value for the number of offsprings generated, λ, may vary considerably between different VRP classes. Test results reveal that by increasing λ, objective values are reduced at the cost of increased run times. We consider objective values as more important than run times, as the latter in most cases are far from the maximum time limit. Therefore, the value of 80 is selected.

The value of heuristic dominance criteria, γ, represents a trade-off between obtained run time and the probability of obtaining an optimal way of combining trips into journeys which are assigned to vehicles in the adSplit-procedure (Section 5.3). By assessing test results, the best objective is obtained with $\gamma = 1$, and is selected, even though it is at the expense of more computational time. Observe that this makes the the labeling procedure in adSplit (Section 5.3) exact, as discussed in Section 18.

For the probability of applying trip optimization, p^{trip}, we know that the trip optimizer operator (5.8) modifies trips in a deterministic manner, and therefore makes individuals more similar. Thus, increasing p^{trip} results in less diversity, which in turn increases the

Table 10.2: Parameter values tuned for the Genetic Algorithms. All values are determined by values found in Appendix A.

Parameter	Value	Description	Tested Values	Model
λ	80	Offsprings generated (Algorithm 1, 5)	[40, 80, 120, 160, 200]	HGA, MPHGA
γ	1	Heuristic dominance criteria (Inequality 5.9)	[1, 2, 3, 4, 5]	HGA, MPHGA
p^{trip}	0.25	Probability of trip optimization (Algorithm 1, 5)	[0, 0.25, 0.5, 0.75, 1]	HGA, MPHGA
p^{mip}	0.75	Probability of ODC-MIP (Algorithm 1, 5)	[0, 0.25, 0.5, 0.75, 1]	HGA
$\|ODC^{new}\|$	12	Number of ODCs generated during re-initialization (Equation 6.1)	[3, 6, 9, 12, 15]	MPHGA, MPABC
k	2	Tournament size (Section 5.11)	[2, 3, 4, 5, 6]	HGA, MPHGA

probability of premature convergence (see Section 2.2.3). As test results provided marginal differences in run times, p^{trip} is set to 0.25 based on obtained objective values. For the probability of applying the OD-MIP, p^{mip}, an increased probability value strictly improves the objective value, but at the expense of increased run times. A reasonable computational efficiency is obtained for $p^{mip} = 0.75$.

The value of $|ODC^{new}|$ is set to 12, as the best results are obtained with respect to both run times and objective values. A larger value of $|ODC^{new}|$ generates more diverse order distributions each time a PHGA has its ODC re-initialized in the MPHGA. Therefore, a value of 12 implies that enhanced search diversity is preferred, which coincides with the discussion of the importance of diversity in Section 2.2.3. The same reasoning applies to why the optimal value of the tournament size k is 2, based on both objective values and run times. With $k = 2$, the selection procedure is identical to the binary tournament selection applied in Vidal et al. (2012). Among the tested values, $k = 2$ evaluate population diversity the most, as it assigns the highest probability of selecting poor individuals to survive to the next generation.

10.1.2 Parameters in the Multi-Periodic Artificial Bee Colony Algorithm

Table 10.3 provides an overview of the parameters which have been tuned for the MPABC, including their selected values. Each parameter has been tuned separately, and the order is determined based on dependencies. In particular, the optimal position update policy is found first, i.e. the parameters determining how employees and onlookers perform local search. Afterwards, the size of the bee colony is determined, followed by the parameters which affect the ODC-update policy. As the MPABC, to the extent of our knowledge, is not proposed to solve similar problems as the one studied in this thesis in literature, no

parameter values are adopted from previous work. In the following, a brief comment on the selected values for some of the parameters listed in Table 10.3 is given.

The reader must be aware that R^i, ρ^o, and N^{ODC} are multiples of the number of dimensions, i.e. number of customers, as the parameters must scale with the problem size. Results obtained for different values of ρ^o unambiguously indicate that it should be set to 0. The fact that a continuous space is used to represent solutions can explain why the random perturbation of the onlooker position should be removed. Small changes of a continuously represented solution can result in significant changes when it is mapped to a discrete representation. Even if the random adjustment by ρ^o only slightly moves the onlooker position in the continuous space, the discreet translation can be amplified.

As for the number of local enhancements n^{LES}, the best value is found to be 3. As the LES-operator cannot deteriorate the objective value, one would think that n^{LES} should be as large as possible. A likely explanation for why 3 is optimal according to the test results, is that LES is a computationally expensive operation, as it requires the adsplit-procedure to evaluate the quality of the changed solution. Therefore, this parameter represents a trade-off between the number of searches each MPABC can perform within given time limit, and the number of LES operations applied.

$N^{onlookers}$ and $N^{employees}$ have been tuned simultaneously, as they together represent a balance between search exploration and exploitation. A large number of employees will result in an increased amount of exploration of the entire solution space, while a large number of onlookers represents a high level of exploitation in the local solution space around employees. With limited computational power, a balance between these factors is paramount. Test results showed that the best balance is obtained by having 20 employees and 8 onlookers per employee, i.e. 160 onlookers.

As discussed in Section 7.2, ξ^{REF} is tested for a range of suitable values. The MPABC is also tested without penalty adjustment, indicated by $-$. Even though larger values of ξ^{REF} report better performance in terms of run times, no adaptive penalty adjustment yields the best results in terms of objective values. Therefore, adaptive penalty adjustment is not applied for the MPABC.

Table 10.3: Parameter values for MPABC. All values are based on results given in the tables reported in Appendix A

Parameter	Value	Description	Tested Values
k^n	1	Range for weight ω^n of random neighbour position (Equation 7.4)	[0.25, 0.50, 0.75, 1]
k^{gb}	0.75	Range for weight ω^{gb} of global best position (Equation 7.4)	[0, 0.25, 0.50, 0.75]
R^i	0.4	Dimensions changed in position update (Equation 7.4)	[0.1, 0.4, 0.7, 1]
ρ^o	0	Onlooker random adjustment (Equation 7.2)	[0, 0.1, 0.2, 0.3, 0.4]
n^{LES}	3	Number of enhancements (Algorithm 7)	[0, 1, 2, 3, 4, 5, 6]
λ^{gb}	1.3	Allowed range from best solution for trials reset (Equation 7.3)	[1.0, 1.1, 1.2, 1.3, 1.4, 1.5]
$N^{onlookers}$	8	Number of onlookers per employee (Algorithm 6)	[3, 5, 8, 10]
$N^{employees}$	20	Number of employees (Algorithm 6)	[5, 10, 15, 20]
N^{scouts}	50	Number of scouts (Algorithm 6)	[25, 50, 75, 100]
N^{ODC}	4	Number of iterations per ODC (Algorithm 6)	[2, 4, 6, 8, 10]
ξ^{REF}	-	Reference proportion of feasible individuals ODC (Algorithm 6). (-) denotes no adjustment	[0.7, 0.75, 0.80, 0.85, 0.90, -]

10.1.3 Parameters in the Combinatorial Journey-Generating Model

For the CJGM, the parameter which balances the number of PHGAs and PABCs solved in parallel, (step 1 in Figure 8.1) must be tuned. Note that the MPHGA and MPABC are tuned in advance, such that the optimal balance of PHGAs and PABCs is decided when they both are calibrated sufficiently. Figure 10.2 illustrates the objective values (y-axis) obtained when the number of PHGAs ranges from 0 to 6 (x-axis), i.e. from 0% to 100%.

Results indicate that the set of heuristics solved in the CJGM should be composed of both PHGAs and PABCs. As similar solution methods are more likely to create journeys with similar characteristics, the probability of obtaining a larger set of journeys is increased if the CJGM exploits a mix of PABCs and PHGAs. When the JBM-solver in the CJGM (Section 8.3) is provided with more journeys to construct a solution from, the probability of of obtaining better solution quality is increased. Based on results shown in Figure 10.2, the number of PHGAs is set to 4 out of 6, i.e. 66.67%.

The N^{sol} parameter is not subject to tuning, as more journeys in the JBM will, as

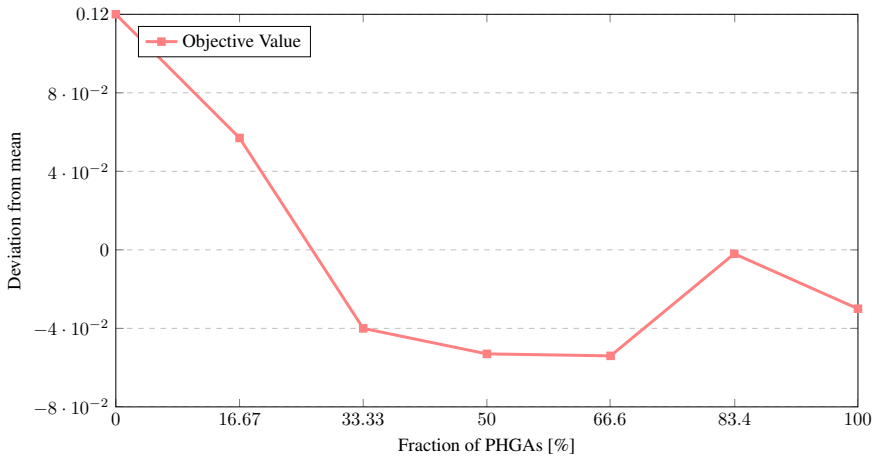


Figure 10.2: Deviation from mean when tuning the fraction of PHGAs in the CJGM. Full table can be found in Appendix A.

already stated, increase the chance of obtaining quality solutions. However, if too many journeys are passed to the JBM (8.3), the solution time will drastically increase. This will reduce the time dedicated to PHGAs and PABCs in the CJGM to find good solutions. Therefore, $N^{sol} = 3$, and the time limit for the JBM is set to 3 minutes (180 seconds). This gives the JBM a total of 18 complete solutions of journeys to choose from when creating a solution. The time for one iteration of solving PABCs and PHGAs in the MPHGA, MPABC, and the CJGM, T^{it}, is increased proportionally with the number of customers for the instance solved. This is motivated by a the increase in computational time needed in the PABC and PHGA operators, as instances grow in size.

10.2 Comparing Exact and Heuristic Methods on Small-sized Instances

In this section, we present and discuss results obtained by solving instances in Table 9.2 with the exact arc-flow model (AFM) (Chapter 4), and the four heuristics proposed in this thesis: the HGA, MPHGA, MPABC, and CJGM.

The time limit for the AFM is set to 8 hours (28800 seconds). The heuristic methods are solved with a time limit of 30 minutes (1800 seconds), and an early stopping criteria if no improved solution is found after 20 iterations. Recall that in the HGA, one iteration is considered solving Algorithm 1. In the MPHGA, step 1-3 in Figure 6.1 amount to one iteration. The same applies to the MPABC, which adapts the framework of the MPHGA

as described in Section 7.1. For the CJGM, one iteration is composed of step 1-3 in Figure 8.1.

Objective values for all methods on the 5 small-sized instances in Table 9.2 are reported in Table 10.4. The mixed integer programming (MIP) *gap* is reported for each instance solved by the AFM. The MIP-gap is by Gurobi calculated as $|ObjBound - ObjVal|/|ObjVal|$, where $ObjVal$ is the MIP objective bound, and $ObjVal$ is the incumbent solution objective. Gurobi considers a solution as optimal whenever the gap is below 0.01%. For all instances where the obtained MIP-gap is non-zero, the model was terminated by the time limit. For the heuristics, best and average values are reported as the deviation (in %) from the best primal solution obtained by the AFM. Objective values for the heuristics are given as % of the exact solutions, where the best found solution and the average solution are reported.

Table 10.4: Results showing how the different solution methods perform compared to the best objective solution found by an exact solver. All mean entries are based on 10 runs, and the best instance is given in the best-columns. The percentages given represents the methods average deviation from the objective value found be the exact solver.

Instance ID	**AFM**		**HGA**		**MPHGA**		**MPABC**		**CJGM**	
	Obj	Gap (%)	Mean (%)	Best (%)	Mean (%)	Best (%)	Mean (%)	Best (%)	Mean (%)	Best (%)
01D1C10V5	6,005	33.2	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
02D1C10V5	6,834	0.00	0.07	0.00	0.00	0.00	0.00	0.00	0.00	0.00
03D1C10V5	6,056	26.5	0.03	0.00	- 0.17	- 0.21	- 0.17	- 0.21	- 0.18	- 0.21
04D2C10V5	5,936	46.9	0.71	0.08	0.04	0.02	0.04	0.02	0.02	0.00
05D2C10V5	6,509	7.21	9.71	0.80	0.60	0.28	0.26	0.02	0.36	0.17
Average			**2.09**	**0.18**	**0.10**	**0.02**	**0.03**	**- 0.03**	**0.04**	**- 0.01**

Table 10.5 shows average run times and standard deviations (in seconds) for all heuristics on the 5 small-sized instances in Table 9.2. For all instances except from 02D1C10V5, the non-zero MIP-gap indicates that the time out limit was reached. Run times are therefore not reported in the table. For 02D1C10V5, 58 minutes (3480 seconds) were used to find the solution.

The results show that the AFM is unable to find solutions with no optimality gap within the given time limit on all instances, except for 02D1C10V5. Note that test instances dedicated to find benchmarks by the AFM, were generated based on results obtained in prior work by Bakken et al. (2019), where an average optimality gap of instances with 10 customers were close to 0.2%. However, larger gaps for similar sized instances reported in Table 10.4 might be a consequence of differences in the data which test instances are

Table 10.5: The table shows the mean and standard deviation for the run time for each solution method, given in seconds.

Instance ID	HGA		MPHGA		MPABC		CJGM	
	Mean (s)	SD (s)	Mean (s)	SD (s)	Mean (s)	SD (s)	Mean (s)	SD (s)
01D1C10V5	123	32.8	485	119	232	35.0	467	301
02D1C10V5	125	13.8	391	9.97	216	20.0	324	319
03D1C10V5	172	42.9	573	163	246	46.8	1014	663
04D2C10V5	103	45.2	731	207	409	82.9	1000	780
05D2C10V5	122	73.4	1099	1037	280	89.6	862	577
Average	**125**	**41.6**	**656**	**156**	**290**	**54.9**	**733**	**528**

generated from. In Bakken et al. (2019), a data generator was constructed to create test instances, whereas this thesis solves for instances generated from real-life data. A brief comparison reveals that customers in instances generated from real-life data have more frequent visits with larger volumes, and therefore tend to be more difficult to solve. In addition, the problem studied in this thesis is solved for one more period than in Bakken et al. (2019), in order to include Saturday as a delivery day.

With a non-zero MIP-gap, it is not possible to determine with certainty whether the AFM finds optimal solutions. However, the HGA, MPHGA, MPABC, and CJGM in general tend to deviate only slightly from the objective obtained by the AFM. Except for the HGA on instance 05D2C10V12, all methods obtain a mean deviation of less than 1% from the AFM solution on all instances. The heuristic methods are in fact all able to find the same solution as the AFM for instance 01D1C10V5 and 02D1C10V5. The latter is particularly important when evaluating solution quality, as the AFM guarantees optimality by having a gap of 0.00%.

For instance 03D1C10V5, the MPHGA, MPABC, and CJGM performs better in terms of objective values than the exact method. As the optimality gap for the solution found by the AFM is non-zero and thus provides no optimality guarantee, these results are plausible. However, an interesting observation is that the heuristic methods find different best solutions. One possible explanation is that the characteristics of instance 03D1C10V5, which relative to the other instances has average values for volume and distances between customers (Table 9.2), enable multiple solutions to be considered of similar quality.

The MPABC performs better than the CJGM with respect to both average solution, and best found solution on instance 03D2C10V5 and instance 05D2C10V5. Recall that the MPABC and the MPHGA are build on solving multiple PHGAs or PABCs, respectively. The CJGM combines journeys generated by both PHGAs and PABCs. Therefore, the

CJGM should find solutions which are at least as good as those obtained by the PABCs and PHGAs it is composed of. One possible explanation for this behaviour is that the PHGA and PABC are both non-deterministic, and therefore not guaranteed to find the same solutions in each run. Another possible explanation is that there are differences in how the methods distribute their run time. The MPHGA and MPABC are able to execute more iterations than the CJGM, as they do not need to solve the exact JBM (Section 8.3) to find new order allocations between iterations.

The CJGM reports a larger average run time in Table 10.5 than both the MPABC and the MPHGA. A larger run time is expected, as it combines solutions found by PHGAs and PABCs when solving an exact journey-based model (Section 8.3). Table 10.5 also shows that the CJGM reports the largest standard deviation with respect to run time. When compared to the HGA, MPHGA and, MPABC, the CJGM is less dependent on obtaining quality journeys in the same run, as it is allowed to combine journeys from multiple runs in one iterations (i.e. PHGAs and PABCs) to construct a solution. Therefore, the probability of finding a good solution in the first iteration is larger for the CJGM when compared to the remaining methods. However, as the length of one iteration in the CJGM often exceeds those in the MPHGA and MPABC due to exploitation of the exact JBM, the total run time grows rapidly when multiple iterations are needed.

10.3 Comparing Methods on Medium and Large-Sized Instances

This section presents and discusses results obtained on test instances in Table 9.3 and 9.4, described in Chapter 9. All methods are tested on 18 instances in total, and detailed results are reported in Table B.1 in the Appendix. The methods are terminated if either of the following two criteria is fulfilled: completing a maximum of 20 iterations without improvement, or reaching a time limit of 30 minutes (1800 seconds). Note that most samples where terminated because the time limit was reached, except for instances of 25 customers for the HGA. This might indicate that further improvement could have been obtained if the methods were given more run time.

A brief comment on how parallelization is exploited in each of the solution methods is provided. Recall that the HGA solves the entire problem for all periods simultaneously, whereas the PHGA and the PABC solve the problem for each period separately. As the MPHGA and MPABC solve multiple PHGAs and PABCs in parallel, respectively, while the CJGM solves a combination of both, the HGA is the only method which does not exploit any level of parallelization. Therefore, the HGA uses less of the available computing power than the other three methods. When doing comparisons, the reader should be aware

that the HGA is subject to this disadvantage. The maximum number of iterations without improvements, N^{it}, is set equal to 20 for all methods. As suggested by Vidal et al. (2012), the number of generations without improvement before diversification of the population, N^{div}, is set to $0.4N^{it}$ for the HGA.

10.3.1 Run Time Analysis on Medium-Sized Instances

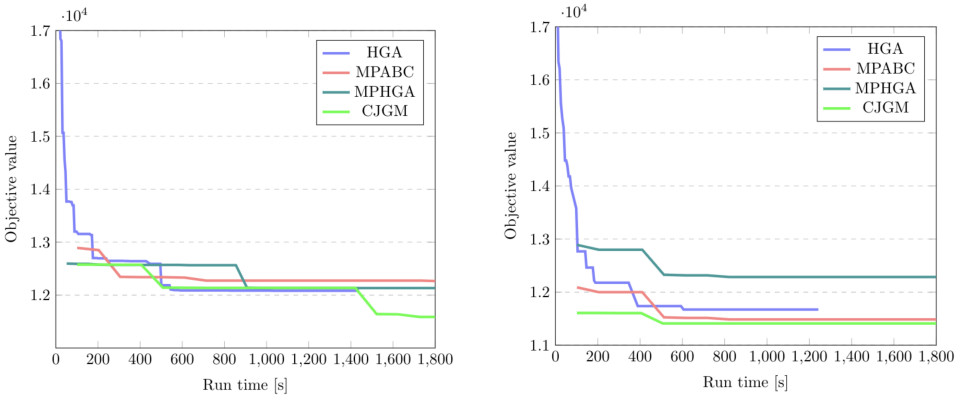
In Figure 10.3, development of objective values (y-axis) found by all methods are illustrated for four instances with 25 customers. The x-axis represents run time (in seconds). In all examples, the HGA terminates before T^{max} is reached, which means that the algorithm has performed N^{it} iterations without improvement. The other methods are terminated due to the time limit.

For all instances except 01D1C25V12, the CJGM consistently finds quality solutions within the first 500 seconds. The MPABC and the MPHGA seem to need more iterations to obtain solutions with similar quality. As the CJGM is allowed to combine solutions found by various heuristics, it is reasonable that it has a larger probability of finding quality solutions in earlier iterations. As shown in Figure 10.3, the MPABC often finds a good initial solution compared to its final best solution. However, it seems to struggle finding solution improvements, which is particularly clear for instance 03D2C25V12.

In general, the HGA tends to initially find solutions which are worse than those found by the other methods. A plausible explanation is that the periodic decomposition exploited in all algorithms except from the HGA makes it easier to obtain good quality solutions in early iterations, as improvements in one period rather than in all periods is more easily adopted to update the current best solution. Without periodic decomposition, improvement of an individual in the HGA in one period may be offset by a degeneration in another period, making the improvement less likely to be included in solutions which are carried to the next generation.

The HGA seems to most rapidly update its current best found solution. This is likely resulting from the fact that it more often evaluates new solutions, compared to the periodic algorithms, which only evaluate solutions when each periodic problem is solved in one iteration. The sudden drops for the HGA probably indicate that a new best solution is found, where the fleet size is reduced by one vehicle. Another interesting observation is that the HGA obtains significantly worse solutions compared to the other methods on instances from data set 2 (D_2), i.e. 03D1C25V12 and 04D2C25V12. This tendency is also supported by Table B.1, where the HGA reports poor solutions on instances generated from D_2 compared to the other methods, which is not observed for instances from D_1.

Plausible explanations for this behaviour can be found by comparing characteristics of



(a) Instance 01D1C25V12

(b) Instance 02D1C25V12

(c) Instance 03D2C25V12

(d) Instance 04D2C25V12

Figure 10.3: Illustration of the run time development of all algorithms for 4 different instances, all consisting of 25 customers. Figure 10.3a and Figure 10.3b are from data set D_1 and Figure 10.3c and Figure 10.3d are from D_2. The objective value for a given time is reported.

the different data sets. As described in Chapter 9, D_2 has a lower average driving distance between customers, but higher average quantity delivered per customer when compared to D_1. Recall that solutions in the HGA are assigned one order allocation each, which, as described in Section 5.5.1, 5.7.2, and 5.10, are continuously evaluated and updated during the search. In contrast, order allocations are less frequently updated for the methods exploiting a periodic decomposition. Consequentially, the HGA might suffer from having the flexibility in moving orders around more frequently, which can explain its reduced performance on instances from D_2. Quality journey solutions often need several iterations to be found, and changing the order distribution frequently might be a distortion rather than an advantage when more volume must be delivered.

10.3.2 Scalability of the Methods

In order for a method to be valuable as decision tool for ASKO when they schedule journeys ahead of a planning period, its ability to obtain quality solutions for real-size instances is important. Thus, the methods are compared on how they scale with an increasing number of customers. Figure 10.4 shows the average deviation (in %) from the best objective found across all methods, on medium and large-sized instances for the HGA, MPHGA, MPABC, and CJGM. A detailed explanation of how values are calculated is provided in the following. First, the average for each instance by any model is calculated. Secondly, all values are normalized based on all averages from every method for each customer size. Lastly, the deviation from the best found solution across all methods is reported for each method. These values are reported in Figure 10.4. As the CJGM on average finds the best solution for all instances, the average deviation from the average CJGM solution for all instance sizes for the HGA, MPHGA, and the MPABC are reported in Table 10.6.

Observe that solutions obtained by the MPHGA are approaching those reported by the CJGM as the deviation for the MPABC grows with the number of customers. This indicates that the exact JBM exploited in the CJGM selects journeys obtained by PHGAs rather than PABCs when constructing solutions for larger instances. These observations are reasonable, as the PABC probably generates solutions which are inferior compared to the PHGA for large instance sizes based on its poor scaling abilities of the MPABC when the number of customers exceeds 50, as shown in Figure 10.4. Table 10.6 confirms these observations, where the MPABC deviation is quadrupled from instances with 50 to 100 customers.

A possible explanation is found by assessing the scalability of the local search operators applied in the MPABC and the genetic algorithms (the HGA and the MPHGA). Recall that the genetic algorithms exploit a local search operator denoted *education* (5.6), which is applied for all solutions with a given probability $p^{ed} = 1$ in each iteration. In the MPABC, the only operator which is greedy during the search, i.e. guarantees solution improvements, is the local enhancement scheme (LES) (Section 7.2.4). However, the LES applies the adSplit-procedure (5.3) to evaluate improvements. As discussed in Section 5.3.3, adSplit is a computational expensive procedure. For this reason, design choices in the MPABC are made such that the LES is applied for a limited number of the explored solutions in each iteration. Note that differences between the MPHGA and the MPABC are less substantial on smaller instances, as adSplit grows exponentially in run time with the number of customers.

Figure 10.4 also shows that the HGA finds poor solutions on medium-sized instances compared to all other methods. It reports a solution which is more than three times the second worst performing method on instances with 25 customers. Also, its deviation is

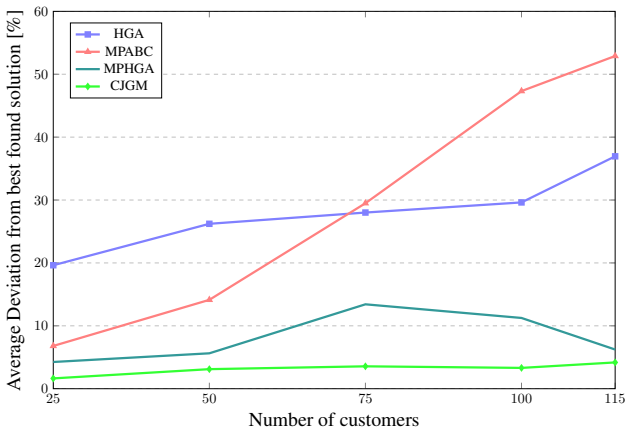


Figure 10.4: Comparison of deviation from the objective of best obtained solution for the instance for all methods and the different instance sizes. The objective deviation is given in (%) of higher objective value than the best solution.

strictly growing with the number of customers. In contrast, the MPHGA finds solutions close to the best objective obtained across all methods for medium-sized instances, and in fact reports lower deviation from 100 to 115 customer instances.

As the the MPHGA and the HGA are differentiated by the periodic problem decomposition, results might indicate that decomposition contributes to improved ability of finding quality solutions. Recall that by decomposing the entire problem into smaller problems for each period, the search for solutions is structurally altered, as changes are evaluated on a periodic level rather than for all periods as one. Without decomposition, solution changes for a period can be offset by poor solutions in other periods. Therefore, results shown in Figure 10.4 indicate that better solutions can be obtained when each period is allowed to optimize journey schedules independently.

Table 10.6: Deviation, given in %, from the average value obtained by the CJGM for all customer sizes for all solution method.

Instance Size	HGA (%)	MPHGA (%)	MPABC (%)
C25	17.31	2.52	5.01
C50	22.49	2.46	10.67
C75	23.76	9.86	25.54
C100	25.44	7.77	42.66
C115	31.49	1.96	46.77
Average	24.10	4.92	26.13

10.3.3 Stability of the Methods

To assess the stability of the algorithms, the coefficient of variation (CV) is calculated and averaged for instances with the same number of customers. CV (in %) for each method on a given instance size, is calculated by dividing the standard deviation of its objective value from all samples by their mean. The lower CV, the more stable the method is. Note that CV does not provide any insight into solution quality. It is simply a measure of how consistent a method is in obtaining solutions within the same value range.

Table 10.7 displays the CV (in %) for all instance sizes and all methods. The CJGM is the most stable method on average, having the lowest CV for all instance sizes except from 50 customers. An overall tendency is that the CV is increased when the instance size grows. This trend is most significant for the MPABC. A possible explanation is that when new areas in the solution space are explored in the MPABC, these areas are randomly selected. Thus, there are no guarantee that areas are not re-visited during the search. The CJGM and the HGA seem to be the most resistant against instability when instances grow in size. Overall, the stability performance of all methods are considered good, as average CVs are all below 3%.

Table 10.7: Average coefficient of variation in % for each solution method based on instance size. Average across all methods for given instance size is given, as well as average CV for each method

Instance Size	HGA (%)	MPHGA (%)	MPABC (%)	CJGM (%)	Average (%)
C10	0.16	0.01	0.01	0.01	0.05
C25	3.14	3.27	2.14	1.33	2.47
C50	4.18	3.29	1.75	1.96	2.80
C75	4.29	4.88	2.38	1.74	3.32
C100	2.01	3.69	6.12	1.91	3.43
C115	2.16	3.47	4.99	2.19	3.20
Average	**2.79**	**2.72**	**2.32**	**1.45**	**2.32**

10.4 A Detailed Study of Mechanisms in the Combinatorial Journey-Generating Model

Based on results discussed in Section 10.2 and 10.3, the CJGM appears superior to the other methods in terms of ability to find quality solutions within early iterations, scalability on real-size instances, and solution stability. Therefore, this section is dedicated to

investigate detailed characteristics of the CJGM. Two topics are investigated: the origin of the journeys which are selected in the final solution, and the contribution to solution improvement which is provided by the exact JBM (Section 8.3).

Figure 10.5 shows where the journeys, which are used in the final solution obtained by the CJGM, originate from. The x-axis shows the percentage of solutions obtained from the PHGA, PABC or both, and the y-axis denotes different instance sizes. Recall that the CJGM solves 6 heuristics, i.e. $\frac{2}{3}$ PHGAs and $\frac{1}{3}$ PABCs, determined by parameter tuning in Section 10.1. The percentage reported in Figure 10.5 represent the average values among all samples for each customer size. In some cases, both methods find the same journeys used in the final solution. This is denoted as *PABC&PHGA*.

The majority of journeys in the CJGM solution originate from PHGAs for all instance sized, as can be seen in Figure 10.5. Particularly, when the number of customers exceed 75, 97% of all journeys originate from PHGAs. These observations match the results shown in Figure 10.4, where the MPABC scales poor with the number of customers. On small-sized instances, the JBM exploits a combination of solutions obtained by either only PHGAs and PABCs, or identical journeys generated by both methods. This indicates that the PABC and the PHGA find solutions with different characteristics, which both prove to be of high quality for smaller instances. As the PABC is able to generate journeys which the PHGA cannot find, the PABC has a positive contribution on the CJGM solution. This motivates a further improvement of the PABC to improve its scalability for larger instances, in order for more diverse quality journeys are passed to the JBM. On the other hand, the PHGA can diversify its search so that it finds the journeys which only the PABC finds. A larger variety of solutions to select from when constructing a final solution improves the search diversity for the CJGM.

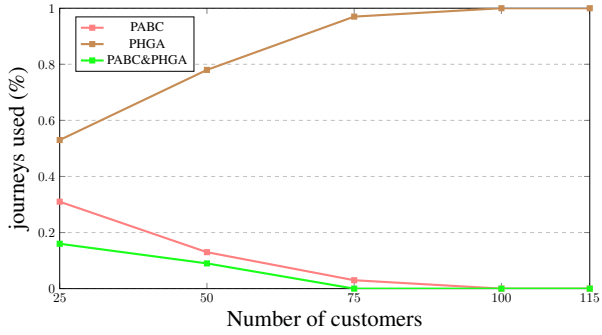


Figure 10.5: Development of where journeys originate from in the CJGM. PABC&PHGA denotes journeys which were found in both a PHGA and a PABC.

The box plot in Figure 10.6 shows the improvement (in %) of solutions found by the heuristics, which is obtained when solving the JBM with these solutions as input. The plot

is created based on objective values for all samples for different customer sizes, based on a total of 1850 data-points. Data-points are values extracted from each iteration of the CJGM when the JBM is applied (see Figure 8 in Section 8.2). The improvement is measured by $(obj^{SOL} - obj^{JBM})/obj^{SOL})$, where obj^{SOL} is the objective of the best solution found by PABCs and PHGAs during the previous iteration, and obj^{JBM} denotes the objective value for the solution obtained by the JBM. The percentage improvement thus shows how much the fitness is reduced from the best solution found in that iteration.

Figure 10.6 shows that the impact of the JBM strictly increases with growth in instance size. For instances with 115 customers, the average contribution to solution improvement caused by the JBM is approximately 4%, corresponding to a 30% increase from its contribution for instances with 25 customers. Recall that Section 10.3 showed that the MPHGA and the MPABC, and thus the PHGA and PABC, are both able to find quality solutions for smaller instances. Therefore, when solutions obtained by the heuristics are already good, the JBM on average obtains less improvement when combining those solutions. This is shown by the large box for instances of 25 customers, where the 25th percentile is close to 0. Note that in some cases, the JBM has a large impact on these instances, where it contributes with a 10% improvement. This might be a consequence of a poorly initialized order distribution in advance of the first iteration. It is important to be aware that solutions found by the JBM are not guaranteed to be of high quality, even if they are improved from the input journeys generated by PHGAs and PABCs. If the input journeys are of poor quality, the combination of them will probably remain poor.

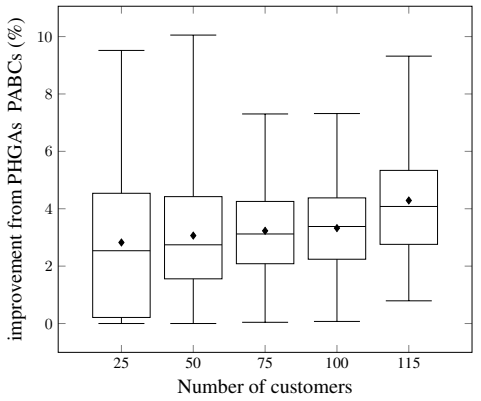


Figure 10.6: Box plot of the improvement the JBM yields to the solutions found by the PHGAs and PABCs in (%). Top and bottom whisker correspond to lowest and highest percentage. The bottom and top of the box corresponds to the 25th and 75th percentile, meaning 50% of all data points lies within the box. The line in the box correspond to the median, and the diamond marks the mean for each customer size.

10.5 A Detailed Study of Full-Sized Solutions

In this section, solutions obtained by the CJGM for full-size instances with a time limit of 2 hours (7200 seconds), i.e customer size 75 for D_1 and 118 for D_2, are studied. The differences in characteristics of solutions obtained for the two data sets are highlighted, and their metrics are compared to the real-life problem setting faced by ASKO.

Both solutions have no overtime at the depot, which implies that the CJGM has found an allocation of orders such that the overtime limit is not exceeded in any period. In general, the best solutions tend to have a minor cost of overtime, often a cost of zero, as the OD-MIP and JBM often manage to allocated the orders efficiently. In terms of driving cost and vehicle usage cost, the objective value in these solutions are composed of roughly 10% traveling cost, and 90% vehicle usage cost. When the CJGM is used as decision tool in real-life planning, the user should carefully consider where the costs are actually allocated: overtime at the depot, vehicle driving time, or fixed costs related to vehicle usage. Skewed cost distribution can make the solution favor minimizing the fleet rather than minimizing driving costs.

For the solutions found by the CJGM, all vehicles are of the smallest vehicle type. These vehicles have shown to be most cost efficient for these problem instances. In addition, the inclusion of time windows in the problem is undoubtedly in favor of small vehicles, as time windows already restrict how many customers a vehicle can serve in sequence before returning to the depot.

Table 10.8 describes how the periodic solutions for each data set differ. Observe that the number of vehicles used in each period are on average 7.8 and 14.6 for data set D_1 and D_2, respectively. These values are lower than expected, as ASKO usually operates a fleet of roughly 30 vehicles in Trøndelag. Three plausible explanations for this behavior are provided: first, an optimal allocation of order throughout the planning period may reduce the number of vehicles needed. Second, the conversion from quantities to volume might be too optimistic, in addition to missing orders for external suppliers, which are not provided. Third, the customers in the problem data are all grocery stores, while ASKO also supplies other customers, such as restaurants and canteens.

Note that all values are lower for period 6 compared to the other periods in Table 10.8. This is expected as period 6 represents a Saturday, which has less demand due to reduced workforce both at the warehouse, and at the grocery stores. In addition, as customers need to stock up before the weekend, and restock afterwards, period 1 and 5 have the highest number of visits. An interesting observation is that an increased number of visits does not lead to a higher number of trips in these periods. Instead, the number of trips are higher for period 2-4, in D_2. As the order quantities are equally distributed throughout period 1-5, with a lower quantity targeted for period 6, the trips for period 2-4 have a larger volume

Table 10.8: Periodic behavior for D_1 and D_2.

Metric	Period 1	Period 2	Period 3	Period 4	Period 5	Period 6	Average
Data set 1							
Vehicles used	9	8	9	9	9	3	7.8
Trips	15	15	14	14	14	4	12.7
Customers visited	69	59	60	61	71	21	57.8
Data set 2							
Vehicles used	19	14	16	14	17	8	14.6
Trips	35	37	36	36	35	8	31.2
Customers visited	117	106	113	105	114	64	103.2

per delivery. The vehicle capacity can be a limiting factor, meaning that more trips are used in order to meet customer demand. Therefore, in periods with many customer visits and lower quantities delivered per customer, fewer trips are used as each trip visits more customers than in periods with fewer customer visits. In periods with fewer customer visits, the vehicle capacity is more restricting, as the volume per delivery is larger. The same tendency is not present in D_1, as the commodity volume per customer is on average lower compared to D_2, which can be seen in Table 9.1.

Table 10.9 shows the distribution of the number of trips that visit a given number of customers for each solution provided by the CJGM. The solution for D_2 visits on average a lower number of customers than the solution for D_1. This is expected, as D_2 has a larger average volume of orders delivered per customers, increasing the need to return to the depot for loading. Table 10.10 shows the distribution of how many trips which are included in a journey in each solution. The solution for D_2 is composed of journeys with a larger average number of trips. To summarize, it is a clear tendency that the solution for D_1 has few but rather longer trips per journey, while the solution for D_2 has shorter trips, but more trips per journey.

Table 10.9: Distribution of trips visiting a given number of customers. No trips have more than 11 customers visited. The weighted average is the average number of customers visited for all trips

Number of trips visiting a given number of customers												
#Customers	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**Weighted Average**
#Trips for D_1	5	5	13	16	12	12	10	2	0	0	1	4.57
#Trips for D_2	21	46	45	36	24	6	2	6	1	0	0	3.31

Table 10.10: Distribution of number of trips used in a journey. No journey have more than 4 trips. The weighted average is the average number of trips for all journeys.

Number of journeys with a given number of trips					
#Journeys	**1**	**2**	**3**	**4**	**Weighted Average**
#Journeys for D_1	26	13	8	0	1.61
#Journeys for D_1	17	40	25	3	2.16

Table 10.11 displays aggregated information of solutions for D_1 and D_2. An interesting result is that the average idle time per journey, i.e. the time a vehicle waits at the customer site before the start of its time window, is almost twice as large for D_2 as for D_1. The solution for D_2 is considered more robust, as deviations in traveling time and unloading times can be offset by additional waiting time at the customer. The 20.0% journeys which are planned with no idle time in the solution for D_1 are more sensitive to deviations, as all deliveries must be completed according to schedule, in order to deliver goods on time. However, much waiting time may be inefficient, especially if deviations from schedules seldom happen.

Note that the average length of a journey and its standard deviation is higher for the D_1 solution, compared to the solution for D_2. This coincides with the fact that D_1 is consisting of several customers concentrated in Trondheim near the depot, while the other customers in D_1 are widely spread throughout Trøndelag. Large variations in journey duration therefore arise naturally. In contrast, Vestfold&Telemark consists of several larger cities which are located with a more even distance from the depot, and is in general less sparsely populated. The average number of periods a vehicle is used is almost equal for both solutions.

ASKO states that a normal filling level in a vehicle is between 35% to 40%, and that they have an overall goal of 45%. Recall that the filling level is measured as the ratio between the space in the vehicle which consists of commodities, and the total space in the vehicle. ASKO considers a filling level of 55% and above as good. In the solutions for D_1 and D_2, a filling level of 79.6% and 77.4% are obtained, which are substantially higher than the filling levels reported by ASKO. The filling levels obtained by the CJGM can be a result of simultaneously optimizing both journeys and order allocations, which differs from how ASKO schedules their weekly operations today.

Table 10.11: Metrics for the two full-sized solutions of D_1 and D_2. Values are aggregated as averages for each data set.

Metric description	D_1	D_2
Average idle time per journey	35 min	68 min
Percentage of journeys with no idle time	20.0%	9.7%
Average journey length (minutes)	255 min	201 min
Standard deviation of journey length (minutes)	156 min	83 min
Average driving time per trip (minutes)	160 min	109 min
Standard deviation of driving time per trip (minutes)	134 min	48 min
Average number of periods a vehicle is used	4.2	4.4
Average filling level of a vehicle	79.6%	77.4%

Chapter 11

Concluding Remarks

This thesis has studied the periodic multi-trip vehicle routing problem with time windows (PMTVRPTW), incompatible commodities, and a heterogeneous fleet. The problem arises when ASKO schedules routes and order allocations, which today are planned sequentially, to serve their customers.

Four different heuristic solution methods are developed, studied, and tested on instances generated from real-life data. For small-sized instances, all methods find solutions with minor deviations from the objective reported by an exact method. For medium and large-size instances, the combinatorial journey-generating model (CJGM) performs superior to the other methods, obtaining solutions with the best average objective values for all instance sizes. The CJGM is also the most stable method, as it reports the lowest average coefficient of variation. The exact journey-based model (JBM) exploited in the CJGM is shown to significantly improve solutions obtained by heuristics (periodic artificial bee colony algorithms and periodic hybrid genetic algorithms) for all instance sizes. The average contribution of the JBM increases with larger instance sizes. For 115 customers, a 4% average improvement is reported, corresponding to a 30% increase from its improvement on instances with 25 customers. The JBM is also shown to benefit from being provided with a diverse set of journey solutions.

For other types of the PMTVRPTW, there might be several variables connecting the planning periods, rather than just how commodity quantities are allocated. This is likely to increase the complexity of the problem solved by the exact method in the CJGM, such that run time needed to find solutions with sufficiently low optimality gaps exceeds what is considered to be applicable in real-life planning. To solve these problems, the multi-periodic hybrid genetic algorithm (MPHGA), the multi-periodic artificial bee colony (MPABC) al-

gorithm, and the hybrid genetic algorithm (HGA) should be considered.

Among the three methods, the MPHGA shows the best performance, and deviates on average only 2% from the CJGM solutions for the largest instances. The MPABC tends to find better initial solutions than the MPHGA for medium-sized instances. However, it is not able to find solutions with the same quality as the MPHGA when the number of customers exceeds 50. Two possible explanations are discussed: the MPABC exploits less information from search history than the MPHGA, and applies a less efficient greedy improvement operator. The HGA finds better solutions than the MPABC for larger instances. However, it is unable to consistently obtain quality solutions for small and medium-sized instances, especially when the volume ordered per customer grows.

Different approaches to simultaneously optimize journeys and order distributions are studied in this thesis. The HGA creates and assigns journeys, and updates the way orders are assigned to periods, for all periods in the planning horizon together. The remaining methods instead fix an initial order assignment, which is used to find journey solutions for each period independently. Iteratively, the fixed order assignment is updated, and new journeys are created and assigned to fit the new way order quantities are allocated to periods.

The former approach more frequently updates the way orders are allocated to periods during the search, whereas the latter dedicates more run time to optimize journeys which fit given order allocations. For the problem studied in this thesis, the methods applying the latter approach are shown to find quality solutions with fewer iterations. This property is particularly valuable when considering real-life implications for ASKO. Their planning process is a dynamic activity, where changes in input data occur frequently within a time frame of minutes. Therefore, decision tools must provide quality solutions in a short amount of time, without the need for complex calibration procedures.

For PMTVRPTW variants, the feasibility of order allocations might be more dependent on the way journeys are created and assigned to vehicles. For example, if there are compatibility restrictions between commodities and vehicles, finding order allocations which have feasible journey solutions can be difficult. For these problems, fixing order allocations and subsequently search for feasible journeys is inefficient. Instead, the approach used in the HGA might be more suitable, as it explores a larger set of order allocations during the search.

This thesis has shown that for practical routing problems where order assignments are treated as decision variables rather than fixed, simultaneous optimization of both journeys and order allocations should be considered. Solutions obtained by the CJGM on real-size instances perform well on metrics used by ASKO to quantify solution quality. This indicates that decision-makers at ASKO and other companies facing related practical problems can benefit from using solution methods developed in this thesis.

Chapter 12

Future Research

The solution methods developed in this thesis are subject to aspects which provide interesting areas for future research. For instance, the proposed methods can be adapted to solve other practical problems with similar characteristics. The PMTVRPTW can be extended with attributes which often arise in real-world routing problems, e.g. multiple depots, site-dependencies, or dynamic customer orders. If stock-level demands are incorporated, the inventory routing problem arises. As real-life routing problems often rely on parameters which are stochastic, e.g. travel times and order volumes, incorporating stochastic elements opens an interesting field of research. Robust solutions may be of interest, as customer demand can deviate from scheduled order volume during the planning horizon.

The PMTVRPTW can also be adapted to include multiple objectives. For instance, time windows are often negotiable for customers. For many distribution networks in practice, delayed or early customer visits can be desirable if costs are reduced. Generating a Pareto front of solutions can provide decision-makers valuable problem insight and decision support.

Another area of research is to improve the performance of the proposed solution methods. We have shown that the CJGM benefits from the ability to select from a diverse set of journeys when constructing the final solution with an exact solver. For instance, the PABC and the PHGA can be further developed. The local search mechanism in the PABC can be improved, as the local enhancement scheme is the only greedy operator which guarantees improved solutions. This operator scales poorly with an increasing number of customers, as it applies the computationally expensive adSplit-procedure whenever a solution is evaluated. For instance, the PABC can benefit from adopting a structure similar to the education-procedure (Section 5.6), which is a greedy operator that does not make use

of the adSplit. Scalability of the PABC can also be enhanced by developing mechanisms which better exploit search history to guide the choice of new search areas in the solution space.

To generate more diverse solutions from the PHGA, another decomposition scheme could be exploited to create subproblems with other properties. For instance, the geometric time window decomposition proposed by Vidal et al. (2013b) could be applied.

Besides studying improvements of the current heuristics exploited in the CJGM, it would be interesting to investigate other methods to efficiently generate quality journeys, e.g. by heuristic column generation, or tabu search, as proposed by several authors for problems with similar characteristics (Taillard, 1999, Archetti et al., 2017). Finally, the CJGM can also be extended to further enhance feedback from the exact solver to modify solutions found by the heuristics. Solutions found by the exact method may contain information which indicates what characterizes quality solutions. For instance, their characteristics can be used to update fitness evaluations of those generated by heuristics in subsequent iterations, or prevent identical journeys from being generated.

Bibliography

Agrawal, P., Kaur, H., and Bhardwaj, D. (2012). Enhanced bee colony algorithm for solving travelling salesperson problem. *International Journal of Control Theory and Computer Modelling*, 2(4).

Archetti, C., Boland, N., and Grazia Speranza, M. (2017). A matheuristic for the multivehicle inventory routing problem. *INFORMS Journal on Computing*, 29(3):377–387.

Archetti, C. and Speranza, M. G. (2014). A survey on matheuristics for routing problems. *EURO Journal on Computational Optimization*, 2:223–246.

ASKO Norge AS (2020). Asko norway - about us. `https://asko.no/en/about-us/`. Retrieved January 11, 2020.

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

Bakken, F. E., Glimsdal, B., and Tvinnereim, L. K. (2019). Exact methods for a periodic multi-trip vehicle routing problem in the food distribution industry. NTNU Project Report, 2019.

Banati, H. and Bajaj, M. (2011). Fire fly based feature selection approach. *International Journal of Computer Science Issues*, 8:473–479.

Baykasoglu, A., Ozbakir, L., and Tapkan, P. (2007). Artificial bee colony algorithm and its application to generalized assignment problem. *Swarm Intelligence: Focus on Ant and particle swarm optimization*, 1.

Beltrami, E. J. and Bodin, L. D. (1974). Networks and vehicle routing for municipal waste collection. *Networks*, 4(1):65–94.

Bent, R. and Van Hentenryck, P. (2010). Spatial, temporal, and hybrid decompositions for large-scale vehicle routing with time windows. In Cohen, D., editor, *Principles and Practice of Constraint Programming – CP 2010*, pages 99–113, Berlin, Heidelberg. Springer Berlin Heidelberg.

Campbell, A. M. and Wilson, J. H. (2014). Forty years of periodic vehicle routing. *Networks*, 63(1):2–15.

Cattaruzza, D., Absi, N., and Feillet, D. (2016a). The multi-trip vehicle routing problem with time windows and release dates. *Transportation Science*, 50(2):676–693.

Cattaruzza, D., Absi, N., and Feillet, D. (2016b). Vehicle routing problems with multiple trips. *4OR*, 14(3):223–259.

Cattaruzza, D., Absi, N., Feillet, D., and Vidal, T. (August 2014a). A memetic algorithm for the multi trip vehicle routing problem. *European Journal of Operational Research*, 236(3):833 – 848. Vehicle Routing and Distribution Logistics.

Cattaruzza, D., Absi, N., Feillet, D., and Vigo, D. (November 2014b). An iterated local search for the multi commodity multi trip vehicle routing problem with time windows. *Computers Operations Research*, 51:257–267.

Chen, H., Li, S., and Tang, Z. (2011). Hybrid gravitational search algorithm with random-key encoding scheme combined with simulated annealing. *IJCSNS*, 11.

Cordeau, G, L., A, M., and Correspondence (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, 52:928–936.

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

Crainic, T. G. (2008). *Parallel Solution Methods for Vehicle Routing Problems*, pages 171–198. Springer US, Boston, MA.

Crainic, T. G., Crisan, G. C., Gendreau, M., Lahrichi, N., and Rei, W. (2009). Multi-thread integrative cooperative optimization for rich combinatorial problems. In *2009 IEEE International Symposium on Parallel Distributed Processing*, pages 1–8.

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

Desaulniers, G., Madsen, O. B., and Ropke, S. (2014). Chapter 5: The vehicle routing problem with time windows. In *Vehicle Routing: Problems, Methods, and Applications, Second Edition*, pages 119–159. SIAM.

El-Sherbeny, N. A. (2010). Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University - Science*, 22(3):123 – 131.

Elshaer, R. and Awad, H. (2020). A taxonomic review of metaheuristic algorithms for solving the vehicle routing problem and its variants. *Computers Industrial Engineering*, 140:106242.

Fleischmann, B. (1990). The vehicle routing problem with multiple use of vehicles. *Fachbereich Wirtschaftswissenschaften, Universität Hamburg*.

François, V., Arda, Y., Crama, Y., and Laporte, G. (2016). Large neighborhood search for multi-trip vehicle routing. *European Journal of Operational Research*, 255(2):422 – 441.

Gaudioso, M. and Paletta, G. (1992). A heuristic for the periodic vehicle routing problem. *Transportation Science*, 26(2):86–92.

Gendreau, M., Hertz, A., and Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):1276–1290.

Glover, F. (1989). Tabu search—part i. *ORSA Journal on Computing*, 1(3):190–206.

Golden, B. L., Wasil, E. A., Kelly, J. P., and Chao, I.-M. (1998). *The Impact of Metaheuristics on Solving the Vehicle Routing Problem: Algorithms, Problem Sets, and Computational Results*, pages 33–56. Springer US, Boston, MA.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control and Artificial Intelligence*. University of Michigan Press, Ann Arbor, MI.

Ibaraki, T., Imahori, S., Nonobe, K., Sobue, K., Uno, T., and Yagiura, M. (2008). An iterated local search algorithm for the vehicle routing problem with convex time penalty functions. *Discrete Applied Mathematics*, 156(11):2050 – 2069. In Memory of Leonid Khachiyan (1952 - 2005).

Iqbal, S., Kaykobad, M., and Rahman, M. S. (2015). Solving the multi-objective vehicle routing problem with soft time windows with the help of bees. *Swarm and Evolutionary Computation*, 24:50 – 64.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization, technical report - tr06. *Technical Report, Erciyes University*.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95-International Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE.

Krause, J., Cordeiro, J., Parpinelli, R. S., and Lopes, H. S. (2013). 7 - a survey of swarm algorithms applied to discrete optimization problems. In Yang, X.-S., Cui, Z., Xiao, R., Gandomi, A. H., and Karamanoglu, M., editors, *Swarm Intelligence and Bio-Inspired Computation*, pages 169 – 191. Elsevier, Oxford.

Laporte, G. (2009). Fifty years of vehicle routing. *Transportation Science*, 43:408–416.

Mancini, S. (2016). A real-life multi depot multi period vehicle routing problem with a heterogeneous fleet: Formulation and adaptive large neighborhood search based matheuristic. *Transportation Research Part C: Emerging Technologies*, 70:100 – 112.

Matos, A. C. and Oliveira, R. C. (2004). An experimental study of the ant colony system for the period vehicle routing problem. In *Ant Colony Optimization and Swarm Intelligence*, pages 286–293. Springer Berlin Heidelberg.

Mingozzi, A., Roberti, R., and Toth, P. (2013). An exact algorithm for the multitrip vehicle routing problem. *INFORMS Journal on Computing*, 25(2):193–207.

Mladenović, N. and Hansen, P. (1997). Variable neighborhood search. *Computers Operations Research*, 24(11):1097 – 1100.

Mor, A. and Speranza, M. G. (2020). Vehicle routing problems over time: a survey. *4OR*, pages 1–21.

Mouthuy, S., Massen, F., Deville, Y., and Van Hentenryck, P. (2015). A multistage very large-scale neighborhood search for the vehicle routing problem with soft time windows. *Transportation Science*, 49(2):223–238.

Nagata, Y., Bräysy, O., and Dullaert, W. (2010). A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows. *Computers Operations Research*, 37(4):724 – 737.

Norgesgruppen AS. (2019). Annual Report for Norgesgruppen AS, 2019. `https://www.norgesgruppen.no/globalassets/norgesgruppen_in_english/ng_annual-and-sustainability-report-2019.pdf/`. Retrieved Mars 6, 2020.

Ostertag, A. (2008). *Decomposition strategies for large scale multi depot vehicle routing problems*. PhD thesis, uniwien.

Paradiso, R., Roberti, R., Laganá, D., and Dullaert, W. (2020). An exact solution framework for multitrip vehicle-routing problems with time windows. *Operations Research*, 68(1):180–198.

Prescott-Gagnon, E., Desaulniers, G., and Rousseau, L.-M. (2009). A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4):190–204.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers Operations Research*, 31(12):1985 – 2002.

Rizzoli, A.-E., Montemanni, R., Lucibello, E., and Gambardella, L. M. (2007). Ant colony optimization for real-world vehicle routing problems. *Swarm Intelligence*, 1:135–151.

Shaw, P. (1998). Using constraint programming and local search methods to solve vehicle routing problems. In Maher, M. and Puget, J.-F., editors, *Principles and Practice of Constraint Programming — CP98*, pages 417–431, Berlin, Heidelberg. Springer Berlin Heidelberg.

Solomon, M. M. (1984). *Vehicle routing and scheduling with time window constraints: Models and algorithms*. PhD thesis, University of Pennsylvania, Pa.

Solomon, M. M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265.

Szeto, W., Wu, Y., and Ho, S. C. (2011). An artificial bee colony algorithm for the capacitated vehicle routing problem. *European Journal of Operational Research*, 215(1):126 – 135.

Taillard, E. D. (1999). A heuristic column generation method for the heterogeneous fleet vrp. *RAIRO - Operations Research - Recherche Opérationnelle*, 33(1):1–14.

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

Toth, P. and Vigo, D. (2002). *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics.

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

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

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013b). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers Operations Research*, 40(1):475 – 489.

Wassan, N., Wassan, N., Nagy, G., and Salhi, S. (2017). The multiple trip vehicle routing problem with backhauls: Formulation and a two-level variable neighbourhood search. *Computers Operations Research*, 78:454 – 467.

Yu, B., Yang, Z.-Z., and Yao, B. (2009). An improved ant colony optimization for vehicle routing problem. *European Journal of Operational Research*, 196(1):171 – 176.

Zhen, L., Ma, C., Wang, K., Xiao, L., and Zhang, W. (2020). Multi-depot multi-trip vehicle routing problem with time windows and release dates. *Transportation Research Part E: Logistics and Transportation Review*, 135:101866.

Appendices

Appendix A

Detailed Results From Parameter Tuning

The numerical values obtained during the parameter tuning section are given below. Each entry is the mean based on 5 runs, normalized for all parameter values for given instance. The aggregated columns shows the mean for all instances for a given parameter value, serving as the values on which decisions have been made. Parameter selection is determined based on a discussion of both objective values and run times.

Table A.1: Offsprings generated (Algorithm 1, 5)

λ	**01C25V12T**		**02C25V12T**		**03C25V12VT**		**04C25V12VT**		**Aggregated**	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
40	1.026	0.343	1.03	0.319	1.084	0.208	0.999	0.375	**1.035**	**0.311**
80	0.995	0.747	0.982	0.645	1.026	0.577	1.009	0.621	**1.003**	**0.648**
120	0.987	0.908	1.002	1.056	0.989	1.002	1.006	0.939	**0.996**	**0.976**
160	0.988	1.256	0.985	1.038	0.973	1.525	1.002	1.022	**0.987**	**1.211**
200	1.005	1.746	1.002	1.942	0.928	1.687	0.983	2.042	**0.979**	**1.854**

Table A.2: Probability of trip optimization (Algorithm 1, 5)

p^{trip}	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
0	1.001	0.923	1.009	0.877	1.024	0.804	1.01	0.769	**1.011**	**0.844**
0.25	0.969	1.203	0.988	0.871	0.962	0.921	0.974	1.023	**0.973**	**1.005**
0.5	0.998	1.009	1.022	0.811	1.035	1.11	1.018	1.055	**1.018**	**0.996**
0.75	1.016	0.857	0.981	1.212	0.943	1.134	0.976	1.181	**0.979**	**1.096**
1	1.016	1.009	1.001	1.228	1.036	1.03	1.022	0.972	**1.019**	**1.06**

Table A.3: Probability of ODC-MIP (Algorithm 1, 5)

p^{mip}	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
0	1.044	0.188	1.172	0.219	1.342	0.244	1.27	0.282	**1.207**	**0.233**
0.25	1.014	0.631	1.007	0.538	0.953	0.575	0.976	0.509	**0.988**	**0.563**
0.50	1.001	0.926	0.962	1.147	0.91	0.921	0.935	0.883	**0.952**	**0.969**
0.75	0.989	1.266	0.938	1.202	0.92	1.209	0.911	1.465	**0.94**	**1.286**
1	0.952	1.989	0.921	1.894	0.875	2.051	0.908	1.861	**0.914**	**1.948**

Table A.4: Heuristic dominance criteria (Inequality 5.9)

γ	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
1	1.003	1.016	0.988	1.11	0.991	1.129	0.991	1.168	**0.993**	**1.106**
2	1.001	1.155	1.004	1.14	0.991	1.122	1.002	0.902	**1**	**1.08**
3	1.002	0.943	1.015	1.095	1	0.967	1	1.091	**1.004**	**1.024**
4	1	1.035	1.002	0.776	0.993	0.978	0.998	0.817	**0.998**	**0.902**
5	0.994	0.851	0.992	0.88	1.025	0.805	1.009	1.022	**1.005**	**0.889**

Table A.5: Tournament size (Section 5.11)

k	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
2	0.987	0.836	0.993	0.817	1.003	0.833	0.989	1.148	**0.993**	**0.909**
3	1.009	1.17	0.996	0.971	1.005	1.021	1.015	1.21	**1.006**	**1.093**
4	1.009	1.072	1.021	1.198	0.995	1.138	0.992	0.75	**1.004**	**1.039**
5	0.987	0.782	1.001	0.963	1.006	1.09	1.007	0.811	**1**	**0.911**
6	1.009	1.14	0.988	1.051	0.991	0.919	0.997	1.081	**0.996**	**1.048**

Table A.6: Number of $newODCs$ during re-initialization (Equation 6.1)

N^{ODC}	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
3	1.009	1.03	0.968	1.008	0.997	1.002	1.024	0.996	**0.999**	**1.009**
6	0.986	1.015	1.019	0.998	1.024	1.023	1.047	1.001	**1.019**	**1.009**
9	0.987	0.984	0.981	1.02	0.984	0.985	1.021	0.996	**0.993**	**0.996**
12	1.017	0.992	0.949	0.975	1.015	0.999	0.989	1.001	**0.992**	**0.992**
15	1.002	0.978	1.084	0.999	0.979	0.992	0.919	1.005	**0.996**	**0.994**

Table A.7: Range for weight ω^n of random neighbour position (Equation 7.4)

k^n	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
0.25	1.096	0.996	1.053	1	1.051	0.984	1.02	1.023	**1.055**	**1.001**
0.50	1.05	1.081	1.007	0.996	1.015	0.98	1.007	0.961	**1.02**	**1.004**
0.75	0.963	0.938	0.975	1.005	0.965	1.032	0.987	1.054	**0.973**	**1.007**
1.00	0.891	0.985	0.965	0.999	0.968	1.004	0.986	0.962	**0.953**	**0.988**

Table A.8: Range for weight ω^{gb} of global best position (Equation 7.4)

k^{gb}	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
0.00	1.051	1.082	1.051	0.926	1.029	1.001	1.03	0.991	**1.04**	**1**
0.25	1.041	0.925	1.02	1.044	1.015	1.038	1.007	1.001	**1.021**	**1.002**
0.50	1.005	0.993	0.986	1.045	0.99	0.972	0.982	0.99	**0.991**	**1**
0.75	0.904	1	0.942	0.984	0.966	0.989	0.982	1.019	**0.948**	**0.998**

Table A.9: Dimensions changed in position update (Equation 7.4)

R^i	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
0.1	0.99	0.972	0.993	0.96	1.051	0.947	1.019	1.03	**1.013**	**0.977**
0.4	1.01	1.087	0.995	0.965	0.993	1.028	0.984	0.963	**0.995**	**1.011**
0.7	1.006	0.946	1.002	0.998	0.987	0.988	0.986	1.022	**0.995**	**0.988**
1.0	0.995	0.995	1.009	1.077	0.969	1.038	1.011	0.985	**0.996**	**1.024**

Table A.10: Onlooker random adjustment (Equation 7.2)

ρ^o	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
0.0	0.886	1.025	0.951	1.058	0.952	0.966	0.967	1.038	**0.939**	**1.022**
0.1	0.998	0.924	1.021	0.975	1.018	0.967	1	0.95	**1.009**	**0.954**
0.2	1.028	1.009	1.024	0.937	1.004	0.984	1.012	1.024	**1.017**	**0.988**
0.3	1.049	0.984	0.999	1.027	1.006	1.042	1.007	0.936	**1.015**	**0.997**
0.4	1.039	1.057	1.004	1.004	1.02	1.042	1.013	1.052	**1.019**	**1.039**

Table A.11: Allowed range from best solution for trials reset (Equation 7.3)

λ^{gb}	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
1.0	1.011	0.941	1.007	0.938	1.015	1.024	1	1.068	**1.008**	**0.993**
1.1	1.001	1.064	1	1.041	0.992	0.955	1.005	0.934	**0.999**	**0.998**
1.2	0.991	1.065	1.002	1.012	0.999	1.03	0.988	1.104	**0.995**	**1.053**
1.3	0.992	0.945	0.998	0.933	0.995	1.021	1	1.027	**0.996**	**0.982**
1.4	1.006	0.863	1.003	0.987	1.007	0.927	1.001	0.923	**1.004**	**0.925**
1.5	1.01	1.064	0.997	1.027	1.007	1.067	1.007	1.012	**1.005**	**1.042**

Table A.12: Number of onlookers per employee (Algorithm 6)

$N^{onlookers}$	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
3	1.026	0.971	0.999	0.996	1.002	1.019	1.006	0.981	**1.008**	**0.992**
5	0.992	1.065	1.004	1.011	1.015	0.983	1.005	0.989	**1.004**	**1.012**
8	0.986	0.986	0.992	0.971	0.986	0.992	0.993	1.02	**0.989**	**0.992**
10	0.996	0.978	1.005	1.022	0.997	1.007	0.996	1.01	**0.999**	**1.004**

Table A.13: Number of employees (Algorithm 6)

$N^{employees}$	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
5	1.052	0.94	1.046	0.94	1.084	0.934	1.041	0.967	**1.056**	**0.945**
10	0.991	0.965	0.995	1.065	1.013	1.011	0.996	1.024	**0.999**	**1.016**
15	0.99	1.068	0.982	1.02	0.964	1.024	0.98	1.024	**0.979**	**1.034**
20	0.967	1.026	0.977	0.975	0.939	1.031	0.982	0.985	**0.966**	**1.004**

Table A.14: Number of scouts (Algorithm 6)

N^{scouts}	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
25	0.994	0.969	1.006	1.016	0.996	0.997	1	0.952	**0.999**	**0.983**
50	0.992	1.061	1.008	1.019	0.991	1.038	0.999	1.022	**0.998**	**1.035**
75	1.01	0.993	0.995	0.978	1.016	1.007	0.993	1.01	**1.004**	**0.997**
100	1.004	0.976	0.99	0.988	0.996	0.959	1.008	1.016	**1**	**0.985**

Table A.15: Reference proportion of feasible individuals ODC (Algorithm 6), - denotes no adjustment

ξ^{REF}	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
0.70	1.035	0.74	1.049	1.032	1.032	1.036	1.06	**0.913**	**1.044**	**0.93**
0.75	1.012	0.786	1.016	1.021	1.011	1.056	1.029	**0.908**	**1.017**	**0.943**
0.80	1.005	0.774	0.998	1.025	1.014	0.772	0.997	**0.883**	**1.003**	**0.863**
0.85	1.001	0.764	1.015	0.746	0.998	0.763	0.992	**0.887**	**1.002**	**0.79**
0.90	1.007	0.877	1.012	0.75	0.988	1.051	0.977	**0.886**	**0.996**	**0.891**
-	0.939	2.058	0.91	1.427	0.956	1.321	0.944	**1.524**	**0.937**	**1.582**

Table A.16: Number of iterations per ODC (Algorithm 6)

N^{ODC}	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
2	0.999	1.063	1.006	0.924	1.016	0.992	1.004	0.963	**1.006**	**0.986**
4	0.997	1.021	1.007	0.964	1.001	1.043	1.002	0.985	**1.002**	**1.003**
6	0.987	0.961	1.001	0.994	0.995	0.934	1.01	0.982	**0.998**	**0.968**
8	1.007	0.943	0.998	1.049	1.007	0.988	0.993	1.037	**1.001**	**1.004**
10	1.011	1.013	0.988	1.069	0.981	1.042	0.992	1.033	**0.993**	**1.04**

Table A.17: Number of enhancements (Algorithm 7)

n^{LES}	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
0	0.988	1.065	0.998	1.045	1.029	0.975	1.011	0.93	**1.006**	**1.004**
1	0.986	0.911	0.998	1.045	0.974	1.054	0.996	1.008	**0.989**	**1.004**
2	0.999	0.989	0.999	1.073	0.989	0.958	0.995	1.021	**0.995**	**1.01**
3	1.005	1.037	0.996	0.928	0.985	1.017	0.987	0.982	**0.993**	**0.991**
4	1.001	1.029	0.996	0.928	1.007	0.981	1.011	1.004	**1.004**	**0.986**
5	1.009	0.963	0.991	1.012	1.006	1.028	0.993	1.015	**1**	**1.004**
6	1.011	1.007	1.021	0.969	1.01	0.987	1.007	1.04	**1.012**	**1.001**

Table A.18: Balance between PHGAs and PABCs in the CJGM. The mean comes from normalizing all runs from the 4 test instances, and finally average these means.

CJGM	01C25V12T		02C25V12T		03C25V12VT		04C25V12VT		Aggregated	
PGA%	Obj	Time	Obj	Time	Obj	Time	Obj	Time	Obj	Time
0.00%	1.166	0.955	1.088	1.031	1.162	1.012	1.067	1.015	**1.121**	**1.003**
16.7%	1.074	1.004	0.969	0.9	1.1	1.014	1.088	1.031	**1.058**	**0.987**
33.3%	0.957	0.996	0.97	1.052	0.945	0.972	0.969	0.975	**0.96**	**0.999**
50.0%	0.925	1.011	0.973	1.111	0.94	0.984	0.953	0.99	**0.948**	**1.024**
66.7%	0.941	0.941	0.963	0.971	0.913	1.016	0.963	0.992	**0.945**	**0.98**
83.3%	0.985	1.079	1.006	0.981	0.975	1.021	1.028	1.024	**0.998**	**1.026**
100%	0.953	1.013	1.03	0.954	0.964	0.982	0.933	0.973	**0.97**	**0.98**

Appendix B

Detailed Results for Medium and Large-Sized Instances

Table B.1 shows the mean and best found objective value for each model with ten samples run for each instance. A total of 18 instances of customers varying from 25 to 115 customers are tested. All methods had a timeout of 30 minutes. The results are the basis for the computational study in Chapter 10. Each instance has been tested for every solution method.

Table B.1: All results from a total of 18 test instances. both the mean and the best found objective value is given for each solution method. All instances have been run a total of 10 times for each solution method.

Instance ID	HGA		MPHGA		MPABC		CJGM	
	Mean	Best	Mean	Best	Mean	Best	Mean	Best
01D1C25V12	12,505	12,083	11,883	11,591	12,074	11,755	11,789	11,587
02D1C25V12	11,890	11,631	11,418	11,126	11,624	11,285	11,154	11,087
03D2C25V12	14,311	13,817	11,135	10,939	11,127	10,882	10,589	10,569
04D2C25V12	13,584	13,165	11,264	10,617	11,982	11,683	11,042	10,628
01D2C50V25	25,313	23,551	20,296	19,879	22,303	20,873	19,925	19,231
02D2C50V25	27,498	25,593	21,766	20,566	22,585	22,205	20,767	19,968
03D1C50V25	22,763	21,420	20,032	19,296	22,286	21,974	19,870	19,298
04D1C50V25	22,646	21,393	20,061	19,274	21,565	21,098	19,623	19,263
05D1C75V37	33,182	31,081	27,791	26,881	31,528	28,548	27,759	26,728
06D2C75V37	42,836	40,650	34,262	32,566	43,923	43,317	34,227	32,855
07D2C75V37	41,720	40,941	39,724	33,403	43,565	42,485	33,988	33,204
08D2C75V37	40,594	39,745	38,772	35,129	41,589	40,788	31,960	31,052
09D2C100V50	55,115	53,723	43,389	42,347	54,534	53,283	42,638	41,311
10D2C100V50	56,589	55,363	51,769	50,505	64,812	63,175	45,525	43,928
11D2C100V50	55,945	53,901	45,643	43,792	65,488	52,874	43,956	42,040
12D2C100V50	51,614	48,001	47,583	44,252	64,529	52,457	42,680	41,909
13D2C115V62	67,819	65,320	50,276	49,575	73,682	72,412	49,944	48,398
14D2C115V62	63,770	61,819	51,764	51,049	73,198	71,154	50,131	47,670

Bakken, Glimsdal, and Tvinnereim

Heuristic Methods for the PMTVRPTW

NTNU
Norwegian University of
Science and Technology