An Evolutionary Algorithm for Waste Collection in Complex City Environments

Dag Coll Mossige
Sondre Lerum Vigerust

Industrial Economics and Technology Management
Submission date: June 2018
Supervisor: Peter Schütz, IØT

Norwegian University of Science and Technology
Department of Industrial Economics and Technology Management

Problem Description

The purpose of this thesis is to develop a solution method for solving waste collection problems in complex city environments of realistic size. Modelling complex city environments involve several extensions to the well known basic routing problems, such as intermediate facilities, one way streets and time restrictions.

Previous research has shown that problem instances of realistic size are intractable for exact methods. As such, the aim of this thesis is to design and implement a heuristic solution method capable of handling larger problem instances. The proposed solution method is applied to instances based on the real world waste collection problem faced by the local authorities in Oslo, the capital city of Norway. The performance of the heuristic will be tested on well known benchmark instances.

Preface

This thesis was written during the spring of 2018 to fulfill the graduation requirements of the study program Managerial Economics and Operations Research at the Norwegian University of Science and Technology. The thesis is based on our project report written during the fall of 2017.

The idea for the thesis arose during the fall of 2016, when the "waste crisis" in Oslo received widespread media coverage. The media attention sparked an interest in the area of waste collection, and we wanted to investigate how the process could be improved using operations research.

The work was conducted with support from the public waste management agency in Oslo, Renovasjonsetaten i Oslo (REN). It should be noted that REN were not responsible for waste collection during the waste crisis. Instead, REN were assigned the thankful job of cleaning up the mess the waste crisis had created, when they had to assume responsibility for waste collection in the middle of the nation-wide media storm.

We would like to thank our supervisor, Peter Schütz, for his dedicated support. The skilled guidance and helpful feedback he provided throughout the process has been invaluable to our work.

We would also like to thank Erik Bakos-Holthe and Rune Schau at REN for sharing their insights in the process of waste collection, and for providing all the information we asked for. Without their help, this thesis could not have been written.

Lastly, Eirik Lerum Vigerust has our gratitude for suggesting the topic for the thesis in the first place, and introducing us to REN.

Trondheim, 2018-06-11

Dag Coll Mossige

Sondre Lerum Vigerust

Summary

As the population of the world is in rapid growth and economic development leads to larger rates of consumption, the field of waste management is growing ever more important.

This thesis is based on the waste collection problem faced by the public waste management agency in Oslo, Norway. The purpose of the thesis is to develop an algorithm able to generate collection routes for waste collection problems based on real-world infrastructure.

We model the task of waste collection as a Capacitated Arc Routing Problem (CARP), a well-known routing problem in which demand occurs along the arcs of a network. The CARP is known to be ”very NP-hard”, and the potential for exact approaches is limited. As such, we present an evolutionary algorithm, developed to deal with large-scale CARP instances.

Two extensions are added to the CARP to realistically model the task of waste collection, namely intermediate facilities and time constraints on shifts.

The performance of the algorithm is tested on three sets of benchmark instances for the CARP. The algorithm performs well for the smaller instances but struggles to find optimal solutions for more complex instances within the maximum number of iterations and with the given run parameters. However, the algorithm performs best on the instances having a topology most similar to the network representing Oslo.

Four real-world instances are generated based on the infrastructure of Oslo. The smallest instance represents about 2% of the city, whereas the largest instance represents the city in its entirety. The EA is applied to the four instances, and the results are discussed in light of the algorithm’s performance on the benchmark instances.

The proposed EA is able to generate a valid collection plan for Oslo in its entirety. However, running the algorithm on the largest instance is very time-consuming. Because of time limitations, the EA was only run for 20 iterations on the instance representing Oslo. For the smaller instances, significant improvements in the solution quality were observed especially during the 200 first iterations. For the larger instances, significant improvements were still being made within the 500 iterations they were run for. Thus, it is likely that the routing plan obtained for the entire city is far from optimal.

Sammendrag

Verdens befolkning vokser raskt, og økonomisk vekst fører til høyere forbruk og økende søppelmengder. Dette fremhever viktigheten av god søppelhåndtering i tiden som kommer.

Denne masteroppgaven er basert på utfordringen med søppelhåndtering Renovasjonsetaten i Oslo (REN) står overfor. Formålet med oppgaven er å utvikle en algoritme som er i stand til å generere ruter for søppelinnhenting for probleminstanser basert på infrastrukturen i Oslo.

I oppgaven er søppelinnsamling modellert som et *Capacitated Arc Routing Problem* (CARP), et velkjent problem innen optimering. Et CARP representerer et rute-problem der oppgaver skal utføres langs kantene av et nettverk, noe som er tilfelle ved eksempelvis snømåking, veisalting, levering av post og søppelhenting. Det er velkjent at CARP tilhører klassen av NP-harde problemer, og eksakte metoder har derfor et begrenset potensiale. På bakgrunn av dette presenteres en evolusjonær algoritme som er utviklet for å håndtere store CARP-problemer. To utvidelser blir lagt til det originale problemet for å representere søppelhentingsproblemet på en mer realistisk måte.

Evolusjonære algoritmer er en type biologisk inspirerte, populasjonsbaserte algoritmer. Virkemåten til slike algoritmer er basert på evolusjonære konsepter som seleksjon, reproduksjon og mutasjon. Den foreslåtte algoritmen testes på tre sett med test-instanser fra litteraturen, i tillegg til fire instanser som er generert basert på områder i Oslo. Den minste av de genererte instansene utgjør cirka 2% av Oslo, mens den største representerer Oslo i sin helhet.

Resultatene fra test-instansene viser at den foreslåtte algoritmen sliter med å finne optimal løsning etter hvert som problemstørrelsen øker. Kjøring på den største instansen fra Oslo gir så høy kjøretid at kjøretidsbegrensninger gjør seg seg gjeldende lenge før løsningen nærmer seg optimalitet.

Den foreslåtte algoritmen er i stand til å generere gyldige innsamlingsplaner for hele Oslo, men på grunn av høy kjøretid er den bare kjørt gjennom 20 iterasjoner. Kjøringene på de mindre instansene viser konvergens før 200 iterasjoner, mens den nest største av Oslo-instansene fortsatt forbedres etter 500 iterasjoner. Dette indikerer at den største instansen trolig trenger langt mer enn 20 iterasjoner for å nå gode løsninger, og at den foreslåtte planen for denne instansen er langt fra optimal.

Contents

Chapter 1

Introduction

1.1 The importance of waste management

During the last decades, the world has seen explosive population growth, rising rates of consumption, and steadily increasing amounts of generated waste. According to Worldometers (2018), the world population grew from 6 billion in 1998 to 7.6 billion in 2018 and is projected to reach 9.2 billion by 2040. While the population growth alone suggests that the waste output of the world is likely to rise, an increasing degree of urbanization is also a contributing factor. The proportion of people living in urban areas grew from 46% in 1998 to 55% in 2018 and estimates by the World Bank Group (2012) suggest that urban residents currently produce about twice as much waste as rural residents. In 2002, the urban residents of the world produced on average 234 kilograms of waste each. By 2012, this number had almost doubled to 438 kilograms per person. These trends and their economic and environmental implications highlight the importance of efficient urban waste management systems in the future.

On a local scale, the task of waste collection and handling is one of the most important aspects of city management. First and foremost, the local environmental impact of poor waste management can be severe. Contamination of water, soil, and air, as well as spreading of infectious diseases are just a few of the consequences. Uncollected solid waste is pointed out as the primary cause of local flooding and air and water pollution by the World Bank Group (2012). Failure to properly collect, process and dispose of waste can also significantly affect an area's perceived desirability.

Furthermore, waste management is a costly affair, which burdens municipal budgets. The World Bank Group estimates that the cost of waste management worldwide will rise from $200 billion in 2012 to $300 billion in 2025, and that waste management often makes up about 20% to 50% of municipal budgets.

1.2 Scope

The importance of waste management, both concerning economic, health-related and environmental factors, makes it an attractive subject of research. Waste management is a complex task, including steps such as generation, collection, disposal, and recycling. It has been widely studied within the field of Operations Research, and OR methods can be applied to various aspects of waste management to improve decision making. Examples of such decisions include the location of new facilities, whether to expand existing facilities, the routing and allocation of collection vehicles, and deciding the optimal way of sorting waste to reduce pollution and financial costs. The interested reader is referred to Beliën et al. (2012) for an excellent review of OR methods for waste management.

According to Beliën et al. (2012), waste collection is the most costly part of waste management, due to the labor intensity of the work and the massive usage of collection vehicles. Nuortio et al. (2006) note that waste collection is one of the most difficult tasks faced by local authorities. Various studies, such as Teixeira et al. (2004) and Tavares et al. (2009) estimate that transportation makes up about 70% to 80% of operational cost related to waste collection, suggesting that route optimization can lead to substantial cost savings.

The background for this thesis is the waste collection problem faced by the local authorities in Oslo, Norway. Until recently, private actors performed the task of collecting waste in Oslo on behalf of the public waste management service, Renovasjonsetaten i Oslo. In October 2016, a company named Veireno won the public tender for waste collection in the entire city. However, it soon became evident that the company was unable to fulfill the contract on the negotiated terms. Consequently, Veireno went into bankruptcy in February 2017. During these months, the company received about 30.000 complaints from thousands of households, as hundreds of tonnes of waste remained uncollected. The period was labeled "the waste crisis" in the Norwegian media (Aftenposten (2017)). In the midst of the waste crisis, Renovasjonsetaten i Oslo took over waste collection for the entire city practically overnight.

Waste collection routing can be appropriately modeled both as a node routing problem and an arc routing problem. Nuortio et al. (2006) argue that what approach is most suitable depends on where waste accumulates. When waste accumulates in central points, as is the case when collecting from larger businesses, hospitals, and schools, a node routing approach is fitting. When waste accumulates scattered along the streets, as is the case for kerbside collection in Oslo, an arc routing approach may be more appropriate. As such, the task of waste collection is modelled as a Capacitated Arc Routing Problem (CARP) in this thesis.

The routing of a fleet of vehicles in a complex road network is a computationally demanding task, and well-known routing problems such as the vehicle routing problem and the CARP belong to the class of NP-hard problems. For this reason, exact methods have had limited success.

Various heuristic approaches have been applied to the CARP. For instance, Hertz et al. (2000) propose the first tabu search method, Lacomme et al. (2001) propose the first evolutionary algorithm and Lacomme et al. (2004b) introduce the first ant colony optimization scheme for the problem.

Evolutionary Algorithms (EA) are population-based metaheuristics inspired by biology. EAs use evolutionary concepts such as populations, generations, reproduction, and mutation to find good solutions. To the best of our knowledge, EAs were first applied to routing problems in the 1990s, by researchers like Blanton Jr and Wainwright (1993) and Potvin and Bengio (1996). Several researchers report promising results using evolutionary approaches for routing problems and note that they are computationally efficient when dealing with large problem sizes. For instance, the genetic algorithm proposed by Lacomme et al. (2001) was better than the best competing algorithms at the time. A more recent contribution is the EA proposed by Chen et al. (2016), which according to the authors is competitive with state-of-the-art CARP heuristics in terms of solution quality and runtime. Still, the large instances referred to in the existing literature are small compared to real-world scenarios, and most evolutionary algorithms are developed for such relatively small instances.

The purpose of this thesis is to develop an evolutionary algorithm for dealing with problem instances that are larger and more realistic than those studied in the existing literature. Sets of widely studied benchmark instances are used to evaluate the algorithm. Furthermore, the EA is applied to larger instances based on the real world problem faced by Renovasjonsetaten i Oslo.

The thesis is organized as follows: Chapter 2 elaborates the background for the work. Thereafter, a review of relevant literature, focusing on the CARP and evolutionary algorithms, is given in Chapter 3. In Chapter 4 the problem of waste collection faced by REN is further described, a mathematical formulation is presented. Chapter 5 discusses evolutionary algorithms in an arc routing context and presents an evolutionary algorithm for the CARP. After that, Chapter 6 introduces sets of benchmark instances from the literature and real-world instances from Oslo. The results obtained by the evolutionary algorithm are presented and discussed in Chapter 7. Lastly, Chapter 8 contains our concluding remarks, whereas Chapter 9 provides suggestions for further research.

Chapter 2

Background

Renovasjonsetaten i Oslo (REN) is currently responsible for collection and processing of municipal solid waste (MSW) in Oslo. MSW is the everyday waste that is discarded by the public. Additionally, REN collects and processes waste from smaller businesses. The waste is collected from small containers placed along streets, larger containers located on designated pickup spots, and from small recycling stations. The task of waste management in Oslo is quite comprehensive: according to SSB (2018), the city's 330 000 households generate 210 000 tonnes of waste annually.

2.1 Current situation

For about 25 years, REN outsourced the task of waste collection in Oslo to private actors. When the private actor responsible for the collection process went into bankruptcy in 2017, REN took over equipment and personnel from the bankrupt company, as well as the responsibility for waste collection. Since then, REN has been managing, planning and executing waste collection in Oslo on their own. REN is responsible for the entire waste management cycle, including monitoring, collection, processing, and disposal.

When REN assumed responsibility, during the "waste crisis" in Oslo, waste collection had not been performed adequately for months. Although the situation regarding waste collection in Oslo has significantly improved after REN took over, they are still struggling to manage their new responsibility. Figure 2.1 shows incoming complaints in April 2018. The several hundred complaints were sent to REN by concerned citizens, and they all regard uncollected waste. Route are manually customized on a day to day basis to cope with the incoming complaints. This costly practice suggests that there is still room for improvement in REN's routing operations.

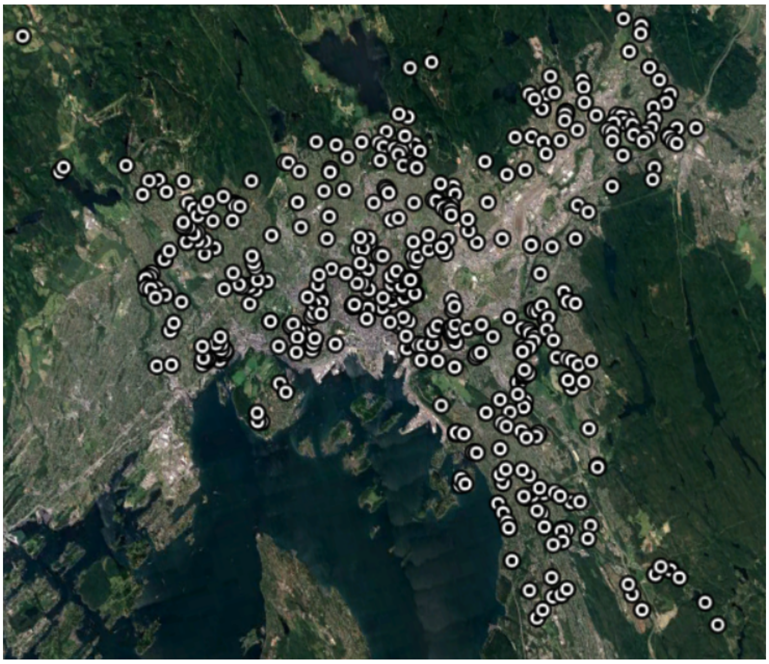


Figure 2.1: Complaints received by REN during April, 2018.

2.2 Waste statistics for Oslo

2.2.1 Waste generation

Today, Oslo has a population of about 674 000. It is by far the largest city of Norway, as well as the capital city. The city is in continuous growth due to a steady stream of people moving there. According to the news agency Reuters (2018), Oslo is one of the fastest growing cities in western Europe. The population rose from 495 000 in 1997 to 674 000 in 2018, and estimates by SSB (2018) suggest the population will rise to between 787 000 and 950 000 by 2040. The growing population itself suggests that waste generation will increase in the future. Figure 2.2.1 shows the total waste output and the per capita waste generation in Oslo for the past eleven years.

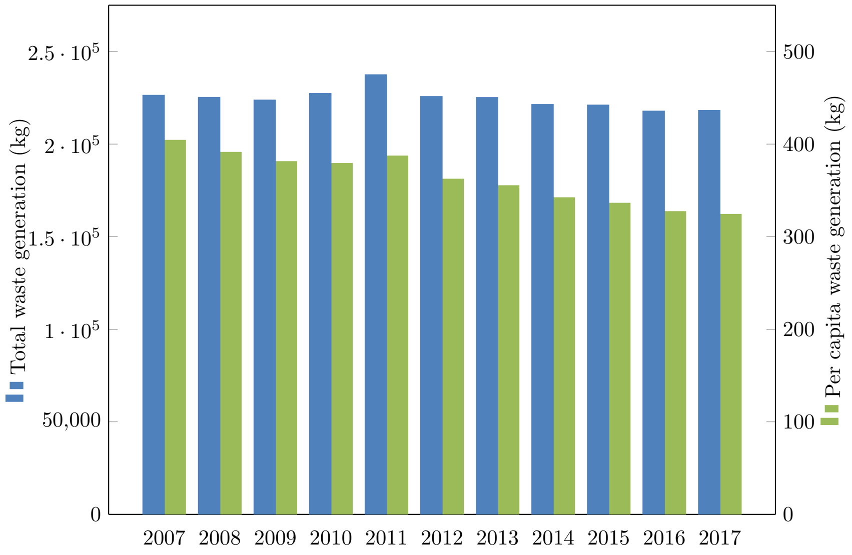


Figure 2.2: Per capita and total waste generation in Oslo. Source: SSB (2018)

As the figure indicates, the total waste output has remained relatively stable during the last decade, despite the growing population. This is due to a steady reduction in waste generation per capita during the same period.

While the total waste generation seems to be stable, it is possible that continued population growth will cause increased waste generation as it seems unlikely that the reduction rate of the per capita waste generation will remain at its current level. Furthermore, an increasing population leads to a broader and more complex network of roads and pickup points. As such, proper routing tools and practices are of increasing importance for REN going forward.

2.2.2 Waste composition

For collection and processing purposes, REN sort waste into seven distinct categories: food waste, residual waste, plastic, paper, glass and metal, garden waste and electronic appliances. Residual waste is the household waste that does not fit into any of the other categories. 2.3 shows the composition of the waste processed by REN.

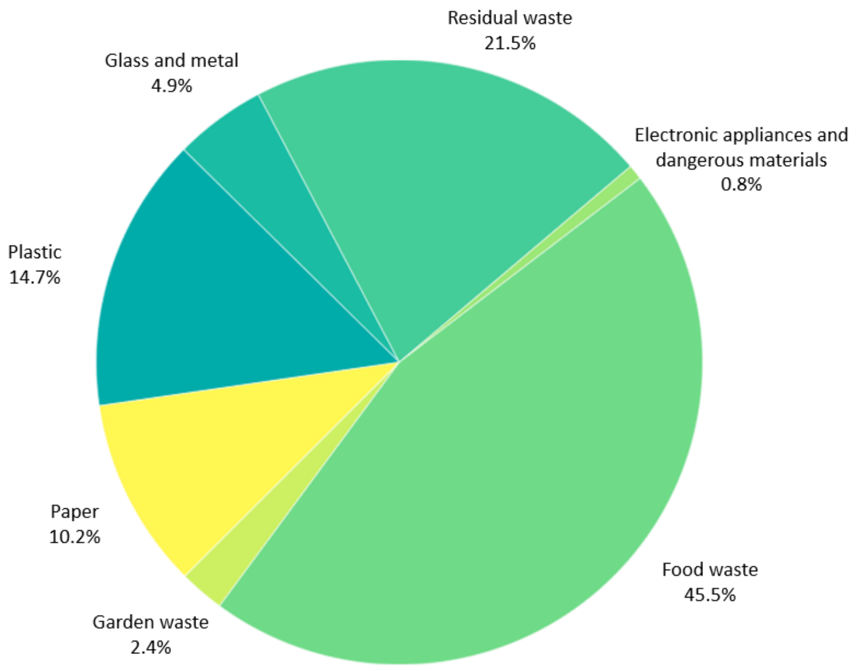


Figure 2.3: Composition of waste processed by REN in 2016.

Four of the waste types, namely food waste, paper, plastic, and residual waste arce collected directly from the households. Food waste, plastic, and residual waste are disposed of in the same containers, whereas paper is disposed of in separate containers. REN do not collect the remaining waste types directly. Instead, people deliver these types of waste to larger central containers, recycling stations or directly to one of REN's waste management facilities.

Independent routes and vehicle fleets for the different waste types. However, food waste, plastic, and residual waste are collected by the same vehicles, from the same containers, on the same routes. For sorting purposes, they are disposed of in color-encoded plastic bags. As shown in figure 2.3, food waste, plastic and residual waste constitute about 80% of the total waste processed by REN.

2.3 The waste collection process

This Section describes the task of collecting plastic, food, and residual waste, from now on referred to simply as “waste”.

Currently, REN is using a commercial software called Spider for routing purposes. Spider is used to generate routes and provides an interface for manual route customization. REN continuously update the routes generated by Spider in an effort to deal with incoming complaints.

REN’s planning and operations are controlled from the headquarter, which collocates with one of their two large waste management facilities. These facilities process and sort waste before it is recycled, burnt or taken to a landfill. In addition to the two larger facilities, REN has five recycling stations in Oslo, where people can bring larger amounts of sorted waste, as well as 27 ”mini recycling stations”. The waste from the recycling and mini recycling stations is collected by REN and brought to one of the two facilities for processing.

In REN’s systems, collection routes are presented as weekly pickup schedules where each schedule services a number of pickup locations on a given day. A pickup location is generally serviced once a week and at the same day and time every week. Each vehicle is scheduled to service about 400 pickup locations during a day, using two shifts. Currently, the shifts are conducted from 06:30 - 13:30 and 14:00 - 20:30. At the end of a shift, the vehicles have to unload at one of the two waste management facilities before returning to the depot.

Shifts are driven by teams. A team drives one shift each of the five work days. Currently, REN employs 47 teams and 27 collection vehicles to collect plastic, paper, and residual waste in Oslo. Most vehicles are used for two shifts each day, although REN has just decided that each vehicle should only drive one shift per day. For this reason, REN is in the process of acquiring additional collection vehicles.

For routing purposes, REN divides the city into four distinct collection zones. This division was made with the purpose of outsourcing each area, rather than for simplifying the routing procedure. REN, however, still uses the division. Therefore, vehicles generally service locations within a single collection zone during a shift.

The pickup locations are located within one of the four collection zones. Figure 2.4 shows the pickup locations and the road network of Oslo. Each pickup location is colored according to the zone it is located in, and contains one or more containers.

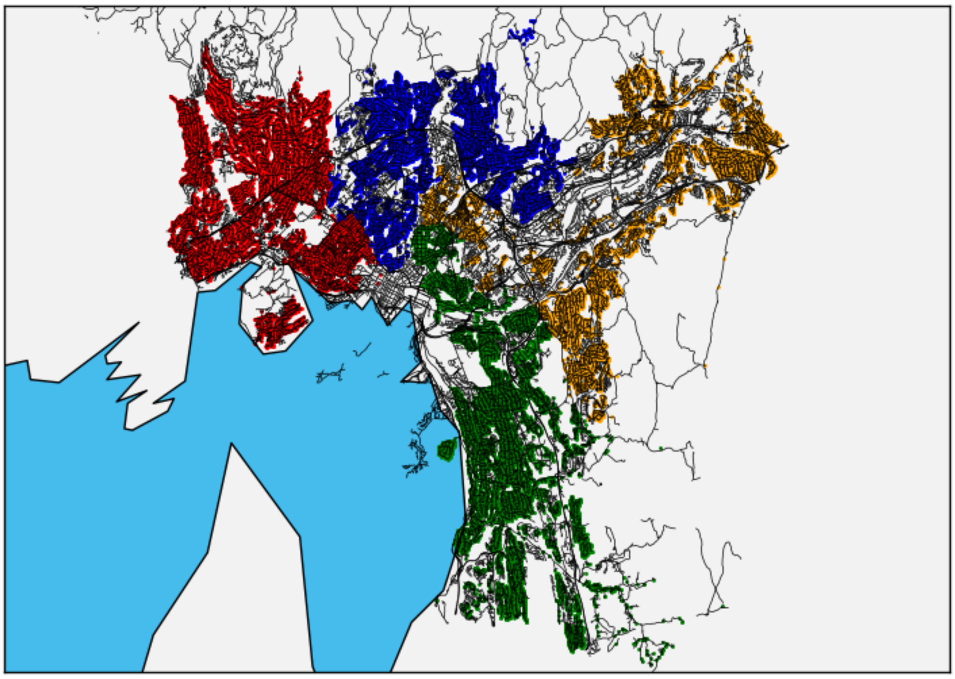


Figure 2.4: The street network in Oslo, as well as the containers for household waste.

There are 46 000 pickup locations across Oslo, which contain about 62 000 containers designated for household waste. About 80% of the locations have only one container, while the remaining 20% room up to 33 containers.

The containers are of six different sizes, ranging from 140 to 660 liters. However, REN employs a near homogeneous fleet where all vehicles can service all containers. In REN's systems, pickup locations also have additional attributes that might impact the collection process, increasing the time it takes to service the locations. These attributes include distance from the containers to the street, whether stairs or an elevator is used to reach them and whether the containers are locked or placed behind locked doors.

Chapter 3

Literature Review

The purpose of this chapter is to set the problem faced by REN in context of previous research, by giving an overview of the relevant literature. The structure of the chapter is as follows: Section 3.1 presents a formalized version of the Capacitated Arc Routing Problem and Section 3.2 discusses several possible extensions to the CARP. Lastly, Section 3.3 gives a survey of construction heuristics, improvement heuristics and evolutionary algorithms for the problem.

3.1 The Capacitated Arc Routing Problem

In arc routing problems, the challenge is to route a fleet of vehicles to visit certain arcs, called *required arcs*, in an underlying network of arcs and nodes. The objective is typically to generate a set of feasible tours that visit all the required arcs, while simultaneously minimizing the total cost. Arc routing is appropriate for modeling several real-world applications such as salt gritting, snow plowing, mail delivery, and waste collection.

Golden and Wong (1981) introduce the original Capacitated Arc Routing Problem (CARP), and prove it to be NP-hard. Golden and Wong define the original CARP on an undirected network, where each arc has a demand and a traversal cost, and vehicles have a capacity limit. A vehicle can traverse an arc any number of times, and all vehicles start and end a trip in the depot. The vehicle fleet is homogeneous, and the maximum demand of any single arc is assumed to be less than the capacity of the vehicles. When a vehicle services a required arc, it has to service the entire arc. The goal is to find a set of feasible trips that satisfy the demand of all edges.

An integer programming formulation of the undirected CARP, as presented by Mullaseril (1997), is described in the following. Let $G = (N, A)$ be an undirected graph. The set A contains all arcs (i, j), where $(i, j) \in A$ and $i, j \in N$. The set N contains all nodes $\{1, ..., n\}$, with node 1 as the depot node. Each arc (i, j) is associated with an arc cost $c_{ij} = c_{ji}$ and a demand $q_{ij} \geq 0$, where $q_{ij} = q_{ji}$. R is a subset of A that contains all arcs with nonzero demand, called required arcs. A required arcs only needs service in one direction. A fleet of K identical vehicles with capacity Q services the demand, and one vehicle corresponds to one trip. No arcs have a demand that exceeds the capacity of the vehicles, so $Q \geq q_{ij}$ for all $(i, j) \in R$.

Two sets of binary variables are used: x_{ijk} is 1 if arc (i, j) is traversed by vehicle k, and otherwise 0. The variable y_{ijk} is 1 if arc (i, j) is serviced when traversed by vehicle k, and 0 otherwise. The complete CARP formulation is given in equations 3.1 - 3.8.

$$\text{minimize} \sum_{k=1}^{K} \sum_{(i,j)\in A} c_{ij} x_{ijk} \tag{3.1}$$

$$\sum_{(j,i)\in A} x_{jik} - \sum_{(i,j)\in A} x_{ijk} = 0 \qquad i \in N, k \in \{1, ..., K\} \tag{3.2}$$

$$\sum_{k=1}^{K} (y_{jik} + y_{jik}) = 1 \qquad (i, j) \in R \tag{3.3}$$

$$\sum_{k=1}^{K} (y_{jik} + y_{jik}) = 0 \qquad (i, j) \in A \setminus R \tag{3.4}$$

$$\sum_{(i,j)\in A} q_{ij} y_{ijk} \leq Q \qquad k \in \{1, ..., K\} \tag{3.5}$$

$$x_{ijk} \geq y_{ijk} \qquad (i, j) \in A, k \in \{1, ..., K\} \tag{3.6}$$

$$X \in C \tag{3.7}$$

$$x_{ijk}, y_{ijk} \in 0, 1 \qquad i \in N, k \in \{1, ..., K\}. \tag{3.8}$$

The objective function (3.1) minimizes the total cost. Constraint 3.2 ensures that the flow in and out of nodes is the same. Constraint 3.3 requires that all required arcs are serviced in one direction, and 3.4 states that only required arcs should receive service. Constraint 3.5 imposes the capacity restrictions. It requires that the sum of demands serviced by a vehicle k does not exceed the capacity of the vehicle. Constraint 3.6 states that an arc (i, j) cannot be serviced by vehicle k unless the vehicle traverses the arc simultaneously. Constraint 3.7 is a symbolic representation of the subtour eliminating constraints - all trips must be connected, and connected to the depot node.

Researchers such as Hirabayashi et al. (1992), Belenguer and Benavent (2003) and Letchford and Oukil (2009) have tried exact methods to solve the CARP directly. However, due to the computational complexity of the problem, these exact methods are only apt to solve relatively small problems. Other researchers utilize the fact that a CARP can be transformed into an equivalent VRP. This equivalence was proven by Pearn et al. (1987), who show that a CARP with $|R|$ required arcs can be transformed into an equivalent VRP with $3\ |R| + 1$ nodes. For example, Longo et al. (2006) and Baldacci and Maniezzo (2006) apply exact VRP methods to VRP transformations of the CARP.

Due to the limited performance of exact approaches, heuristic solution methods for the CARP are studied to a far greater degree in the literature, and various heuristics are applied to solve the problem. Notable approaches include for instance the tabu search proposed by Hertz et al. (2000), the variable neighborhood search of Hertz and Mittaz (2001), the genetic algorithm by Lacomme et al. (2001), and the ant colony optimization procedure of Santos et al. (2010).

The CARP heuristics in the literature are generally benchmarked against one or several sets of CARP benchmark instances. The most widely used instance sets are the GDB (Golden et al. (1983)), KSHS (Kiuchi et al. (1995)), EGL (Li and Eglese (1996)), large-EGL (Brandão and Eglese (2008)), BBCM (Benavent et al. (1992)), and BMCV (Beullens et al. (2003)) instance sets. More recently, Liu et al. (2014) present a CARP instance generator which incorporates realistic extensions such as one-way-streets and turn restrictions. Kiilerich and Wøhlk (2018) introduce a set of CARP instances based on real-world infrastructure that are larger and have more extensions than the ones mentioned.

For a more detailed overview of the CARP, its extensions and solution methods, we referer the interested reader is referred to the excellent reviews of Wøhlk (2008) and Corberán and Prins (2010), and to Mourão and Pinto (2017) for a survey of more recent advances.

3.2 CARP Extensions

There are many extensions to the basic CARP introduced in Section 3.1. These extensions are generally added to represent real-world scenarios better. In the following, some relevant extensions will be introduced and briefly described.

Objective function

Generally, the objective is to minimize the total cost of traversals. However, other cost structures could be appropriate. Ulusoy (1985), for instance, use the sum of the total traveling cost and a distinct cost for each vehicle type that is used. Furthermore, while most researchers look at a single objective function, a multi-objective approach can be more appropriate in some cases. Lacomme et al. (2003) minimize both the total cost and the duration of the longest trip, whereas Zhang et al. (2017) seek to minimize the number of vehicles used and the total cost.

Network

The CARP as introduced by Golden and Wong (1981) is defined on an undirected network. However, this is not representative of many real-world scenarios. Street networks in cities usually include one-way streets and U-turn restrictions. Furthermore, meandering may not be allowed, meaning that vehicles must service the arcs in both directions. For these reasons, many researchers have extended the CARP to be valid for other types of networks, and with additional restrictions. Lacomme et al. (2001) for instance, define the CARP on a mixed multigraph with undirected arcs, directed arcs and parallel arcs.

The Capacitated Arc Routing Problem with Intermediate Facilities (CARPIF), introduced by Ghiani et al. (2001), represents scenarios where vehicles can unload or restock several times during a trip. Other notable papers on the CARPIF include Polacek et al. (2008), Ghiani et al. (2010) and Willemse and Joubert (2016).

Uncertainty

Another important aspect when considering a CARP is whether the problem is stochastic or deterministic. Demand and traveling time, for instance, are often subject to uncertainty. However, both are generally treated as deterministic variables, even though this is not the case in many real-world scenarios. Waste accumulation rates, amounts of snow to be plowed, traffic traveling times may be related to considerable uncertainty. A few researchers, such as Laporte et al. (2010), Fleury et al. (2004) and Fleury et al. (2005) consider uncertainty by looking into the stochastic CARP.

3.3 Heuristics for the CARP

Heuristics for the CARP can be divided into two types, namely construction heuristics and improvement heuristics. Construction heuristics, such as greedy randomized adaptive search build solutions from scratch. Improvement heuristics, such as simulated annealing, tabu search, variable neighborhood search, and evolutionary algorithms, improve existing solutions in an iterative process (Gendreau and Potvin (2005)). These improvement heuristic are metaheuristics; they can be applied to a range of different problems. The next three sections are dedicated to construction heuristics, improvement heuristics, and evolutionary algorithms, respectively, and discuss these in relation to the CARP.

3.3.1 Construction heuristics

During the 1970s and 1980s, heuristics such as the Construct-Strike of Christofides (1973) and the Augment-Merge and Path-Scanning methods of Golden et al. (1983) were proposed for the CARP. These heuristics are all problem specific, meaning that cannot easily be generalized to other problem types. Generally, Wøhlk (2008) notes, the solutions found by these algorithms are 10 to 40 percent above the optimal solutions. Dror (2012) describes the three heuristics in further detail.

In more recent years, extensions of these problem-specific heuristics are widely used as construction heuristics for the CARP. For instance, Lacomme et al. (2004a) use them, in addition to the route-first cluster-second procedure introduced in Ulusoy (1985), to create good initial individuals for a genetic algorithm.

3.3.2 Improvement heuristics

The first improvement heuristic for the CARP is proposed by Eglese (1994) for a road gritting problem. The authors use a simulated annealing approach, that generates random neighborhood moves, and accepts them by default if they improve the existing solution. Otherwise, the moves are accepted with a certain probability. Wøhlk (2006) also presents a simulated annealing approach for the CARP.

The first tabu search for the CARP, named CARPET, was proposed by Hertz et al. (2000). Tabu search is an iterative local search procedure. In each iteration, the algorithm chooses the best solution in the neighborhood of the current solution as the new current solution. It is called tabu search as a list of illegal, or ”tabu” moves is kept in memory to avoid cycles and local optima. CARPET performed better than existing methods at the time. For a detailed description of CARPET, the reader is referred to Hertz et al. (2000). Various other researchers have also applied tabu search methods to the problem. For example, Amberg et al. (2000) propose a tabu search for the multiple depot CARP, Archetti et al. (2006) apply it to the classical CARP and Brandão and Eglese (2008) suggest a deterministic tabu search.

A Very Large Neighbourhood Descent (VND) algorithm for the undirected CARP is suggested by Hertz and Mittaz (2001). The authors report that the VND algorithm is competitive with CARPET, and outperforms CARPET for larger problem instances where it provides better solutions and shorter runtimes.

A Genetic Algorithm (GA) which outperforms the best CARP heuristics at the time, including CARPET, is proposed by Lacomme et al. (2001). Genetic algorithms for the CARP are discussed in further detail in subsection 3.3.3.

Another important class of metaheuristics is Ant Colony Optimization (ACO), in which pathfinding is inspired by the behavior of ants. Dorigo et al. (2006) provide an excellent introduction to this class, that has been applied to various versions of the CARP. Lacomme et al. (2004a) propose an ACO based algorithm, and compare it to CARPET and the mentioned genetic algorithm by Lacomme et al. (2001). While the ACO algorithm performed better than CARPET, the researchers note that it could not compete with the GA regarding computational time. Santos et al. (2010) also apply ACO to the CARP, and report that their algorithm was competitive with the best algorithms at the time, such as CARPET and the GA of Lacomme et al..

3.3.3 Evolutionary Algorithms

Evolutionary Algorithm (EA) is a term describing algorithms that are inspired by fundamental evolutionary concepts, such as populations, generations, selection, reproduction and mutation. Genetic Algorithms and Memetic Algorithms (MA) are very similar, and both belong to the class of evolutionary algorithms. However, MAs generally apply a more extensive search procedure than GAs.

The first published GA for the CARP is introduced by Lacomme et al. (2001). Their GA considers three extensions of the basic CARP, namely directed arcs, a set of required arcs with servicing costs in addition to traversal costs, and prohibited U-turns. Lacomme et al. (2004a) build on this work, and suggest several memetic algorithms for a similar version of the CARP.

A memetic algorithm for the Node Edge Arc Routing Problem (NEARP), in which demand can occur both along edges and at nodes, is proposed by Prins and Bouchenoua (2005). The authors argue that VRP and CARP are inadequate in representing all real-world scenarios. A memetic algorithm for the Split Delivery CARP, in which multiple vehicles can participate in servicing a single required arc, is proposed by Labadi et al. (2008).

In Tang et al. (2009), a Memetic Algorithm with Extended Neighbourhood Search (MAENS) is proposed for the CARP. The authors use a new operator called the Merge-Split, which has a large step size that allows it to search through a greater neighborhood, reducing the chances of getting stuck in local optima. The Merge-Split operator is a combination of the Path-Scanning and Ulusoy's tour splitting heuristics. MAENS obtains the best solution in 175 out of 181 CARP benchmark instances and finds improved best-known solutions to 16 of them.

A memetic algorithm for the open CARP is suggested by Fung et al. (2013). In the open CARP, vehicles do not need to return to the depot at the end of a trip. The approach utilizes the equivalence of the CARP and the VRP by transforming the open CARP to the corresponding open VRP before applying the memetic algorithm.

In Liu et al. (2014) a memetic algorithm is suggested and tested for three real-world capacitated arc routing scenarios, namely winter gritting, inspection of electric power lines and waste collection. A modified neighborhood search operator is used to improve the rate of convergence.

In Chen et al. (2016) a genetic algorithm is combined with a tabu threshold procedure, and a new crossover operator is proposed. The authors report that the algorithm finds best known or improved best-known results for nearly all of the benchmark instances to which it is applied.

A bi-objective memetic algorithm for the periodic CARP is suggested by Zhang et al. (2017). A new operator called the Route Decomposition operator is introduced. The authors report that their algorithm outperforms competing heuristics for the periodic CARP.

Chapter 4

Problem formulation

The following chapter describes the waste collection problem faced by REN and presents a mathematical formulation for the problem.

Section 4.1 describes the problem, followed by an overview of assumptions that are made in section 4.2. Thereafter, Section 4.3 presents a mathematical formulation for problem. Section 4.4 elaborates potential objective functions and consequences of these, whereas Section 4.5 introduces two symmetry breaking constraints. For the reader's convenience, Section 4.6 contains the complete mathematical formulation.

4.1 Problem description

The basis for the problem is the task of waste collection faced by REN. In short, the challenge is routing a fleet of vehicles to collect waste from all private households in the large, complex city environment of Oslo.

A fleet of identical waste collection vehicles performs the waste collection. The vehicles drive in shifts and can conduct a given number of shifts each day. A shift has a start time and an end time, and the duration of all shifts is equal. The vehicles leave a central depot at the beginning of each shift and return before the end of the shift. Each vehicle has a given waste capacity and must unload at one of the available waste management facilities when it reaches its capacity limit. A vehicle can unload multiple times during each shift. The vehicles also have to unload at one of the facilities at the end of a shift, before heading back to the depot.

A trip is a path driven by a vehicle between consecutive visits to the depot or a waste management facility. A trip therefore always starts and ends in the depot or a waste management facility. A vehicle can drive multiple trips during a shift, within the given shift duration.

A city environment like the one found in Oslo can be described as a network of streets and intersections. A street between two neighboring intersections is referred to as a street segment. The network includes both one-way streets and "regular" two-way streets. For most of the two way-streets, the vehicles can collect waste from both sides of the street regardless of the driving direction. For some two-way streets, however, the vehicles have to drive along the street in both directions to collect waste from both sides.

The vehicles collect waste from a series of pickup locations that lie along streets. Each pickup location must be serviced precisely once within the planning horizon.

The term deadheading refers to a vehicle that drives a street segment without collecting waste. Picking up waste along a segment takes significantly more time than deadheading it. The cost associated with servicing a street segment is therefore additional to the cost associated with deadheading it.

The overall goal of the problem is to find a set of shifts that minimize the total cost while adhering to the capacity, time and collection constraints.

4.2 Assumptions and simplifications

In order to model the problem mathematically, several assumptions and simplifications are introduced. Firstly, the proposed model is deterministic. Secondly, service is conducted on street segments, not on individual containers. Thus, the total waste volume associated with a street segment is the aggregate of the volume of all bins along the given segment. If a vehicle collects from a street segment, it must collect from all of its associated containers. Therefore, a vehicle cannot service a street segment if the amount of waste along it exceeds the remaining vehicle capacity. This also leads to another assumption: no street segment can have a total waste volume that exceeds the maximum vehicle capacity.

Lastly, the model disregards periodic variations in traffic. In practice, certain areas, such as districts near the city center, should be serviced during certain times of day to avoid traffic.

4.3 The Capacitated Arc Routing Problem with Intermediate Facilities and Time Constraints

Let $\mathcal{G} = (\mathcal{N}, \mathcal{A})$ be a directed graph with nodes $\mathcal{N}$ and arcs $\mathcal{A}$. $\mathcal{N}$ represents the intersections in a network of road segments, whereas $\mathcal{A}$ represents the road segments between intersections. The set $\mathcal{A}$ consists of all possible arcs (i,j), where $i,\ j \in \mathcal{N}$. A street that can be traversed in both directions is represented with two oppositely directed arcs. Such a segment between nodes i and j is represented by the arcs (i,j) and (j,i).

The case where a vehicle may collect from both sides of the street when driving in one direction is referred to as meandering. The set $\mathcal{A}^R$, $\mathcal{A}^R \subset \mathcal{A}$, represents all arcs that require service, and in general, demand is added to both oppositely directed arcs (i,j) and (j,i). If meandering is allowed for a required arc, service must only be conducted on one of the arcs for demand to be satisfied. The set $\mathcal{A}^{TWS}$, which is a subset of $\mathcal{A}$ represents the arcs where meandering is not allowed. It contains all required arcs (i,j) that require service in both directions. For these arcs, both oppositely directed arcs must be serviced to satisfy the demand.

The set $\mathcal{N}^E$ contains all nodes that represent waste management facilities, where $\mathcal{N}^E \subset \mathcal{N}$. The node $\hat{D}$ represents the depot node.

$\mathcal{R}$ represents the set of all possible shifts that can be generated, defined as $\mathcal{R} := \{1, ..., \bar{r}\}$, where $\bar{r}$ is the maximum number of shifts that can be conducted in the planning horizon. Each shift $r \in \mathcal{R}$ starts in the depot and has to return to the depot within the maximum shift duration, T^{max}. During a shift, a vehicle can conduct one or more trips.

A trip, k, is the path traveled between two consecutive visits at either the depot or a waste management facility. Thus, a new trip always starts when leaving the depot or a waste management facility, and ends when visiting the depot or a waste management facility. The parameter $\bar{k}$ gives the maximum number of trips per shift. The set $\mathcal{K}$ represents all possible trips and is defined as $\mathcal{K} := \{1, ..., \bar{k}\}$.

The binary variable x_{ijkr} describes whether arc (i, j) is serviced in trip k of shift r, while the integer variable y_{ijkr} is equal to the number of times an arc is traversed, with or without service, on trip k of shift r. The binary variable a_r is set to 1 if shift r is used, 0 otherwise. Likewise, l_{kr} is 1 if trip k is used on shift r, and 0 otherwise.

The parameter V_{ij} describes the total volume of waste along an arc (i, j), and V^{max} is the volume capacity of the vehicles. T^D_{ij} denotes the time it takes to deadhead the arc (i, j), and T^S_{ij} is the extra time it takes to service arc (i, j) compared to deadheading the arc.

An overview of the notation is provided in the following.

Notation

Indices:

i,j,n	Nodes
$\hat{D}$	Depot node
k	Trip
r	Shift

Parameters:

$\bar{r}$	Maximum number of shifts in the planning horizon
$\bar{k}$	Maximum number of trips per shift
V_{ij}	Total waste volume on arc (i, j)
V^{max}	Maximum capacity of any vehicle
T_{ij}^{D}	Total traversal time of arc (i, j) when deadheading the arc
T_{ij}^{S}	Additional traversal time of arc (i, j) when servicing the arc
T^{max}	Maximum allowed cumulative time for any shift
M^{V}	Represents a large volume, e.g. V^{max}
M^{T}	Represents a large time, e.g. T^{max}
M^{C}	Represents a large number of times an arc can be traversed during a trip
M^{N}	Represents a large number of times a node can be visited during a trip

Sets:

$\mathcal{N}$	Set of all nodes
$\mathcal{N}^{E}$	Set of all dumpsites ($\mathcal{N}^{E} \subset \mathcal{N}$)
$\mathcal{A}$	Set of all possible arcs, i.e. all streets the vehicles can drive
$\mathcal{A}^{R}$	Set of all required arcs, i.e. all arcs with positive waste volumes ($\mathcal{A}^{R} \subset \mathcal{A}$)
$\mathcal{A}^{TWS}$	Set of all arcs which requires service in both directions ($\mathcal{A}^{TWS} \subset \mathcal{A}^{R}$)
$\mathcal{R}$	Set of all shifts, $\mathcal{R} := \{1...\bar{r}\}$
$\mathcal{K}$	Set of possible trips, $\mathcal{K} := \{1...\bar{k}\}$
$\mathcal{S}$	Set of possible subtours
$\mathcal{S}_s$	Set of nodes representing a subtour ($\mathcal{S}_s \subset \mathcal{S}$)

Variables:

x_{ijkr}	1 if arc (i,j) is serviced on trip k, shift r, 0 otherwise
y_{ijkr}	Integer, equals the number of time arc (i,j) is serviced on trip k, shift r
a_r	1 if shift r is used, 0 otherwise
l_{kr}	1 if trip k is used on shift r, 0 otherwise
c_{nkr}	1 if node n is visited on trip k, shift r, 0 otherwise

Constraints

The constraints of the model belong to two categories: resource constraints and bookkeeping constraints. The resource constraints regard duration and capacity, and are given in constraints 4.1 and 4.2.

$$\sum_{(i,j)\in\mathcal{A}} x_{ijkr}V_{ij} \leq V^{max} \qquad k \in \mathcal{K}, r \in \mathcal{R} \tag{4.1}$$

$$\sum_{(i,j)\in\mathcal{A}} \sum_{k\in\mathcal{K}} (y_{ijkr}T^T_{ij} + x_{ijkr}T^S_{ij}) \leq T^{max} \qquad r \in \mathcal{R}. \tag{4.2}$$

Constraint 4.1 ensures that the waste volume of a vehicle can not exceed the vehicle capacity, while constraint 4.2 states that no shift can have a duration longer than the maximum duration of a shift.

Further, the bookkeeping constraints of the model are presented. First, constraints are added to ensure that each required arc is serviced.

$$\sum_{r\in\mathcal{R}} \sum_{k\in\mathcal{K}} (x_{ijkr} + x_{jikr}) = 1 \qquad (i,j) \in \mathcal{A}^R \setminus \mathcal{A}^{TWS} \tag{4.3}$$

$$\sum_{r\in\mathcal{R}} \sum_{k\in\mathcal{K}} (x_{ijkr} + x_{jikr}) = 2 \qquad (i,j) \in \mathcal{A}^{TWS} \tag{4.4}$$

Constraint 4.3 states that each required pair of arcs that only needs servicing in one direction is serviced exactly once. Constraint 4.4 expresses that if a pair of oppositely directed arcs does not allow meandering, both arcs in the pair must be serviced.

Each shift used in a solution must start in the depot, leave the depot exactly once and enter the depot exactly once. The following constraints add these requirements to the model:

$$\sum_{j\in\mathcal{N}} \sum_{k\in\mathcal{K}} y_{\hat{D}jkr} = a_r \qquad r \in \mathcal{R} \tag{4.5}$$

$$\sum_{j\in\mathcal{N}} y_{\hat{D}j1r} = a_r \qquad r \in \mathcal{R} \tag{4.6}$$

$$\sum_{i\in\mathcal{N}\setminus\{\hat{D}\}} \sum_{k\in\mathcal{K}} y_{i\hat{D}kr} = a_r \qquad r \in \mathcal{R}, \tag{4.7}$$

where 4.5 ensures that a vehicle leaves the depot exactly once during a shift, and 4.6 expresses that this must happen during the first trip of the shift. Constraint 4.7 ensures that a vehicle visits the depot node exactly once during a shift.

All trips should start either in the depot or a waste management facility. The constraint (4.8),

$$\sum_{j\in\mathcal{N}} y_{ij(k+1)r} = \sum_{j\in\mathcal{N}} y_{jikr} \qquad i\in\mathcal{N}^E, k\in\mathcal{K}\setminus\{\bar{k}\}, r\in\mathcal{R}, \tag{4.8}$$

ensures that a new trip starts from a waste management facility node if the previous trip in the shift ended up in the respective node.
The constraint (4.9),

$$\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}^E} y_{ijkr} + \sum_{i\in\mathcal{N}} y_{i\hat{D}kr} = l_{kr} \qquad k\in\mathcal{K}, r\in\mathcal{R}, \tag{4.9}$$

expresses that either the depot or one of the waste management facilities should be visited exactly once for each trip that is used.

For each trip, the number of traversals entering a node is the same as the number of traversals leaving the node. Constraint (4.10) ensures this.

$$\sum_{i\in\mathcal{N}\setminus\{j\}} y_{ijkr} = \sum_{i\in\mathcal{N}\setminus\{j\}} y_{jikr} \qquad j\in\mathcal{N}\setminus(\mathcal{N}^E,\{\hat{D}\}), k\in\mathcal{K}, r\in\mathcal{R}. \tag{4.10}$$

The constraint (4.11),

$$x_{ijkr} \leq y_{ijkr} \qquad (i,j)\in\mathcal{A}, k\in\mathcal{K}, r\in\mathcal{R}, \tag{4.11}$$

is added to ensure that an arc cannot be serviced in a trip unless it is traversed on the same trip.

Furthermore, the constraints

$$l_{kr} \leq a_r \qquad k\in\mathcal{K}, r\in\mathcal{R} \tag{4.12}$$
$$y_{ijkr} \leq l_{kr}M^C \qquad (i,j)\in\mathcal{A}, k\in\mathcal{K}, r\in\mathcal{R} \tag{4.13}$$

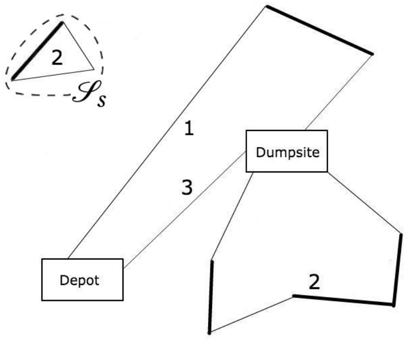
are added. Constraint 4.12 states that no trip should be used unless its respective shift is used, and 4.13 expresses that no arc should be traversed unless its respective trip is used.

The above formulation does not ensure connectivity of each trip. To ensure connectivity, the constraints

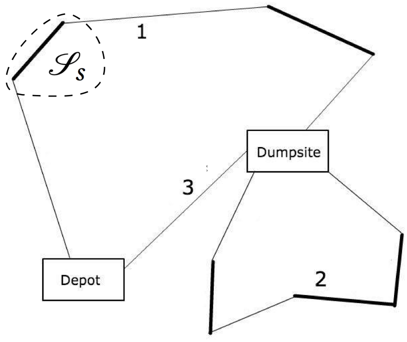
$$\sum_{i\in\mathcal{N}} y_{inkr} \leq c_{nkr}M^N \qquad n\in\mathcal{N}, k\in\mathcal{K}, r\in\mathcal{R} \tag{4.14}$$
$$c_{nkr} \leq \sum_{i\in\mathcal{S}_s}\sum_{j\in\mathcal{N}\setminus\mathcal{S}_s} y_{ijkr} \qquad \mathcal{S}_s\in\mathcal{S}, n\in\mathcal{S}_s, k\in\mathcal{K}, r\in\mathcal{R} \tag{4.15}$$

are added to the formulation. Constraint 4.14 ensures that c_{nkr} is 1 if node n

is visited during the respective trip. Constraint 4.15 eliminates subtours. The constraint can be interpreted as "if node n is in subset $\mathcal{S}_S$ and the node is visited during trip k, shift r, then there must be at least one traversal leaving the subset of nodes during the respective trip." Figure 4.1 illustrates a possible solution to the problem with and without the connectivity constraints. The nodes included in the subtour in figure 4.1a represent a subset $\mathcal{S}_S$, and applying the connectivity constraints ensures that there is at least one arc leaving the subset, thus eliminating the subtour. The numbers in the figure denotes trips.



(a) Without subtour elimination



(b) With subtour elimination

Figure 4.1: Conceptual sketch of shift with and without subtour elimination.

Lastly, the following domain constraints are added:

$$x_{ijkr} \in \{0,1\} \qquad (i,j) \in \mathcal{A}, k \in \mathcal{K}, r \in \mathcal{R} \tag{4.16}$$
$$y_{ijkr} \geq 0, \text{ integer} \qquad (i,j) \in \mathcal{A}, k \in \mathcal{K}, r \in \mathcal{R} \tag{4.17}$$
$$a_r \in \{0,1\} \qquad r \in \mathcal{R} \tag{4.18}$$
$$l_{kr} \in \{0,1\} \qquad k \in \mathcal{K}, r \in \mathcal{R} \tag{4.19}$$
$$c_{nkr} \in \{0,1\} \qquad n \in \mathcal{N}, k \in \mathcal{K}, r \in \mathcal{R}. \tag{4.20}$$

4.4 Potential objectives

Multiple alternative objectives can be considered. Firstly, one could seek to minimize costs directly. This requires detailed estimates of the costs associated with the purchase, operation, and maintenance of the vehicles, as well as staffing costs.

Another option is to minimize the number of shifts needed. However, such a formulation does not necessarily facilitate optimal routing of the vehicles. Small changes in the shifts may not increase the number of shifts needed, even though other properties of the solution, such as total deadheading time, may be worsened.

A fourth alternative is to minimize either the time spent on deadheading or the total makespan of the collection process. An objective function seeking to minimize the total makespan of the collection process is used.

4.5 Improvements of the model

As the order of the shifts and trips does not matter, there is symmetry in the problem. The following symmetry breaking constraints break this symmetry and reduce the solution space.

$$a_r \geq a_{r+1} \qquad r \in \mathcal{R} \setminus \{\bar{r}\} \tag{4.21}$$

$$\sum_{k \in \mathcal{K}} l_{kr} \geq \sum_{k \in \mathcal{K}} l_{k(r+1)} \qquad r \in \mathcal{R} \setminus \{\bar{r}\} \tag{4.22}$$

Constraint 4.21 states that a shift $r+1$ should only be used if shift r is used. Furthermore, constraint 4.22 ensures that shift $r+1$ cannot consist of a larger number of trips than the shift r.

4.6 Complete formulation

$$\text{minimize} \sum_{r\in\mathcal{R}}\sum_{k\in\mathcal{K}}\sum_{(i,j)\in\mathcal{A}}[x_{ijkr}T^S_{ij} + y_{ijkr}T^D_{ij}]$$

subject to

$$\begin{array}{ll}
\displaystyle\sum_{(i,j)\in\mathcal{A}} x_{ijkr}V_{ij} \le V^{max} & k\in\mathcal{K}, r\in\mathcal{R} \\
\displaystyle\sum_{(i,j)\in\mathcal{A}}\sum_{k\in\mathcal{K}}[y_{ijkr}T^T_{ij} + x_{ijkr}T^S_{ij}] \le T^{max} & r\in\mathcal{R} \\
\displaystyle\sum_{r\in\mathcal{R}}\sum_{k\in\mathcal{K}}[x_{ijkr} + x_{jikr}] = 1 & (i,j)\in\mathcal{A}^R\setminus\mathcal{A}^{TWS} \\
\displaystyle\sum_{r\in\mathcal{R}}\sum_{k\in\mathcal{K}}[x_{ijkr} + x_{jikr}] = 2 & (i,j)\in\mathcal{A}^{TWS} \\
\displaystyle\sum_{j\in\mathcal{N}}\sum_{k\in\mathcal{K}} y_{\hat{D}jkr} = a_r & r\in\mathcal{R} \\
\displaystyle\sum_{j\in\mathcal{N}} y_{\hat{D}j1r} = a_r & r\in\mathcal{R} \\
\displaystyle\sum_{i\in\mathcal{N}\setminus\{\hat{D}\}}\sum_{k\in\mathcal{K}} y_{i\hat{D}kr} = a_r & r\in\mathcal{R} \\
\displaystyle\sum_{j\in\mathcal{N}} y_{ij(k+1)r} = \sum_{j\in\mathcal{N}} y_{jikr} & i\in\mathcal{N}^E, k\in\mathcal{K}\setminus\{\bar{k}\}, r\in\mathcal{R} \\
\displaystyle\sum_{i\in\mathcal{N}}\sum_{j\in\mathcal{N}^E}[y_{ijkr} + \sum_{i\in\mathcal{N}} y_{i\hat{D}kr}] = l_{kr} & k\in\mathcal{K}, r\in\mathcal{R} \\
\displaystyle\sum_{i\in\mathcal{N}\setminus\{j\}} y_{ijkr} = \sum_{i\in\mathcal{N}\setminus\{j\}} y_{jikr} & j\in\mathcal{N}\setminus(\mathcal{N}^E,\{\hat{D}\}), k\in\mathcal{K}, r\in\mathcal{R} \\
x_{ijkr} \le y_{ijkr} & (i,j)\in\mathcal{A}, k\in\mathcal{K}, r\in\mathcal{R} \\
l_{kr} \le a_r & k\in\mathcal{K}, r\in\mathcal{R} \\
y_{ijkr} \le l_{kr}M^C & (i,j)\in\mathcal{A}, k\in\mathcal{K}, r\in\mathcal{R} \\
\displaystyle\sum_{i\in\mathcal{N}} y_{inkr} \le c_{nkr}M^N & n\in\mathcal{N}, k\in\mathcal{K}, r\in\mathcal{R} \\
\displaystyle c_{nkr} \le \sum_{i\in\mathcal{S}_s}\sum_{j\in\mathcal{N}\setminus\mathcal{S}_s} y_{ijkr} & \mathcal{S}_s\in\mathcal{S}, n\in\mathcal{S}_s, k\in\mathcal{K}, r\in\mathcal{R} \\
a_r \ge a_{r+1} & r\in\mathcal{R}\setminus\{\bar{r}\} \\
\displaystyle\sum_{k\in\mathcal{K}} l_{kr} \ge \sum_{k\in\mathcal{K}} l_{k(r+1)} & r\in\mathcal{R}\setminus\{\bar{r}\} \\
x_{ijkr}\in\{0,1\} & (i,j)\in\mathcal{A}, k\in\mathcal{K}, r\in\mathcal{R} \\
y_{ijkr}\ge 0,\ \text{integer} & (i,j)\in\mathcal{A}, k\in\mathcal{K}, r\in\mathcal{R} \\
a_r\in\{0,1\} & r\in\mathcal{R} \\
l_{kr}\in\{0,1\} & k\in\mathcal{K}, r\in\mathcal{R} \\
c_{nkr}\in\{0,1\} & n\in\mathcal{N}, k\in\mathcal{K}, r\in\mathcal{R}.
\end{array}$$

Chapter 5

Solution Method

In this chapter, we propose an evolutionary algorithm developed to handle large-scale capacitated arc routing problems, similar to the waste collection problem faced by REN.

Evolutionary Algorithms (EAs) is a term describing biologically inspired, population-based metaheuristics. The term *population-based* indicates that it operates on a set of solutions simultaneously, rather than on a single solution. In this setting, a solution is referred to as an individual. A population consists of a set of individuals, and a generation is a population in a given iteration of the algorithm. For instance, the set of solutions in the fourth iteration is referred to as the fourth generation. The best solutions, i.e., the fittest individuals, in a generation have the highest likelihood of propagating to the next generation.

A *chromosome* represents the genetic material of an individual. In the reproduction phase, individuals are selected for reproduction, and the chromosomes of the selected individuals are combined to create new offspring. The reproduction phase is referred to as crossover, and the crossover corresponds to the mating part of evolution. The crossover results in one or more new individuals that are the products of the two parents. Random alterations may be applied to some of the individuals, and this procedure constitutes the mutation part of an EA. For a more in-depth explanation of evolutionary algorithms, the reader is referred to Whitley (1994).

The first two sections of this chapter present the technicalities of our algorithm. The remaining sections elaborate on concepts in evolutionary algorithms and explain how our EA has implemented these concepts. This chapter is organized as follows: Section 5.1 elaborates on our choice of chromosome representation, and Section 5.2 presents two operators needed for constructing and altering chromosomes. A broad overview of the proposed EA is given in Section 5.3. Section 5.4

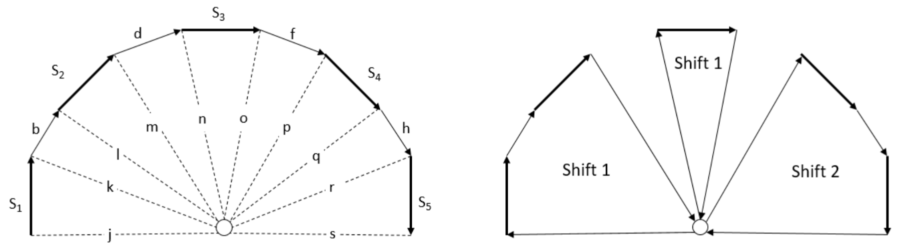
introduces a construction heuristic that is used to generate the initial populations, whereas Section 5.5 describes how the fitness of the individuals is measured. Section 5.6 explains how feasible solutions are extracted from chromosomes. Section 5.7 elaborates on the selection and crossover phase, whereas Section 5.8 discusses mutation procedures. Lastly, Section 5.9 describes how the solutions generated by the EA can be transformed into actual routing plans.

5.1 Chromosome Representation

An important aspect of an evolutionary algorithm is to decide how an individual should be represented by it's corresponding chromosome. One alternative is to let chromosomes represent actual solutions to the problem simply. In evolutionary terms, a chromosome is then the same as a *phenotype*. However, this is not a necessity. The other option is to let the chromosome be a *genotype*, an abstraction of the underlying solution. The actual solutions, phenotypes, are in this case created by applying a decoding procedure on the genotypes.

In the routing context, chromosomes are generally genotypes. A very common abstraction is to let chromosomes represent sequences of tasks. These sequences specify the order in which the tasks are executed. This chromosome representation does not contain the paths between each consecutive task. Instead, shortest paths are assumed. Trip delimiters are usually not included in this chromosome representation, implying that the chromosomes represent *giant tours* that do not adhere to capacity or time constraints. A chromosome represented in this manner needs to be split into a set of feasible trips to get actual solutions.

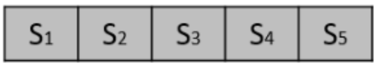
The process of decoding giant tour chromosomes into feasible solutions is illustrated in figures 5.1 and 5.2. Figure 5.1a shows a giant tour chromosome where capacity and time restrictions are ignored. The bold lines represent tasks, while the thin lines represent deadheaded arcs. Dotted lines are arcs that are not used. After applying an appropriate decoding procedure to the giant tour, which accounts for the time and capacity restrictions, the resulting solution is found. Such a solution is shown in 5.1b. The solution consists of two shifts, where shift 1 has two trips, and shift 2 has one trip.



(a) Giant tour (b) Solution

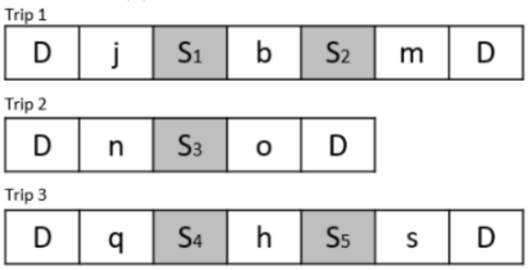
Figure 5.1: Difference between chromosome and solution.

In figure 5.2, the chromosome representing the giant tour, as well as the corresponding solution, is shown. D denotes the depot, letters represent the arcs from the network in 5.1, and S_1-S_5 are the tasks from the network. Grey entries represent tasks, and white entries represent the arcs of the shortest path between consecutive tasks. In the chromosome, these shortest paths are not explicitly represented. Rather, the cost of the shortest paths between each pair of tasks is stored in a precomputed distance matrix. Thus, the shortest paths are given implicitly through lookup in the distance matrix. The chromosome representation introduced above has been used in several papers including Lacomme et al. (2001), Lacomme et al. (2004b), Prins (2004), Prins and Bouchenoua (2005) and Lacomme et al. (2006). These papers all apply a decoding procedure that splits a giant tour optimally. This procedure is described in Section 5.6.



S_1	S_2	S_3	S_4	S_5

(a) Chromosome, genotype



Trip 1

D	j	S_1	b	S_2	m	D

Trip 2

D	n	S_3	o	D

Trip 3

D	q	S_4	h	S_5	s	D

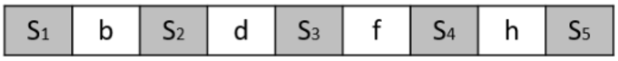
(b) Corresponding solution, phenotype

Figure 5.2: Difference between chromosome and solution.

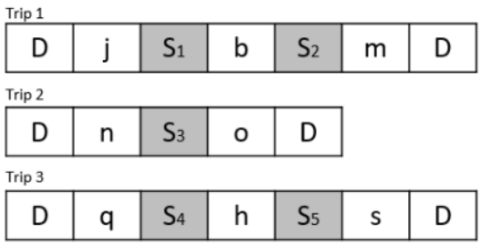
Our chromosome representation

Computing a matrix containing the distance between all task-pairs and all tasks and the depot is trivial for small instances. However, the size of such a matrix grows exponentially with the number of tasks. As does the number of shortest path calculations needed to fill the matrix. For an instance containing 50 required arcs, 2 500 shortest path calculations must be made, given that one operates on a directed network. Creating such a matrix is manageable. For a network with 7 281 required arcs, like the one representing Oslo, however, the number of shortest path calculations is over 53 million. Each shortest path calculation also gets more demanding as the size of the network increases. Thus, the approach of pre-calculating such a distance matrix is problematic in terms of scalability.

The chromosome representation applied in our GA is based on the one prevalent in the arc routing literature. The chromosomes are abstractions of the underlying solutions and require a decoding procedure to produce valid solutions. Each chromosome is given without trip delimiters and constitutes a *giant tour*. However, the representation is different in an important way. Rather than using implied shortest paths between tasks, and a distance matrix containing shortest paths between each task-pair, our chromosomes contain explicit shortest paths. The reason is that this eliminates the need for a distance matrix with shortest path costs between all task-pairs. A chromosome, as well as the trips of the underlying solution, are illustrated in figure 5.3a. Note that the resulting solution is equal to the one in figure 5.2a.



(a) Chromosome, genotype



(b) Corresponding solution, phenotype

Figure 5.3: Difference between our chromosome and solution.

5.2 Operators

In order to construct initial chromosomes and perform crossovers and mutations, we introduce two operators. The Task Removal Operator removes tasks from a giant tour and then repairs it by adding new shortest paths. The Path Insertion Operator uses a heuristic measure to find a suitable position to insert a path into a giant tour. Note that in this setting, a giant tour refers to a tour that does not adhere to capacity and time restrictions. A giant tour does not necessarily include all tasks in a network.

5.2.1 The Task Removal Operator

The purpose of the Task Removal Operator is to remove a set of tasks from a giant tour and then repair the giant tour to ensure connectivity after removal. The input of the operator is a giant tour and path containing at least one task. The network in figure 5.4 is used to explain the process. The thick, dotted lines, denoted S_1 - S_5, represent tasks.

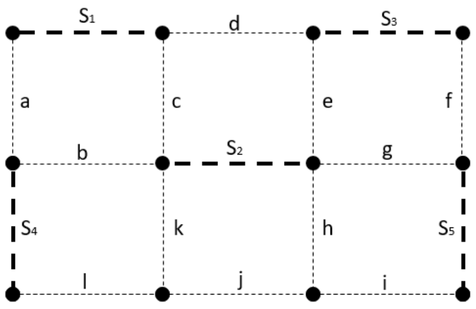
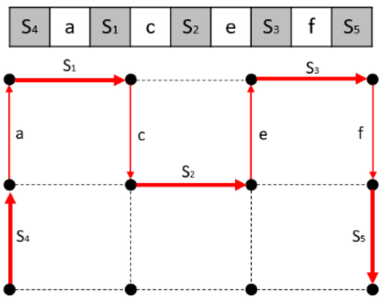


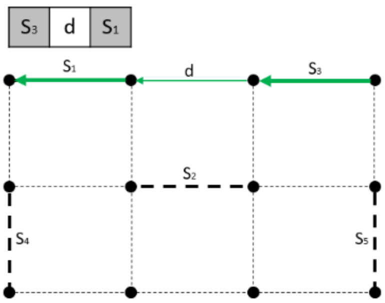
Figure 5.4: Underlying network

Figure 5.5 illustrates the process step by step. The string representations above the figures show the permutation of tasks at the given step of the process. The Task Removal Operator is used to remove the tasks found in the input path, colored green, from the giant tour, colored red. Figure 5.5a shows the giant tour before removal, whereas figure 5.5b shows the input path. The operator scans the giant tour and removes all tasks that are also in the input path from the giant tour. The traversals that connected removed tasks to their neighboring tasks in the giant tour are also removed, resulting in the unconnected giant tour in figure 5.5c. In the string representation of the giant tour the removed tasks and traversals are colored red. The operator scans the giant tour again and identifies unconnectivity.

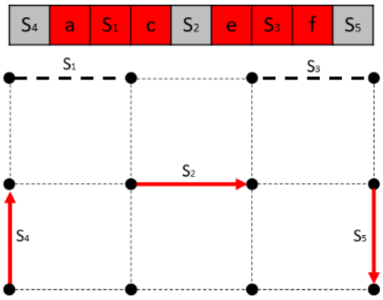
It then repairs the giant tour by adding new shortest paths between S_4 and S_2, and S_2 and S_5. The result is the tour shown in figure 5.5d.



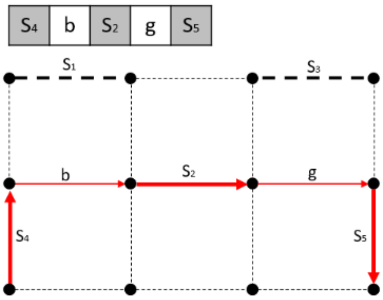
(a) Original giant tour



(b) Input path



(c) Duplicate tasks removed



(d) Connectivity restored

Figure 5.5: The Task Removal Operator

5.2.2 The Path Insertion Operator

The purpose of the Path Insertion Operator is to insert a connected path into a giant tour in the position where it adds the least cost to the tour. The input path can be inserted between any pair of consecutive tasks in the giant tour, or at the giant tour's start or end.

Since a giant tour services each task exactly once, all tasks that are present in both the input path and the giant tour are removed from the giant tour before insertion. The Task Removal Operator described in Section 5.2.1 is used for this purpose.

After removal of duplicate tasks, the operator scans the giant tour and evaluates the heuristic cost of inserting the input path at each possible position. For each pair of neighboring tasks, the sum of the Euclidean distance from both tasks to the start point and end point of the path is calculated. Furthermore, the heuristic cost of inserting the input path at the beginning and the end of the giant tour is calculated. The heuristic distance measure used for the start and end positions is the Euclidean distance from the task to the first or last task, multiplied by 2.

This is done to ensure that insertion at these positions are not favored to the other positions, as other positions are scored by the sum of the two distances to the neighboring tasks in the giant tour.

The sum of Euclidean distances is used as a cost measure rather than the actual cost of the shortest paths. The rationale for using this heuristic measure is that the Euclidean distance is easier to find than the shortest path between the start and end node of the input path and each pair of adjacent tasks in the giant tour.

When the point of least cost insertion is identified, the path is inserted at this point. This involves finding the shortest path between the two end nodes of the path, and the two neighboring tasks in the giant tour. If the input path does not contain any directed arcs, the path is inserted in the direction where it gives the least cost. Otherwise, the path is inserted in its original direction.

Figure 5.6 shows the insertion of a path containing one task, S_2, into a giant tour. Only a part of the giant tour is shown, for simplicity. The tasks in the giant tour are indicated in red. The blue, dotted lines show the Euclidean measurements made to decide the point of insertion. The insertion operator inserts task S_2, choosing its position based on the minimal sum of Euclidean distances to the tasks currently in the chromosome. The lowest sum of Euclidean distances is found when inserting the path between task S_1 and S_3, as figure 5.6c illustrates. The input path is therefore placed between task S_1 and S_3 in the giant tour.

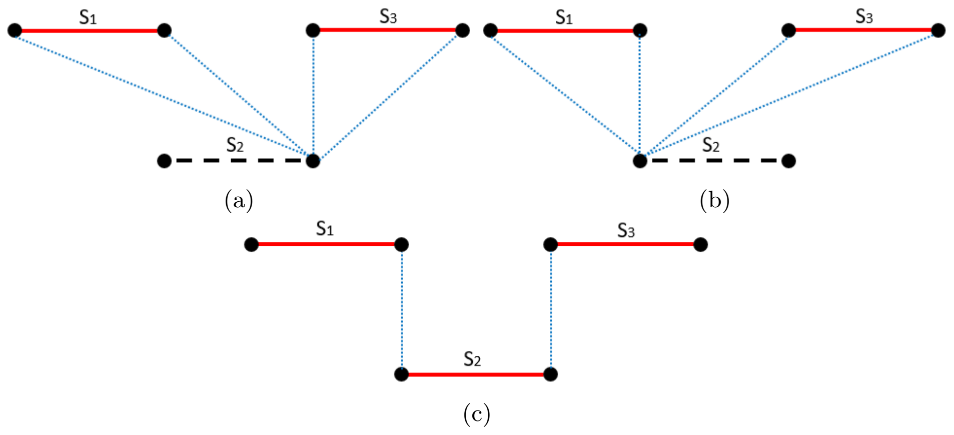


Figure 5.6: The Path Insertion Operator

5.3 Algorithmic overview

The proposed algorithm has four main phases: construction, selection and crossover, mutation and evaluation. The latter three phases constitute the evolutionary part of the algorithm. In figure 5.3 the algorithmic flow is illustrated.

First, the desired number of chromosomes is generated using the construction heuristic presented in Section 5.4. Then, the iterative part of the algorithm starts with the crossover phase, where parent chromosomes are randomly chosen and combined to create the next generation. In the evaluation phase, the fitness value of each chromosome in the new generation is calculated. The process is then repeated with the new generation until a termination criterion is met.

5.4 Construction heuristic

A common approach to create an initial population is to use a combination of construction heuristics and randomly generated chromosomes. The purpose is generally to find relatively good solutions within reasonable time, while simultaneously adequately covering the solution space with the randomly generated chromosomes. For instance, Lacomme et al. (2001) and Zhang et al. (2017) use this approach.

We propose a construction heuristic that generates chromosomes for the initial population of the EA. The construction heuristic has two main steps. The first step randomizes the order of a set containing all tasks, whereas the second part builds a giant tour based on the permutation of the tasks.

The first step of the process is to randomize the order of the task set. Figure 5.8 illustrates this step. In the figure, a set containing the five tasks labeled S_1 - S_5 is permutated randomly.

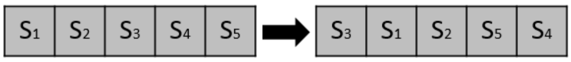


Figure 5.8: Permutation of tasks.

The next step builds the giant tour. The first task in the permuted set initializes the giant tour. The remaining tasks are chosen for insertion in the order given by the set, starting out with the second element. The tasks are inserted into the giant tour using the Path Insertion Operator presented in Section 5.2. This operator uses a simple heuristic based on Euclidean distances to decide where to insert each task.

The result of this process is a giant tour where the Euclidean distance between each consecutive task is minimized with respect to the permutation of the ini-

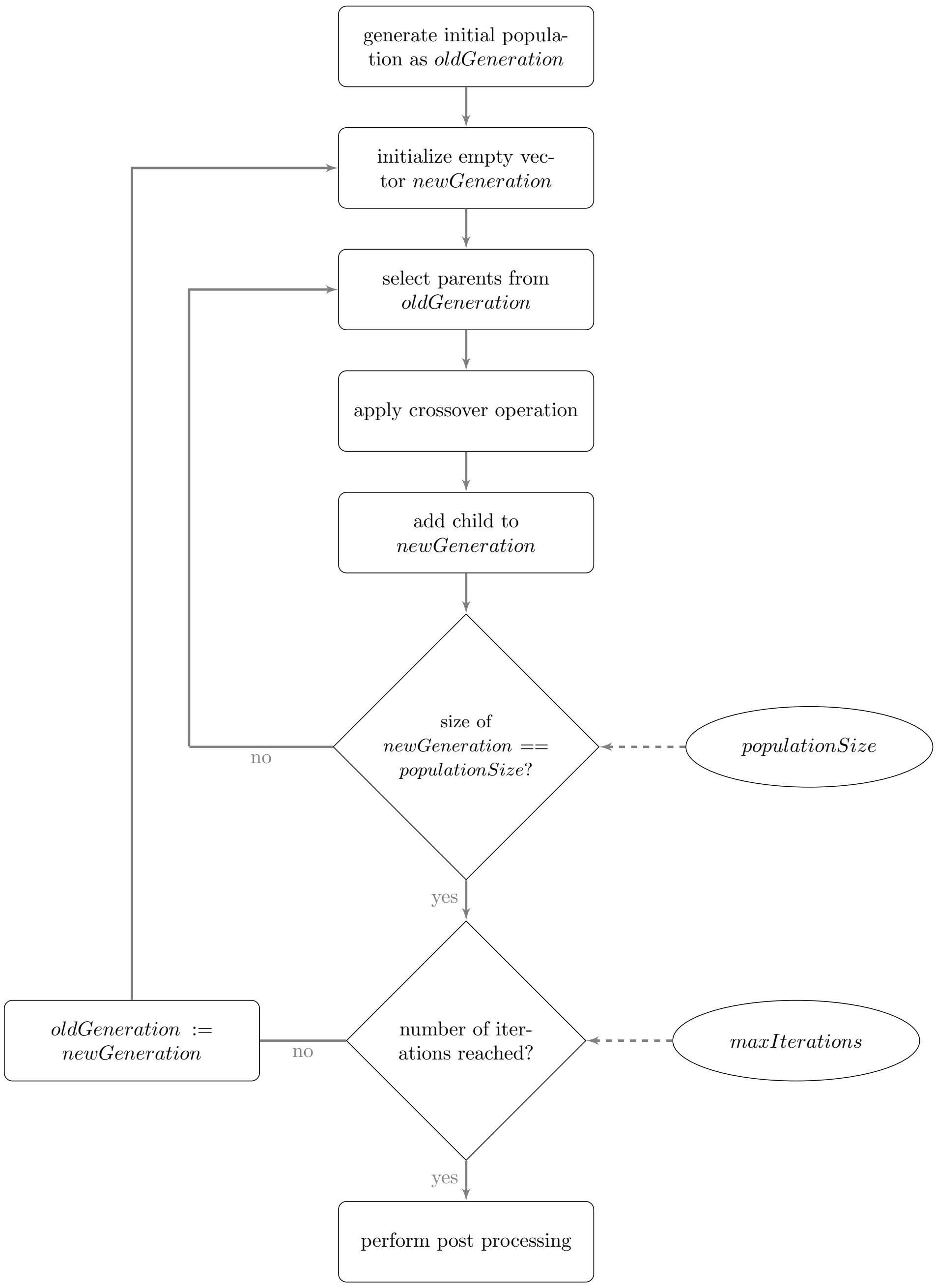

generate initial population as *oldGeneration*
initialize empty vector *newGeneration*
select parents from *oldGeneration*
apply crossover operation
add child to *newGeneration*
size of *newGeneration* == *populationSize*?
populationSize
no
yes
number of iterations reached?
maxIterations
oldGeneration := *newGeneration*
no
yes
perform post processing

Figure 5.7: Flow and main components of the EA.

tial randomized set. It is represented with explicit shortest paths between task, calculated by the Path Insertion Operator.

Lastly, the fitness score of the constructed chromosome is calculated using the splitting procedure which is presented in Section 5.6. The initial population is sorted after increasing fitness values, and the population is fed into the EA.

5.5 Fitness evaluation

The fitness score of each chromosome is given as the total sum of the trip durations in the corresponding solution. A lower fitness score, therefore, means higher fitness, as the objective is to minimize the total duration.

The fitness cannot be extracted directly from the genotype. Rather, the fitness is achieved by decoding the genotype into its corresponding phenotype. In other words, a decoding procedure is used to split the giant tours of the chromosomes into feasible trips and shifts. Then, the total duration of the resulting solution is calculated and used as fitness score for the chromosome. The procedure of splitting a giant tour into a feasible solution is presented in Section 5.6.

5.6 The splitting procedure

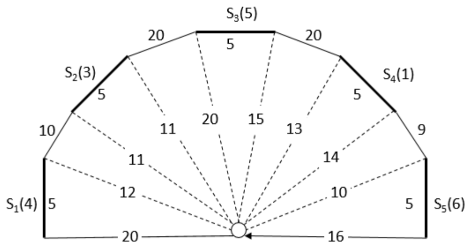
The purpose of a splitting procedure is to split a giant tour into a set of feasible trips in a way that optimizes the objective function.

Ulusoy's splitting procedure

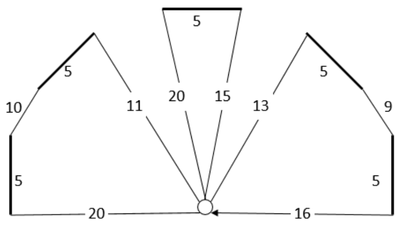
Ulusoy (1985) presents a splitting heuristic that finds the optimal split of a given giant tour. The giant tour, a complete graph with t tasks, is reduced by keeping only the depot and the t tasks. Thus, the reduced giant tour is represented simply by a permutation of the t tasks. A distance matrix of size $(t+1)\cdot(t+1)$ is pre-computed to know the cost of the shortest path between any two tasks, and between any task and the depot.

Figure 5.9a shows a giant tour in a network containing five tasks. The tasks, indicated by bold lines, are denoted S_1 to S_5. The numbers given in parenthesis are the demands associated with each task. All other numbers represent traversal costs.

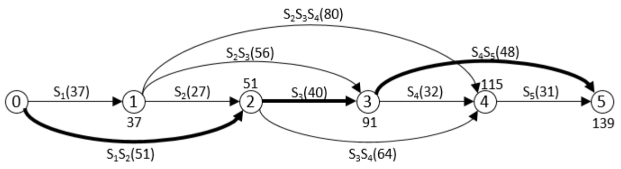
The splitting algorithm works by generating an auxiliary graph H with $t+1$ nodes, with the depot as node 0 and the tasks labeled from 1 to t, as depicted in figure 5.9c. The shortest path from the depot node 0 to t in H corresponds to the optimal split of the given chromosome.



(a) Giant tour with 5 tasks



(b) Resulting trips



(c) Auxiliary graph and resulting shortest path

Figure 5.9: Principle of split algorithm.

In figure 5.9c, each arc represents a trip servicing the tasks the arc is denoted with before heading back to the depot. For instance, the arc labeled $S_1(37)$ represents a trip servicing only task S_1 before heading back to the depot. This trip has a cost of 37. Furthermore, the arc labeled $S_1S_2(51)$ represents a trip of cost 51 servicing both S_1 and S_2 before heading back to the depot. The optimal spit is indicated by the bold arcs in the graph in figure 5.9c and illustrated in figure 5.9b.

Prins et al. (2009) present an efficient implementation of Ulusoy's splitting algorithm without generating H explicitly. With pre-computed distances, the complexity of this implementation is $O(t)$. Neither Prins et al. (2009) nor Ulusoy (1985) consider cost restrictions on sets of trips, such as shift durations, in their splitting procedures. However, Prins (2004) applies a similar split procedure considering cost constraints on a capacitated VRP.

Our procedure

The proposed splitting algorithm is based on the split procedure presented in Prins (2004). Our split procedure is modified to work with explicit shortest paths between tasks rather than a pre-computed distance matrix. To make the implementation efficient, however, the distance from each node in any task to the depot is pre-computed and stored as an attribute to each task node.

The pseudocode for our implementation is presented in algorithm 1. Here, node 0 is the depot node. For each of the t tasks, S_1 to S_t, the split algorithm keeps two labels: $C(S_j)$ representing the cost of a shortest path from 0 to S_j adhering to the time and capacity constraints, and $P(S_j)$, which is the predecessor of S_j

on this path. D_{S_{j-1},S_j} is the cost of the shortest path between two consecutive tasks, $w(S_i)$ is the cost of task S_i and $q(S_i)$ is the demand in task S_i. The splitting algorithm operates on a set of tasks, while the costs of shortest paths between tasks are extracted from the complete chromosome.

The main loop of the algorithm iterates over the set of tasks. For each task S_i, the algorithm forward iterates over the next tasks S_j, updating $C(S_j)$ if the path with a trip starting with S_i improves $C(S_j)$'s value. j is incremented until either the capacity limit Q or the time limit L is reached, or until the iterator reaches the end of the set.

The cost D_{S_{j-1},S_j} between two neighbouring tasks S_{j-1} and S_j is found by scanning the complete chromosome, until S_{j-1} is located. Then, iterating further, the cost of the traversal on the current position of the iterator is added to D_{S_{j-1},S_j} for each iteration until S_j is reached.

In figure 5.9, $V = \{0, 37, 51, 91, 115, 139\}$ and $P = \{0, 0, 0, 2, 2, 3\}$. The path constituting the optimal split has a cost of 139, and can be extracted by backtracking parents from task t to node 0.

Algorithm 1 *Split*-algorithm

```
1: $C_0 := 0$
2: $P_0 := 0$
3: for $i := 1$ to $t$ do $C_i := \infty$
4: end for
5: for $i := 1$ to $t$ do
6:     $j := i$
7:     $load := 0$
8:     repeat
9:         $D_{S_{j-1},S_j}$ := GetDistanceBetweenTasks($S_{j-1}$,$S_j$)
10:        $load := load + q(S_j)$
11:        if $i = j$ then
12:            $cost := D(0, S_i) + w(S_i) + D(S_i, 0)$
13:        else
14:            $cost := cost - D(S_{j-1}, 0) + D(S_{j-1}, S_j) + w(S_j) + D(S_j, 0)$
15:        end if
16:        if $(load \leq Q)$ and $(cost \leq L)$ then
17:            if $(C_{i-1} + cost < C_j)$ then
18:                $C_j := C_{i-1} + cost$
19:                $P_j := i - 1$
20:            end if
21:        end if
22:        $j := j + 1$
23:    until $(j > t)$ or $(load > Q)$ or $(cost > L)$
24: end for
25: // Extract distance between subsequent tasks
26: procedure GetDistanceBetweenTasks($S_a$,$S_b$)
27:     $D_{S_a,S_b} := 0$
28:     $path \leftarrow$ complete chromosome
29:     $k := 0$
30:     while $path(k) \neq S_a$ do
31:         $k := k + 1$
32:     end while
33:     $k := k + 1$
34:     while $path(k) \neq S_b$ do
35:         $D_{S_a,S_b} := D_{S_a,S_b} + w(path(k))$
36:         $k := k + 1$
37:     end while
38:     return $D_{S_a,S_b}$
39: end procedure
```

5.7 Selection and crossover

In the selection phase, parents are chosen to create offspring for the new generation. In a network routing context, the binary tournament approach is widely used for this purpose. In a binary tournament, two chromosomes are selected randomly, and the one with the best fitness score is selected as the first parent. The second parent is selected in the same way. The process is repeated until the required number of parent-pairs is reached. This approach is used by for instance Baker and Ayechew (2003) and Prins (2004) for the VRP, and by Lacomme et al. (2004a) for the CARP.

The crossover phase produces offspring for the next generation by combining the parent chromosomes. This can be done in many different ways. The most notable in a routing setting include 2-point crossover (Baker and Ayechew (2003)), order crossover, and linear order crossover (Oliver et al. (1987)).

In a 2-point crossover, two points are chosen at random in the parent chromosomes. This splits the parent chromosomes into three parts. The first and last part of Parent 2 are combined with the second part of Parent 1, to create the first child. The second child is created by reversing the parent roles. The two-point crossover is illustrated in figure 5.10.

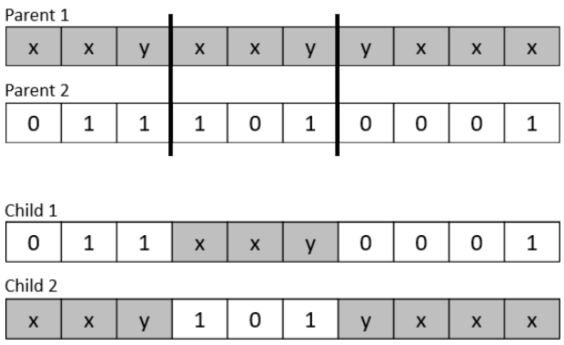


Figure 5.10: 2-point crossover

The most widely used crossovers for EAs solving arc routing problems, are the Order Crossover (OX) and Linear Order Crossover (LOX). These procedures are used in Prins (2004), Lacomme et al. (2004a) and Xing et al. (2010), to name a few. Prins and Bouchenoua (2005) note that the LOX is suitable for linear chromosomes that have a definite beginning and end, whereas the OX is apt for circular chromosomes.

LOX works in the following manner: Let P1 and P2 denote the two parent strings. Similar to two-point crossover, two split points, i and j, where $i \leq j$, $i \geq 0$ and j is smaller than the size of the string, are chosen randomly. P2(i) to P2(j) is copied into the first child, C1, at positions i to j. Now C1(i) to C1(j) is identical to P2(i) to P2(j). Then, P1 is scanned, and any element in P1 that is not already in C1 is added to the first position not yet assigned in C1. The roles are switched and the process repeated to create the second child, C2. The LOX procedure is illustrated in figure 5.11. Here, $i = 3$ and $j = 6$. The strings between position 3 and 6 is copied from P1 to C1 and P2 to C2 respectively.

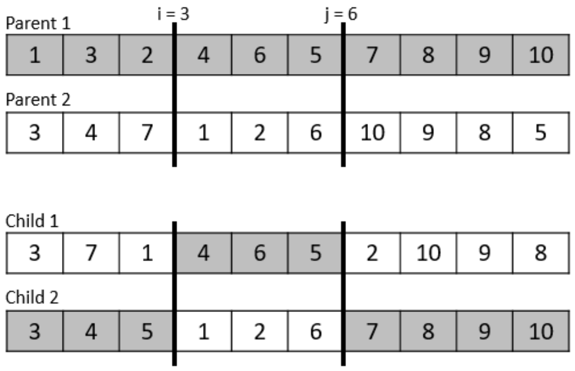


Figure 5.11: Linear Order Crossover (LOX)

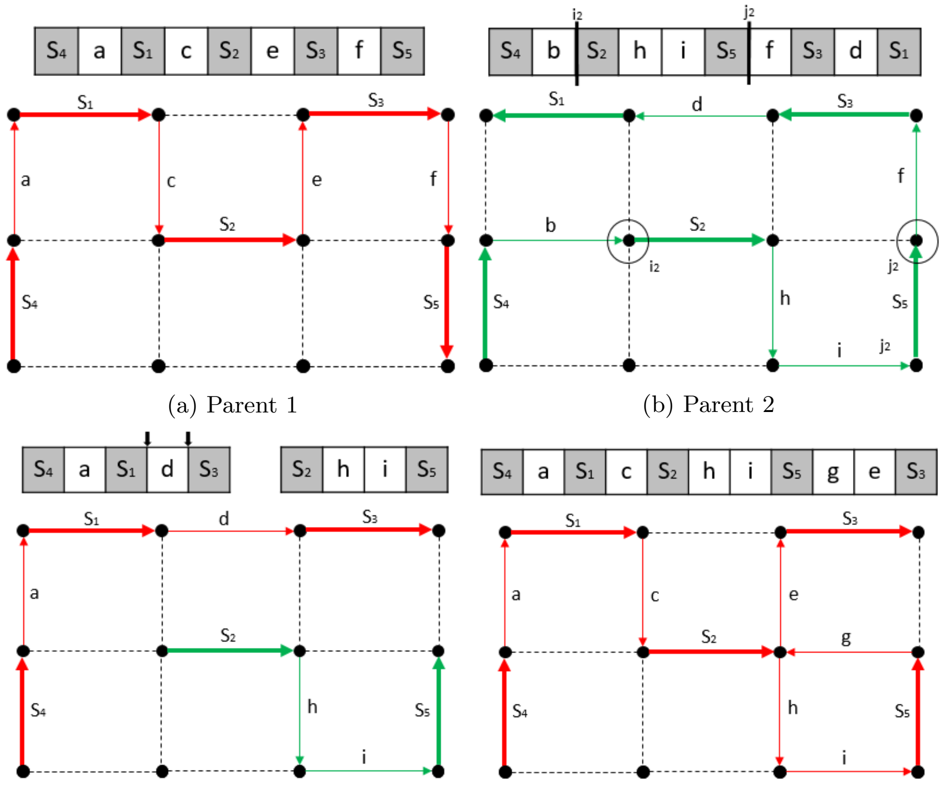
Our approach

The selection process in our EA is two-fold. First, a set of *elite chromosomes* are chosen. Elite chromosomes are the chromosomes with the best fitness scores in a population. These chromosomes are copied, and the copy is passed on to the next generation without going through the crossover phase. Second, using binary tournament, the appropriate number of parent pairs are selected. A chromosome can be chosen as a parent multiple times in each generation.

For the crossover operation, two points in each of the parents in a parent pair are randomly selected. This practice is different from most crossover operations prevalent in the literature, such as the LOX, where the same points are used for both parents. The reason for selecting different points for each parent is that these chromosomes consist of both tasks and the explicit shortest paths between them, as opposed to chromosomes only containing tasks. This chromosome representation allows the chromosomes to be of different lengths. Thus, a point selected for one of the chromosomes might be out of range for the other. The generated points are referred to as i_1 and j_1 for Parent 1, and likewise i_2 and j_2 for Parent 2.

Figure 5.12 shows how two parents are combined to generate the first child. The second child is created by repeating the process with reversed parent roles.

Figures 5.12a and 5.12b show the string and graph representations of the two parents, as well as the crossover points i_2 and j_2 for Parent 2. First, the giant tour of Parent 1 is copied into the child. Thereafter, the Path Insertion Operator presented in Section 5.2.2 is applied to insert the path P2(i_2) to P2(j_2) into the child chromosome. The tasks present in the input path are removed from the chromosome, and the chromosome is repaired by finding new shortest paths between the remaining unconnected tasks S_1 and S_3. Figure 5.12c shows the state of the giant tour after the duplicate tasks have been removed and the giant tour has been repaired, as well as the path to be inserted. The black arrows in the string representation of Parent 1 point to the positions identified as the points of least cost insertion. The operator inserts the input path from Parent 2 between these points to create the offspring Child 1. The string and graph representations of Child 1 is shown in figure 5.12d.



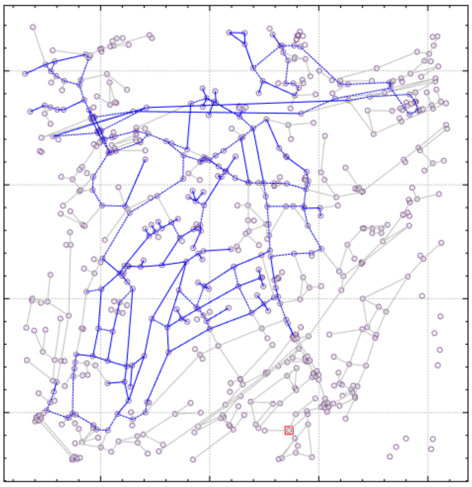
(a) Parent 1 (b) Parent 2

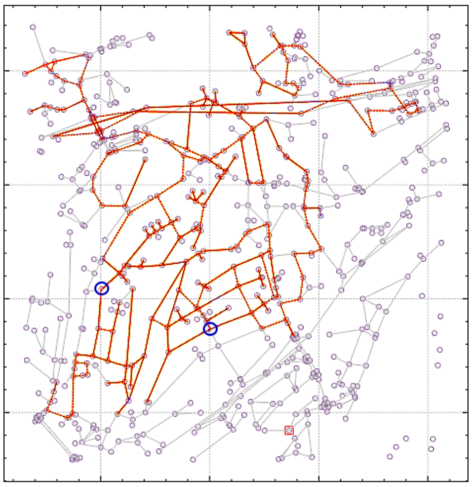
(c) Constructing child 1 (d) Child 1

Figure 5.12: The crossover mechanism.

The purpose of inserting the copied path at the point of least additional heuristic costs, is to make the crossover procedure more aggressive in improving fitness than crossover operations where paths are inserted at random points.

Figure 5.13 shows the crossover operation applied on a small part of Oslo. The colored lines in figures 5.13a and 5.13b show the giant tour of each parent. The path between the blue marks in figure 5.13b is copied and inserted in the offspring, as illustrated in figure 5.13c.


(a) Parent 1


(b) Parent 2

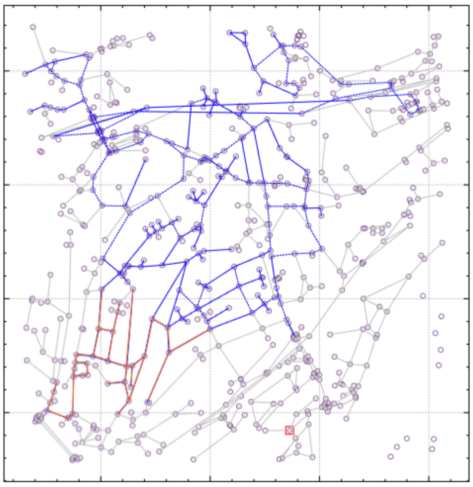

(c) Child 1

Figure 5.13: The crossover mechanism.

5.8 Mutation

The simplest mutation procedures include operators that move or swap the position of certain tasks. More advanced mutation procedures are based on local search techniques. Lacomme et al. (2001) argue that using local search procedures as mutation operators is more efficient than the simple, random mutations.

For the proposed EA, three mutation operators are implemented and tested. Because of the size and complexity of the problem, the improvement of chromosomes during mutation is emphasized. Hence, the mutations are accepted only if the fitness of the chromosome is improved. Two of the operators are informed search heuristics aimed at improving the fitness of chromosomes, while the last operator applies a random swap mutation.

The first mutation procedure uses the Path Insertion Operator presented in Section 5.2.2 for a single task. A random task is thus removed from the chromosome and reinserted at the point of least heuristic cost.

The second mutation procedure also utilizes the Path Insertion Operator. However, in this mutation, the path between two random points in the chromosome is removed from the chromosome, before it is reinserted into the chromosome at the point of least heuristic cost.

The last mutation operator removes two random tasks from the giant tour and swap their positions, before calculating new paths to repair the now unconnected giant tour.

Tests performed on the mutation operators show that none of the proposed procedures were able to improve fitness of the chromosomes. The fitness values of the resulting chromosomes after mutation were in the best cases as good as the fitness values before mutation. Additionally, the mutation operators caused increased iteration times due to the added workload. As such, the mutation operation was omitted when the results presented in chapter 7 were generated.

5.9 Post-processing

The purpose of the post-processing phase is to convert the output of the EA into a valid routing plan for a real-world waste collection problem. The EA finds and outputs a set of feasible shifts, and the shifts are distributed on days, vehicles and teams in the post processing.

A routing schedule is generated by starting at Shift 1 in the output from the EA, assigning the current shift to the first day, vehicle and team available for assignment. The process is repeated for the next shifts, until all shifts are assigned.

By assigning shifts in the order of which they are created by the EA, one achieves a flexible routing plan where areas in geographical proximity are serviced close in time. However, by assigning shifts to the first free team, vehicle and day, one can achieve an unbalanced schedule if there are more vehicles and personnel available than strictly needed. If so, the schedule may need to be balanced out after assignment.

Chapter 6

Case Description

The algorithm was run on two types of instances: widely studied benchmark instances from the literature, and instances generated based on the real-world infrastructure of Oslo. The benchmark instances are introduced in Section 6.1 whereas Section 6.2 introduces the set of real-world instances and explains how they are generated.

6.1 Benchmark instances

Three sets of instances are used for benchmarking the EA. The three sets are subsets of the GDB benchmark set introduced by Golden et al. (1983), the EGL benchmark set provided by Li (1992) and Li and Eglese (1996), and the BMCV benchmark set introduced by Beullens et al. (2003). Optimal values or tight lower and upper bounds are available for all the instances. The instances, their descriptions, and results obtained by other researchers are available online at http://logistik.bwl.uni-mainz.de/benchmarks.php. The larger CARP instances introduced by Kiilerich and Wøhlk (2018) were not used, as the bounds for these instances are not known.

Tables 6.1, 6.2 and 6.3 present the characteristics of the benchmark instances. In the tables, N is the number of nodes in the network, A is the total number of arcs and R is the number of required arcs. The next two columns give the fraction of required arcs to the total number of arcs and the optimal value for the respective instance. If the optimal value is unknown, the upper and lower bounds are given instead.

Table 6.1: GDB instances

	N	A	R	Density	Optimal value	LB	UB
GDB 1	12	22	22	1.00	316	-	-
GDB 4	11	19	19	1.00	287	-	-
GDB 7	12	22	22	1.00	325	-	-
GDB 14	7	21	21	1.00	100	-	-
GDB 15	7	21	21	1.00	58	-	-
GDB 17	8	28	28	1.00	91	-	-
GDB 19	8	11	11	1.00	55	-	-

Table 6.2: EGL instances

	N	A	R	Density	Optimal value	LB	UB
egl-e1-A	77	98	51	0.52	3 548	-	-
egl-e3-A	77	98	87	0.89	5 898	-	-
egl-s1-A	140	190	75	0.39	5 018	-	-
egl-s2-C	140	190	147	0.77	16 425	-	-
egl-s3-C	140	190	159	0.84	17 188	-	-

Table 6.3: BMCV instances

	N	A	R	Density	Optimal value	LB	UB
C01	69	98	79	0.81	-	4 145	4 150
C08	66	88	63	0.72	4 090	-	-
C17	43	56	42	0.75	3 555	-	-
D15	97	140	107	0.76	3 990	-	-
E07	73	98	44	0.45	4 155	-	-

Adaption for benchmark instances

The proposed EA uses Euclidean distances both to efficiently calculate shortest paths to find low-cost insertions of paths in the crossover phase. To the best of our knowledge, a benchmark set for the CARP containing node coordinates does not exist. Thus, to enable the algorithm to solve instances without available heuristic costs, the EA is modified to use exact costs for these purposes. As such, for the benchmark instances, all costs are calculated using the shortest path algorithm introduced by Dijkstra (1959).

6.2 Real-world instances

The EA was applied to four real-world instances generated based on the infrastructure of Oslo. Table 6.4 summarizes the characteristics of the instances, while figure 6.1 illustrates their domains. In the figure, the red circle shows the location of the depot. The RW0 instance represents all of Oslo.

Table 6.4: Real-world instances

	N	A	R	Density	Pickup locations
RW1	1 058	1 252	159	0.13	1 569
RW2	3 107	4 066	553	0.14	4 223
RW3	6 202	8 601	1 647	0.19	9 003
RW0	23 577	31 160	7 282	0.23	46 208

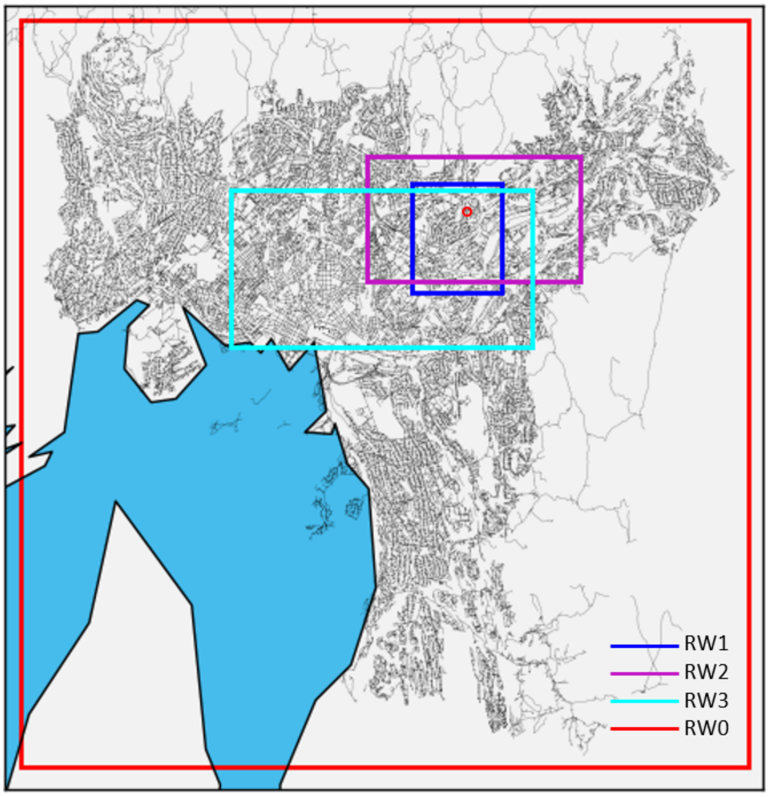


Figure 6.1: Domains of the real-world instances.

6.2.1 Generating the instances

The real-world instances are generated by combining data from two sources - container data and GIS data. Detailed GIS data for Oslo was retrieved from the Norwegian Mapping Authority (NMA), Norway's national mapping agency. The GIS files contain detailed information about the road network in Oslo, including coordinates and names of streets, intersections and street addresses. Container data was supplied by REN. This data includes coordinates for the pickup locations and the number of containers at each location as well as their volume.

In the GIS files, many streets between two intersections are split into multiple consecutive segments. To simplify the network, such segments are merged into a single segment connecting the two intersections. By merging such segments, the number of arcs in the resulting network is reduced from about 84 000 to approximately 70 000.

Containers are assigned to arcs using a two-stage process. First, the appropriate street is identified by matching the street name and the address of the container. Thereafter, each container is assigned to the street segment that minimizes the ratio $\frac{d_{ib}+d_{jb}}{d_{ij}}$. Here, d_{ib} denotes the Euclidean distance from the container to the start node of the segment, d_{jb} is the Euclidean distance from the container to the end node of the segment and d_{ij} is the distance between the start and end nodes of the segment.

Figure 6.2 shows how a container is assigned to a street segment. From the street address of the container, it is known that the container belongs to the street colored red, either segment (n_1,n_2) or segment (n_2,n_3). The assignment ratios are given as $\frac{d_{n_1B}+d_{n_2B}}{d_{n_1n_2}}$ and $\frac{d_{n_2B}+d_{n_3B}}{d_{n_2n_3}}$, respectively. As B is closer to segment (n_2,n_3) than to segment(n_1,n_2), this is the segment with the lowest assignment ratio, and thus the segment to which the container is assigned.

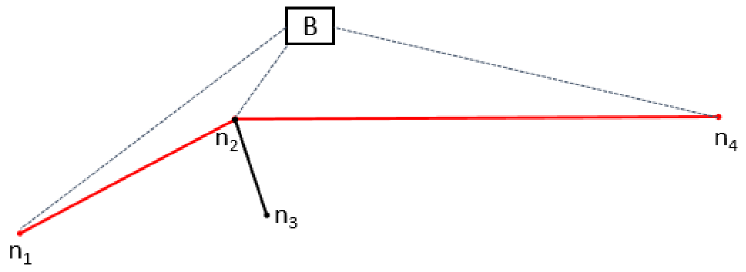


Figure 6.2: Illustration of the container assignment procedure.

For some street segments, meandering is not allowed. To ensure that these are serviced in both directions, these street segments are replaced by two oppositely directed one-way street segments. Containers are assigned to one of the two segments depending on which side of the road they are located.

The original GIS data contains a significant number of small dead-ends. Many of these dead-ends represent driveways which the waste collection trucks do not enter. Therefore, all dead-ends with a length less than 100 meters are identified. If such a dead end is connected to exactly two other arcs, the dead end is removed. Thereafter, the two remaining segments are merged, before the containers initially assigned to the dead end are reassigned to the resulting, merged segment. This operation more than halves the number of arcs in the network, to about 31 000. The length limit of 100 meters represents a trade-off between retaining a realistic representation of the network and reducing it's complexity.

The process of removing such dead ends is illustrated in figure 6.3. The blue segment in figure 6.3a is a dead end with a length of less than 100 meters. The orange segment represents a required arc, while the black segment has no demand. The dead end is removed (figure 6.3b), before the two remaining segments are merged into one (figure 6.3c). The resulting arc has a demand equal to the waste associated with all of the three original segments.

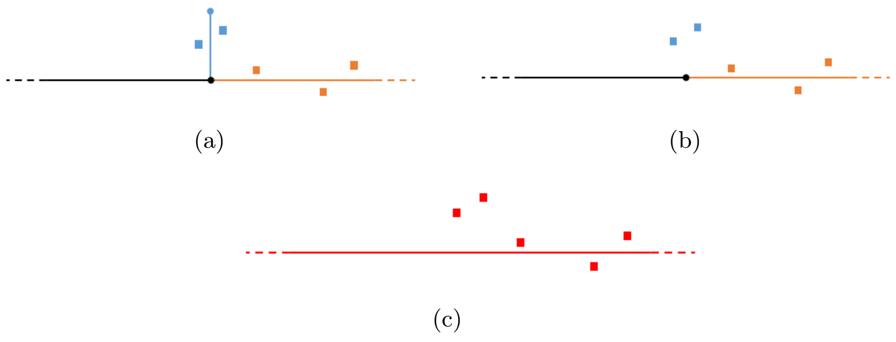
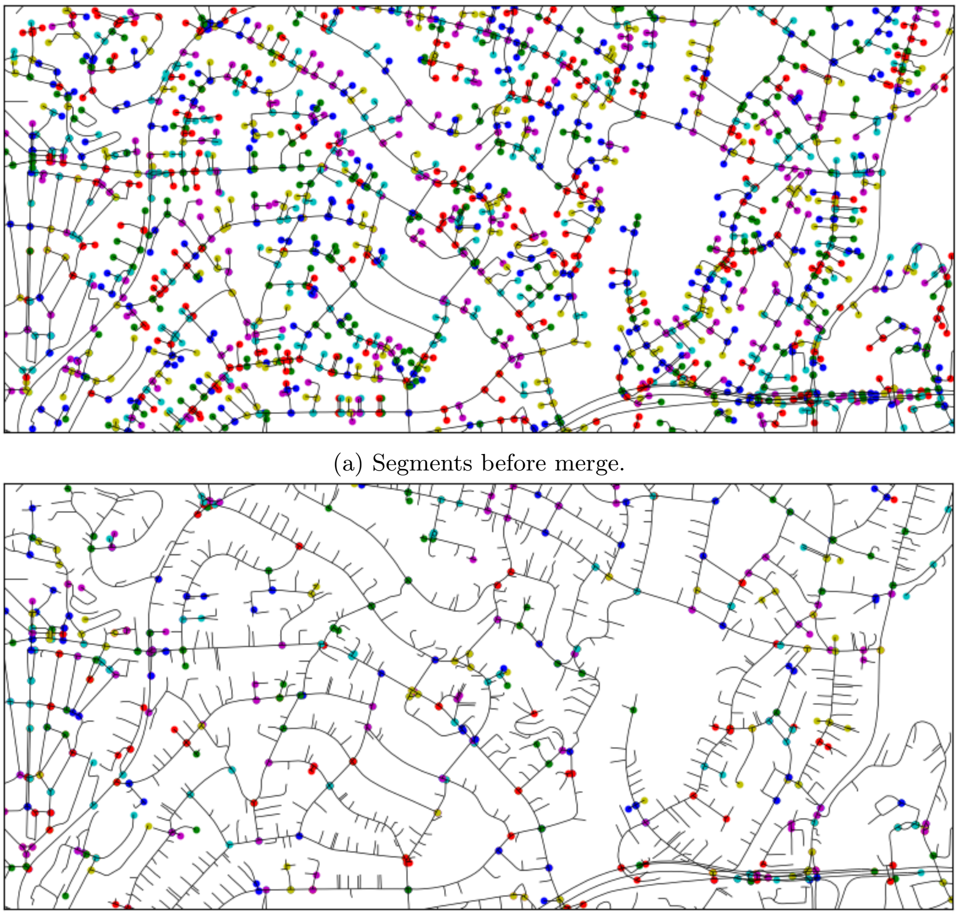

(a) (b) (c)

Figure 6.3: Aggregation of dead ends.

Figure 6.4a shows a small section of Oslo before the alterations of the network, while 6.4b shows the same section after the alterations. The colored nodes represent the ends of segments in the network. Thus, the figure shows how the network is reduced through aggregation of dead ends and merging of consecutive segments.

The real-world instances are generated by providing coordinates for the north-eastern, and south-western corners of rectangles encompassing parts of Oslo of different sizes. All the generated instances contain the depot.

To ensure a connected network in each of the generated instances, Djikstra's algorithm is applied to find the shortest path from each arc in the network to the depot. If the algorithm does not find a path to the depot for a given arc, it means the arc is unconnected. These unconnected arcs are removed from the network.


(a) Segments before merge.

(b) Segments after merge.

Figure 6.4: A section of Oslo before and after the merge and removal of segments.

6.2.2 Input parameters

For the real-world instances, all containers are assumed to contain waste volumes equal to their intrinsic volume. Furthermore, the traversal time of segments while deadheading is approximated according to $d_{ij}/10$, where d_{ij} is the segment length. Similarly, the servicing time for each segment is calculated according to $n_{ij} \cdot 45$, where n_{ij} is the number of bins along segment ij, and 45 is the assumed number of seconds it takes to empty a bin. A vehicle has a volume of $23m^3$, and a shift has a duration of 7.5 hours. These run parameters are summarized in table 6.5.

Table 6.5: Real-world instance parameters

Vehicle speed (deadheading)	10 m/s
Pickup time per bin	45 s
Vehicle capacity	23 000 L
Time limit per shift	27 000 s

Chapter 7

Computational Study

In this Chapter, the performance of the EA is studied in detail by running it on 17 different benchmark instances. The rationale for this is that these instances have known bounds or optimal values, and can bring insight into the potential and limitations of the proposed algorithm. The insight gained from the benchmark analysis is then used to assess the results obtained by the EA on the four real-world instances.

The chapter is organized in the following way: Section 7.1 presents the implementation of the algorithm. In Section 7.2, the effect of different population sizes and share of elite chromosomes are investigated. Thereafter, the benchmark results are discussed and analyzed in Section 7.3. Lastly, Section 7.4 summarizes and discusses the results obtained by the EA on the real-world instances.

7.1 Technical specifications

The EA is written in C++, and the implementation is parallelized to enhance performance. The parallel processes are run on a cluster, and a Message Passing Interface (MPI) is used to coordinate the parallelization. Both the construction of each chromosome and each crossover operation are parallelized to run on separate cores. The specifications of the cluster nodes are presented in table 7.1.

Table 7.1: Specifications of cluster nodes

OS	Linux
CPUs	40 x 2.40 GHz
RAM	92.91 GB

Figure 7.1 shows how the workload is distributed between the coordinating process and the worker processes for a population of six chromosomes with two

elite chromosomes. $p0$ represents the coordinating process, while $p1$ to $p6$ represent the worker processes. The solid segments of a process show that the process is active in the current stage, while the dashed segments indicate that the process is inactive, i.e., waiting for input from another process to continue. The green lines represent information being shared between processes, in the direction indicated by the arrow. The red arrow indicates the section which is repeated until the termination criterion of the EA is met.

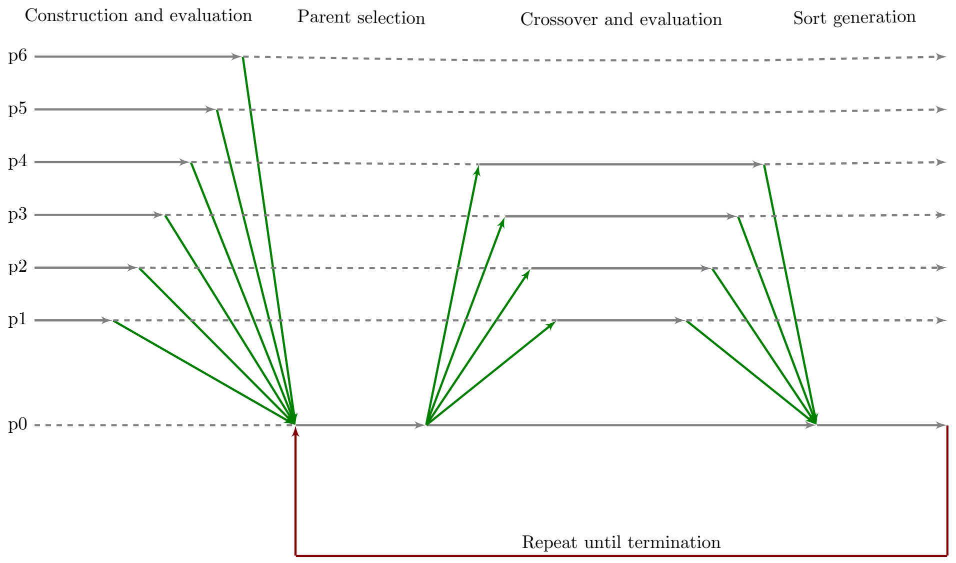


Figure 7.1: Work flow of the processes.

In the construction phase, each of the worker processes generates and evaluates a chromosome for the initial population. These chromosomes are passed to $p0$, which gathers the chromosomes in a population. Thereafter, $p0$ selects two parents for each crossover operation and passes these to the worker processes. While the respective worker processes apply the crossover, $p0$ adds the elite chromosomes from the old generation to the new generation. The worker processes perform the crossover, evaluate the offspring chromosomes and pass the fittest offspring back to $p0$. $p0$ adds the offspring to the new generation and sorts the generation. If the termination criterion is met, the program terminates. Otherwise, the new generation is used as the basis for a new parent selection and crossover.

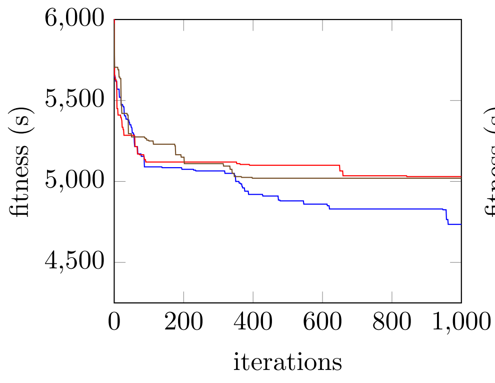
7.2 Run parameters

The EA was tested with varying run parameters on the benchmark instance C01 and RW1 instances, to analyze the impact of the population size and fraction of elite chromosomes. The purpose of the assessment was to decide what population composition should be applied in the further analysis. The EA was tested with five different population, namely 20, 60, 100, 200 and 300. For each population size, tests were run with 20% elite chromosomes, 30% elite chromosomes and 40% elite chromosomes. The tests were conducted with 1000 iterations on the benchmark instance and 200 iterations on the real-world instance.

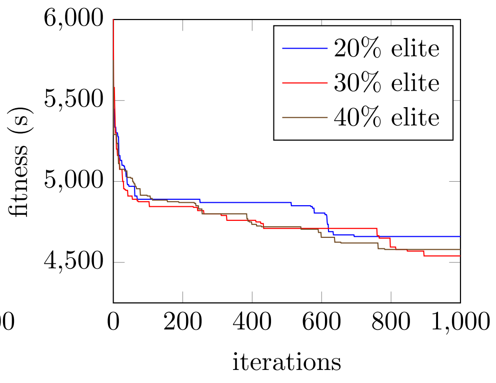
The results from the benchmark test are presented in Section 7.2.1, while the results from the real-world test instance are presented in Section 7.2.2.

7.2.1 Benchmark tests

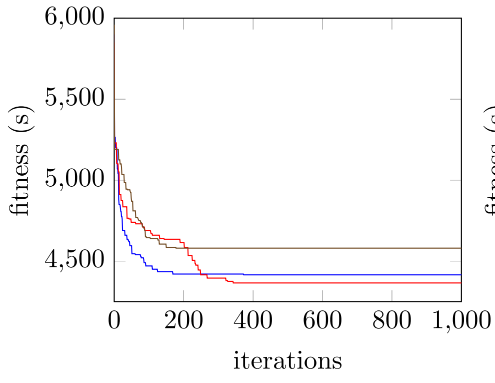
Figure 7.2 shows the fitness score of the best individual in each generation for the test runs on the C01 instance.



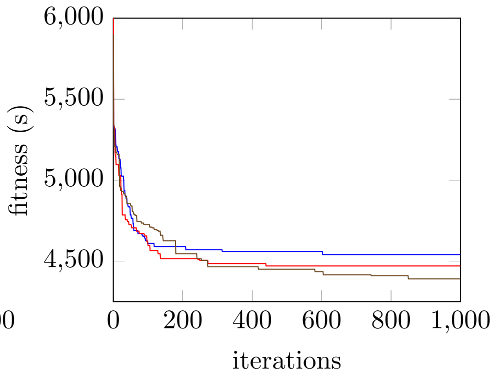
(a) Population size = 20



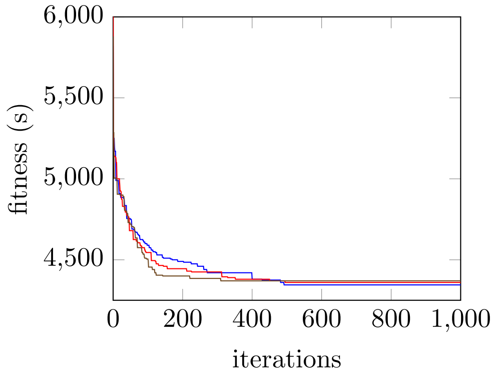
(b) Population size = 60



(c) Population size = 100



(d) Population size = 200



(e) Population size = 300

Figure 7.2: Evolution for the C01 instance.

Figure 7.2 shows that composition with the 20% elite chromosomes performed far better than the other compositions for a population size of 20. However, the 20% composition is the worst performer for the population size of 60. The 30%

elite composition outperforms the other compositions for the population sizes of 60 and 100, while the 40% composition is the worst performing with a population size of 100. For the population size of 200, however, the 40% composition is superior. As the population size grows to 300, there is no significant difference between the performance of the three compositions.

Studying the graphs of figure 7.2, it seems that for the smallest population, the compositions with a low fraction of elite chromosomes are superior. As the population size increases, however, the compositions with a larger fraction of elites perform better. This indicates that the optimal fraction of elites is dependent on the population size.

A higher fraction of elite chromosomes means more chromosomes are passed on unaltered to the next generation, and that fewer crossovers are performed. This implies that a higher share of elite chromosomes gives less diversity. Small populations are less diverse than large ones, and thus, a low fraction of elite chromosomes is favorable for these instances. For the larger, more diverse population of 300, however, this effect is not observed.

The results show a negative correlation between population size and fitness. As population sizes increase, the fitness score of the best chromosome in each generation decreases, yielding better results.

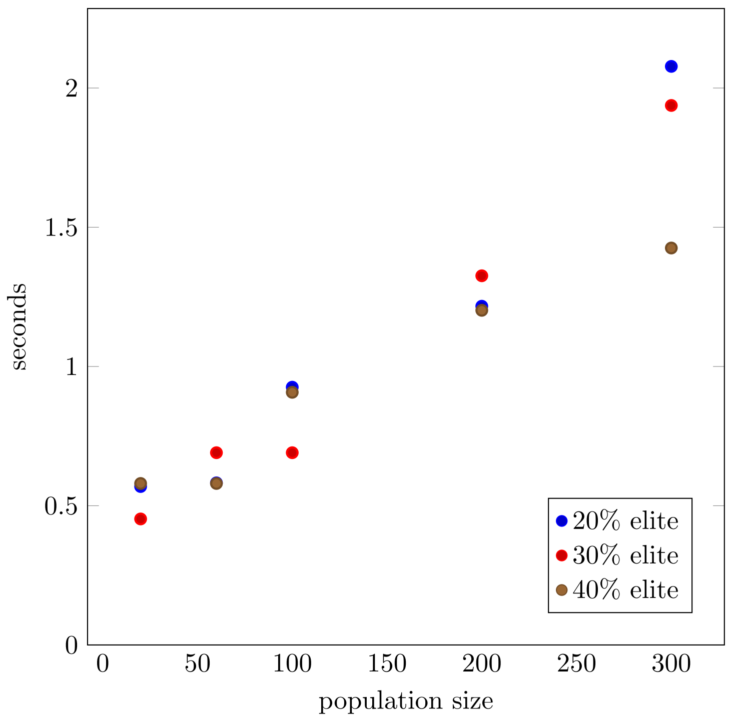


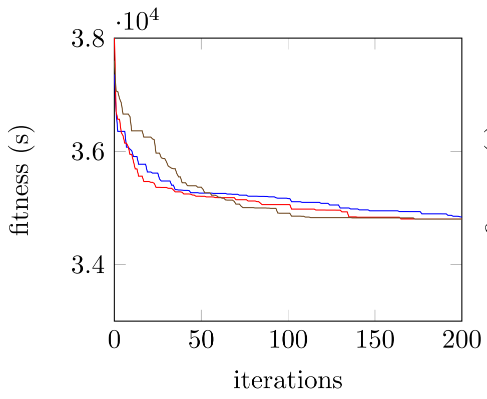
Figure 7.3: Mean iteration times for the tests.

Figure 7.3 shows the average duration of the iterations for each test run. Since the algorithm is run on a cluster where nodes can be shared, the workload on these may vary, and hence the CPU time may vary between runs. However, studying the duration of each run gives indications of actual performance.

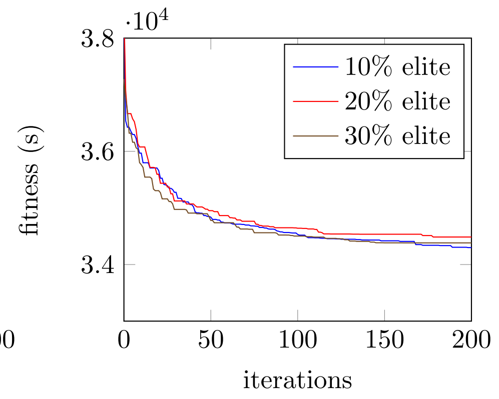
The plot indicates a trend of increasing iteration time with increasing population size. Furthermore, it seems that the increase in iteration time from one population size to the next is larger for populations with a low fraction of elites. This trend reflects the fact that all crossover operations are performed in parallel, and that the duration of this operation equals the time it takes for the most calculation intensive crossover operation, which does not depend directly on the population size. The findings imply that time limitations may force a trade-off between population size and the number of iterations one can conduct.

7.2.2 Real-world instance tests

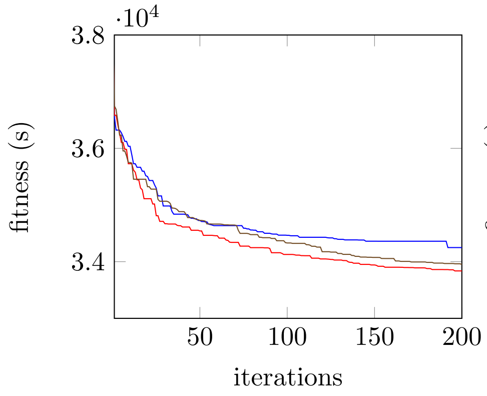
The impact of the run parameters was also tested for the RW1 instance, to examine the behavior of the EA on a network more resembling the one representing Oslo. RW1 is significantly larger than C01 and has a lower fraction of required arcs.



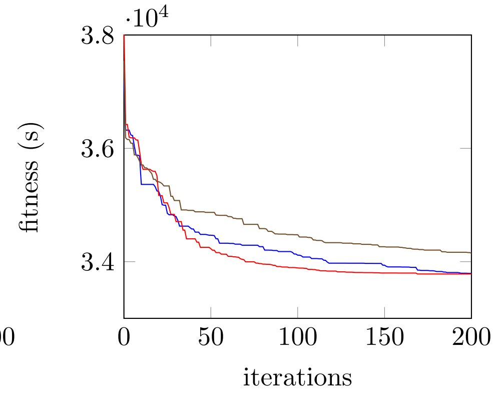
(a) Population size = 20



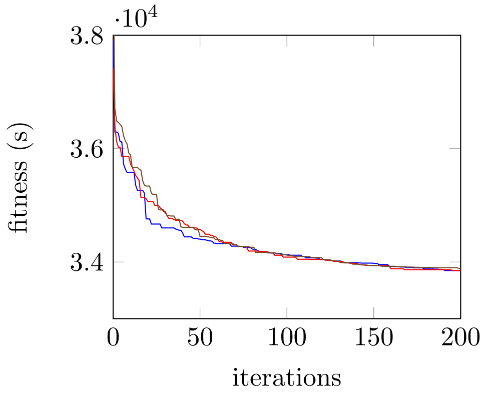
(b) Population size = 60



(c) Population size = 100



(d) Population size = 200



(e) Population size = 300

Figure 7.4: Evolution for the RW1 instance.

Figure 7.4 illustrates the results for the runs on the RW1 instance. Compared to the test run on C01 in Section 7.2.1, the share of elite chromosomes seems to have less impact for the smallest population sizes of 20 and 60. However, similarly to the C01 test, there is a significant difference for the population sizes of 100 and 200, and the effect of the fractions of elite chromosomes seems to diminish entirely with a population size of 300.

Regarding population size, the same trend appears for RW1 as for C01. Increasing the population size leads to a reduction in the fitness score and better solutions.

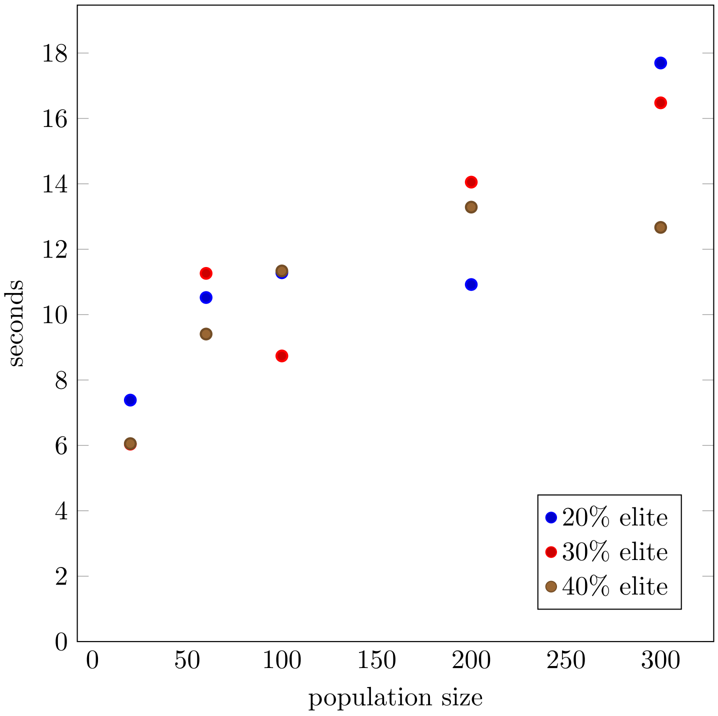


Figure 7.5: Mean iteration times for the test runs on the RW1 instance.

Figure 7.5 shows the mean iteration times for the runs on RW1. The mean iteration times are in line with the discoveries made for the C01 instance. As the population size increases, so does the iteration time, and for larger populations, the compositions with the highest share of elites have the fastest iterations. It is worth noting that the mean time per iteration ranges from 6 to 18 seconds for RW1, compared to 0.5 to 2 seconds for C01. The likely reason for this difference is that RW1 is far larger than C01. This means that the shortest path computations that are made during crossover are more demanding for the larger instance.

A more extensive investigation is needed to optimize the size and composition of the population with respect to performance. Nevertheless, the results from both instances suggest that larger population sizes are needed to reduce the risk of premature convergence and achieve the best possible solution quality. As such, the population size of 300 is chosen to generate the results in the following section. As the share of elite chromosomes seems to be of little importance given a population of this size, the 40% elite composition is chosen for further runs as it has the lowest runtime per iteration.

7.3 Algorithmic performance

7.3.1 Benchmark results

The EA was run on three sets of benchmark instances, namely the GDB, EGL and BMCV instances introduced in chapter 6. Furthermore, the mathematical formulation presented in chapter 4 was implemented in the commercial solver Xpress and applied to the GDB instances. Table 7.2 shows the results from the runs on the GDB instances, both for the EA and Xpress, while tables 7.3 and 7.4 provide the results for the EGL and BMCV instances, respectively.

Table 7.2: Results for the GDB instances

	EA			Xpress			
	Fitness	% gap	Sol. time (s)	Obj. value	Best LB	% gap	Sol. time (s)
gdb1	316	0.0	5.2	316	301.5	5.6	1 000
gdb4	287	0.0	4.1	287	287	0.0	943
gdb7	325	0.0	4.7	325	293.5	9.7	1 000
gdb14	100	0.0	4.4	100	100	0.0	1
gdb15	58	0.0	2.7	58	58	0.0	2
gdb17	91	0.0	4.6	92	87	5.4	1 000
gdb19	55	0.0	0.1	55	55	0.0	0

Table 7.3: Results for the EGL instances

	Fitness	% gap	Sol. time (s)
egl-e1-A	3 608	1.7	196.9
egl-e3-A	6 349	7.6	1 038.3
egl-s1-A	5 199	3.6	335.8
egl-s2-C	17 735	8.0	1 761.7
egl-s3-C	18 610	8.3	4 819.7

Table 7.4: Results for the BMCV instances

	Fitness	% gap	Sol. time (s)
C01	4 385	5.7	1 497.4
C08	4 275	4.5	596.9
C17	3 700	4.1	39.2
D15	4 235	6.1	932.1
E07	4 155	0.0	591.4

7.3.2 Exact method

The results presented in table 7.2 indicate the shortcomings of the exact method. The solver performed well for three of the GDB instances, which were solved to optimality in 2 seconds or less. However, it was unable to find the optimal solution for two of the instances within the maximum runtime. In comparison, the EA solved all seven instances to optimality within a maximum of 5.2 seconds. The CARP, being an NP-hard problem, is difficult to tackle with exact methods, and it is expected that a heuristic method such as the EA should perform better. The low runtimes combined with optimal results indicate that the EA has a potential for solving problems of greater size.

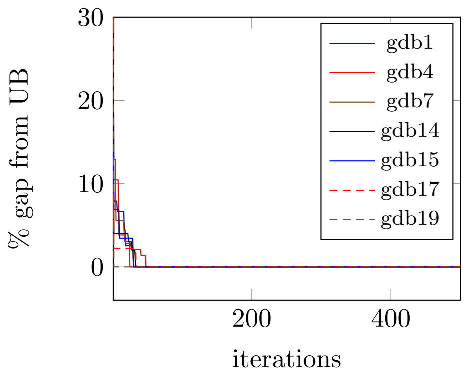
7.3.3 Solution quality

Figure 7.6 illustrates the evolution of the best chromosome in each generation for the runs on the benchmark instances. The first axis shows iterations, while the second axis shows the percentage gap to the best known upper bound for the given problem instance.

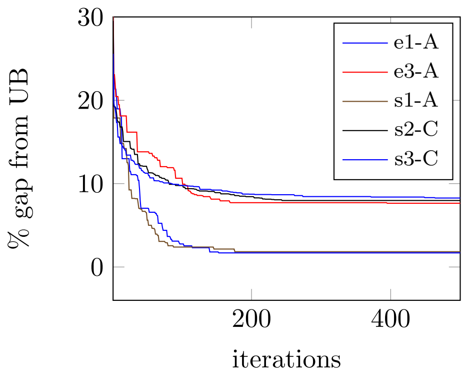
While the EA identified the optimal solution for all GDB instances, the optimal value was found for only one of the ten EGL and BMCV instances. All the GDB instances were solved to optimality in less than 50 iterations, while E07 used 475 iterations to reach optimality. The EGL and BMCV instances have more extensive networks, making them harder to solve. Furthermore, while the GDB instances have tasks on all arcs, a fraction of the arcs in the BMCV and EGL instances are without tasks. This might explain why the EA performs differently on the instance sets.

From figure 7.6b an 7.6c, it seems that the EA converges within about 200-300 iterations for the EGL and many of the BMCV instances. In most cases, the EA converges towards a value above the optimal, indicating that the EA is stuck in local optima.

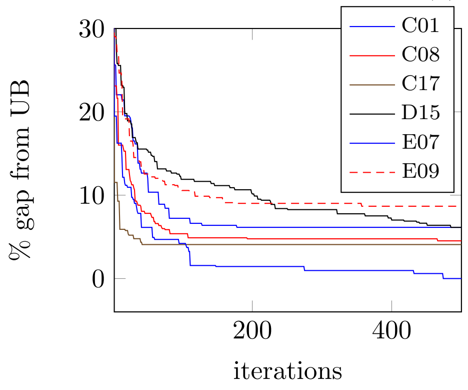
The solutions obtained for three of the instances shown in figure 7.6b an 7.6c stand out as significantly better than the others. These instances, egl-e1-A, egl-s1-A, and E07, are characterized by having the lowest fractions of required arcs of all the benchmark instances. The three instances have 52%, 39% and 45% required arcs, respectively. This observation indicates that the EA might perform better when the fraction of required arcs is low. As the network representing Oslo consists of 23% required arcs, the algorithm's performance on real-world instances may be stronger than for the benchmark instances.



(a) GDB



(b) EGL



(c) BMCV

Figure 7.6: Fitness score of best chromosome in each generation.

7.3.4 Convergence

All GDB instances, which are far smaller than the EGL and BMCV instances, were solved within 50 iterations. From the results, it appears that as the size and complexity of the instances grow, the algorithm needs more iterations before it reaches a point of convergence. This is in line with our expectations, as larger instances, in general, have larger solution spaces, which the EA will need more iterations to explore.

As suggested in Section 7.3.3, the EA seems to converge towards local optima for most of the runs. A possible cause of this premature convergence is that the initial solutions do not adequately cover the solutions space of the problem. While the results indicate that the EA tends to get stuck in local optima, the parameter analysis in section 7.2 shows that larger population sizes tend to yield better solutions. A possible explanation for this observation is that that a larger population size gives a lower probability of getting stuck in local optima, as the initial population is likely to cover more substantial parts of the solution space.

Another potential reason for the premature convergence is that no mutation procedures are used. Implementing a mutation procedure with stochastic elements that increase the neighborhood size for each solution could lessen the probability of converging prematurely, and in turn, increase the quality of the obtained solutions.

A third factor contributing to the EA getting stuck in local optima could be the heuristic search procedure applied to find the insertion point in the crossover phase. This procedure was implemented to speed up the convergence of the algorithm. However, it increases the chance of getting stuck in local optima, since it always selects the apparent best move. A possible solution to this problem could be to add a random element in the selection of the insertion point in the crossover procedure. However, this might lead to a slower rate of convergence. As such, there might be a trade-off between a faster rate of convergence and improved solution quality.

7.3.5 Stochasticity of the EA

The EA has stochastic elements both in the construction and crossover phase. Therefore, running the algorithm multiple times on the same instance should yield different results, and in general lead to an improved best solution.

To examine how the solution quality varies with multiple runs, the EA was run ten times with identical input parameters for the egl-s3-C instance. The test results are plotted in figure 7.7

It is clear from the figure that running the algorithm multiple times yields different solutions. The worst runs seem to converge at an optimality gap of about 10%, whereas the best solution reached a gap of about 7% within 200 iterations, and did not seem to converge within the iteration limit. The reason for this is likely the result of two factors: The EA is a stochastic method, so each run is different. However, the converging runs seem to do so as a result of getting stuck in local optima. These results suggest that to utilize the potential of the current EA, multiple runs should be conducted.

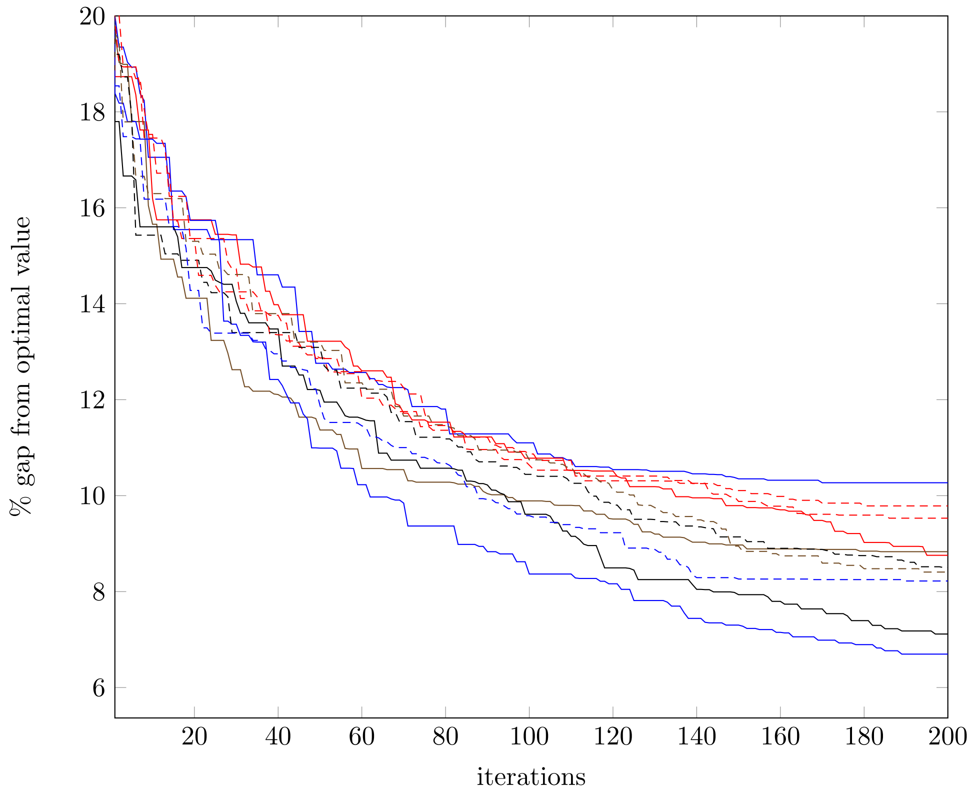


Figure 7.7: Gap for 10 runs of the egl-s3-C instance with the same parameters

7.3.6 The construction heuristic

It is apparent that the initial solutions provided by the construction heuristic were better for the GDB set than for the other sets, with a gap to optimality of between 5% to 40% for all instances. The EGL instances started out worse, with gaps of about 25% to 50%, whereas the initial BMCV solutions had gaps of about 30% to 50%. Generally, it seems that the construction heuristic constructs better solutions for the smaller instances. The construction heuristic seems to perform slightly worse than the classical heuristics of Path-Scanning, Augment-Merge and Ulusoy's, which according to Wøhlk (2008) construct solutions that are generally around 10 % to 40 % above the optimal solutions.

7.3.7 Impact of initial population

Table 7.5 lists the optimality gaps of the best chromosome in the initial population and the best chromosome in the final generation for the ten runs that were conducted on the egl-s3-C instance. The values in the rows of the table are graded so that the lowest values are darkest. As such, a positive correlation between the initial and final fitness score is indicated either by two corresponding dark values or two corresponding light graded values.

Table 7.5: Gap after construction and after 200 iterations for 10 runs of the egl-s3-C instance with the same input parameters.

	1	2	3	4	5	6	7	8	9	10
% gap after construction	23.89	23.67	30.58	28.72	20.89	21.72	23.00	27.19	28.00	29.33
% gap after evolution	10.27	8.76	8.83	7.12	6.70	9.53	8.41	8.50	8.22	9.79

From the results in the Table, it is evident that the population with the fittest chromosome after construction also achieved the best fitness value after 200 iterations. However, the Pearson correlation coefficient for the fitness before and after evolution is 0.074, indicating no strong correlation between the initial and final fitness values. Thus, the fitness of the best chromosome in the initial population does not seem to be critical for the performance of the EA.

The diversity of the initial population, however, could also impact the quality of the resulting solutions, as discussed in Section 7.3.4. Our algorithm uses a heuristic distance measure combined with a randomly permutated list of tasks to create initial solutions. It is likely that the heuristic measure, while producing better solutions, also leads to a less diverse initial population. As such, means that make the initial population cover the solution space more thoroughly should be considered. Using additional different construction heuristics to create the initial population, as well as adding randomly generated chromosomes are examples of such means.

As discussed in Section 7.2, larger populations seem to perform better because they are likely to be more diverse. However, larger populations lead to longer run times for the EA. By further diversifying the initial population, it is possible that the performance of the EA with lower populations sizes will improve. This is beneficial as the total run time would be reduced.

7.3.8 Iteration times

As one can see in figures 7.3 and 7.5, the CPU time for each iteration increases with the size of the instance. For the largest instances, and RW0 in particular, the iteration time is too high to be able to reach close to optimal values within realistic time limits. As such, to enable the EA to be of practical use for very large instances, the algorithm should be streamlined to reduce iteration times.

Three bottlenecks have been identified in the algorithm. First and foremost, the crossover operation uses significant CPU time. The considerable time usage is likely in part caused by iteration over long chromosomes. However, the most time-consuming operation is probably the shortest path calculations. The algorithm runs no faster than the slowest worker process, and with a large population size, the probability of at least one shortest path computation being computationally demanding increases. The current algorithm calculates four shortest paths for each insertion, namely to and from each of the end nodes of the path. By disregarding the potential savings of reversing the path, the number of shortest path calculations per crossover can thus be reduced to 2.

The second bottleneck is related to the MPI interface. As instances grow, so do the chromosomes, and long chromosomes require more memory compared to short ones. As the MPI interface passes a file containing the chromosome between processes, the speed of this operation depends on the file size, and hence the length of the chromosome to send.

The third bottleneck is the operation of converting chromosomes to string representations and string representations back to chromosomes in each end of an MPI communication procedure. Since our implementation of the chromosomes depends on pointers to other objects, the chromosomes cannot be represented by the standard data types accepted by the MPI interface. However, by representing the chromosomes in terms of such data types, the conversion process can be omitted. This will save time in both ends of a communication process. For a crossover, two communication processes are conducted. As such, four conversions per crossover can be omitted by changing the chromosome implementation.

7.4 Results for the real-world instances

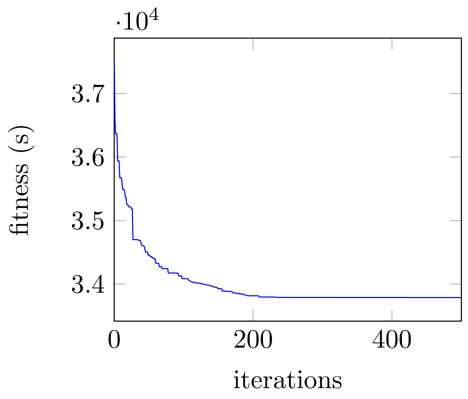
In this Section, the results from the runs on the real-world instances are presented and discussed in light of the findings in Section 7.3.

The real-world instances were run with the parameters given in 6.2.2. The RW1, RW2 and RW3 instances were all run for 500 iterations. However, due to the significant iteration time for the run on RW0, this instance was only run for 20 iterations.

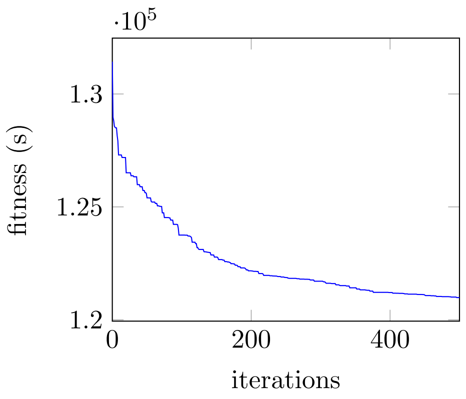
The results are presented in Table 7.6, and the fitness of the best chromosomes in each generation are plotted in Figure 7.8.

Table 7.6: Results for the real-world instances

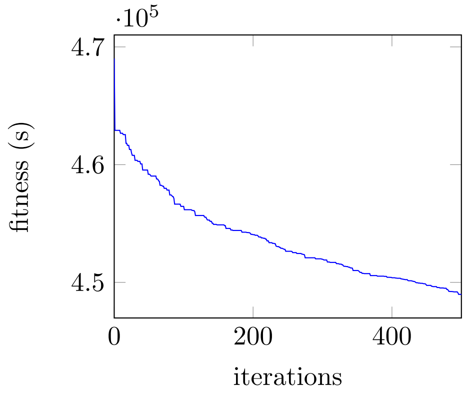
	Fitness	Sol. time (s)
RW1	33 789.5	7 052.8
RW2	120 986.3	71 831.3
RW3	448 980.2	450 276.8
RW0	2 219 689.25	395 194.3



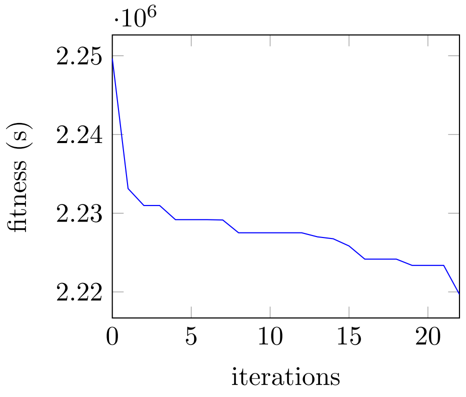
(a) RW1



(b) RW2



(c) RW3



(d) RW0

Figure 7.8: Evolution of the best chromosome in each generation for the RW instances.

7.4.1 Convergence

The plots in 7.8 indicate that the fitness of the best chromosome in a population converges slower as the size of the instance increases. This is in line with the observations made in Section 7.3.4.

Figure 7.8a shows that the EA reaches a point of convergence after about 200 iterations for RW1, an instance far larger than the benchmark instances. The EA tends to get stuck in local optima for these, suggesting that this could also be the case for RW1. As such, performing multiple runs of the RW1 instance could improve the best-found solution.

From figure 7.8b it appears that the EA has not converged within the 500 iterations it was run for on the RW2 instance. The RW2 instance is far larger than the benchmark instances, so this observation is in line with our expectations.

Furthermore, the observation implies that there is room for improving the solution for RW2 by simply adding more iterations. The most significant improvements for the RW2 run, however, occur before the 200 iteration mark.

The results for the RW3 instance are plotted in figure 7.8c. From the figure, it appears that the EA does not converge within the 500 iterations. The curve is far steeper at 500 iterations for RW3 than for RW2, which could indicate that the EA is further away from reaching a point of convergence.

The RW0 instance is by far the largest instance and was only run for 20 iterations. As even the smallest benchmark instances needed about 50 iterations to reach convergence, we can conclude that the results obtained from the run on the RW0 instance are far from optimal.

Furthermore, the results presented in Figure 7.8 shows that while the fitness for the runs on the RW1 and RW2 instances seem to converge, the fitness graph for the RW3 instance is in significant decline at iteration 500. As such, the fitness of the RW3 run is still not approaching an optimum. These observations suggest that the RW0 instance needs even more than 500 iterations to approach optima.

7.4.2 Iteration times

In terms of iteration times, the results obtained from the runs on the real-world instances are in line with the findings presented in Section 7.3. The results presented in Table 7.6 shows that iteration times increase with the size of the instance. While the RW1 instance used 16 seconds per iteration on average, the RW0 instance used five hours. Thus, one can conclude that the EA has issues regarding iteration times for instances of this size and that the algorithm is insufficient for optimizing routing problems at the size of Oslo.

7.4.3 Applicability of the results

To assess the applicability of the results, the solution obtained from the run on the RW3 instance is used. The size of this instance is significant, and the resulting routing plan was generated by running the EA for 500 iterations.

For the RW3 instance, the EA obtained a solution using 20 shifts. As this instance contains about 9 000 pickup locations, each shift services 450 pickup locations on average. In comparison, REN estimates that each of their shifts service about 400 pickup locations. As the best-obtained solution for the RW3 instance does not seem to be close to optimal but still appears to be better that REN's current routes, this suggests that the input parameters presented in Section 6.2.2 are imprecise. As both the pickup times and vehicle speeds are simple estimations, the results indicate that these parameters should be more carefully modeled to achieve solutions of practical applicability. However, the results also show that the algorithm yields solutions of the same magnitude as today's routing practice suggests.

Figure 7.9 shows the domain of the RW3 instance, as well as the streets traversed in each shift. Each shift is marked with different colors. The red dot in the figure shows the depot. Figures 7.10 and 7.11 shows two shifts from the obtained solution, with each trip within the given shift indicated by a different color. Appendix A provides a suggested routing schedule for the solution obtained for RW3, assigning the shifts as suggested in Section 5.9 given two vehicles driving two shifts each day. The appendix also includes illustrations of all shifts.

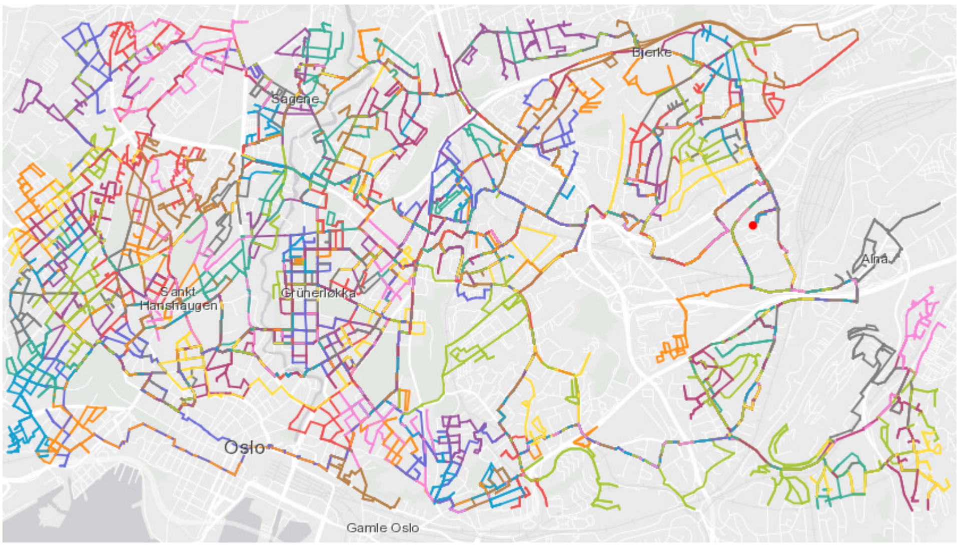


Figure 7.9: The shifts in the obtained solution for RW3.

One of the benefits of using giant tours as chromosomes is that close-to-optimal solutions yield trips within the same area for each shift. This gives predictability and flexibility in the planning and operations. If a vehicle has excess capacity and the trips within a shift are in close proximity, the vehicle may relieve pressure from other trips within the same shift. The trips of Shift 9 in Figure 7.10 exhibits this property.

However, as Figure 7.11 indicates, this is not the case for all shifts in the solution. Visual inspection of the trips indicates that tasks from three to four different areas are serviced within the same shift. This observation indicates that the obtained solution is not yet approaching an optimum.

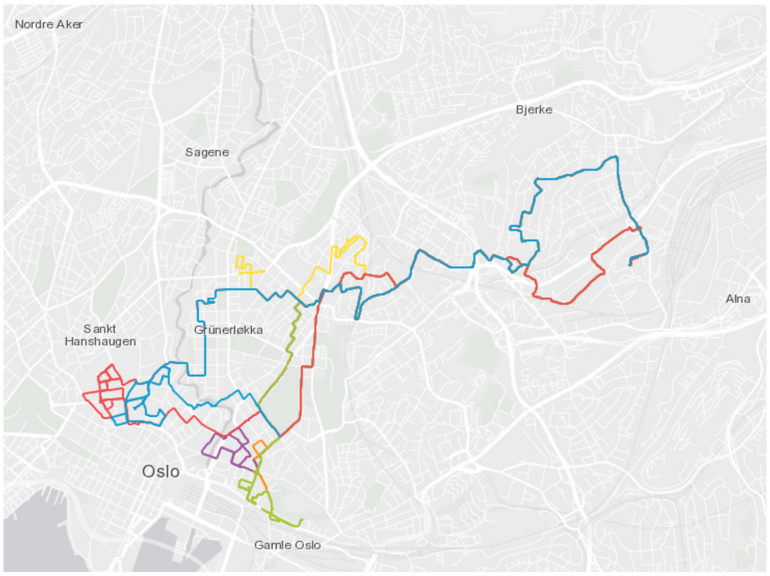


Figure 7.10: Shift 9 of the obtained solution.

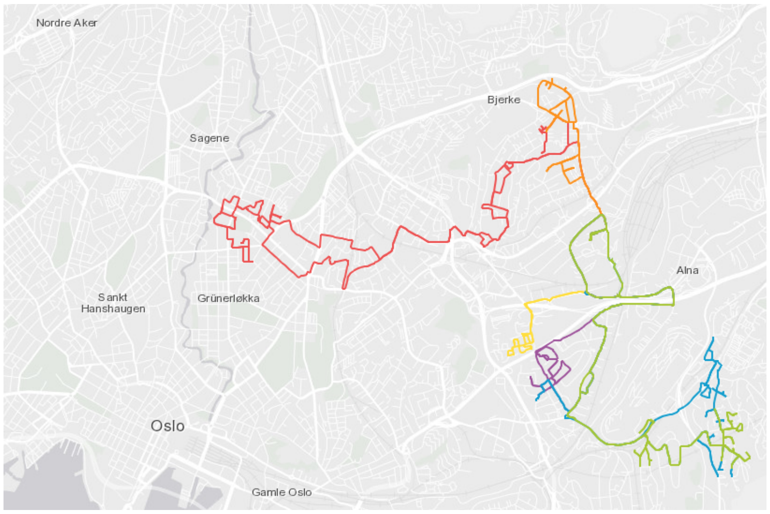


Figure 7.11: Shift 17 of the obtained solution.

Chapter 8

Concluding Remarks

This thesis studies the waste collection problem faced by the public waste management agency in Oslo. A mathematical formulation, describing the problem as a Capacitated Arc Routing Problem with intermediate facilities and time restrictions, is introduced. It is an extension of the original CARP, in which vehicles can unload or replenish multiple times during a tour, and the tours have a maximum duration.

An evolutionary algorithm, inspired by concepts prevalent in the literature, is proposed. The EA is based on a hybrid crossover operation which incorporates a heuristic search procedure. The search procedure is used to decide where a random fraction of one parent chromosome should be inserted into another parent chromosome. The purpose of the search procedure is to speed up the convergence of the algorithm. Initial chromosomes for the EA are built using a stochastic construction heuristic utilizing a greedy heuristic measure. The EA is tested on well-known benchmark instances from the literature and applied to instances based on real-world infrastructure.

The results from the benchmarking show that the proposed EA performs well for small instances, while it struggles with the solution quality as the complexity of the instances increases. However, the instances for which the algorithm performed best were those with a topography most resembling the real-world instances.

The problem regarding solution quality seems to be that the EA gets stuck in local optima. As a result, the algorithm converges before reaching optimal values. Three potential causes of this are identified.

Firstly, it is possible that the initial population does not adequately cover the solution space. The EA obtained better fitness scores with larger, more diverse populations, which indicates that this is the case. This problem can be solved by applying additional construction heuristics and adding randomly generated chromosomes when creating the initial population.

Secondly, the crossover operation applies a search procedure with the purpose of speeding up the convergence of the algorithm, which may contribute to the premature convergence. By adding a random element when choosing the insertion point could help avoid this effect.

Thirdly, the EA was run without mutation, as the three proposed mutation operators were unable to improve the fitness of the chromosomes and increased the run time of the algorithm. This lack of a mutation phase leads to an EA that is less explorative. Experiments with mutation operators and more advanced search procedures could improve the performance of the EA by counteracting premature convergence.

The proposed algorithm was able to produce feasible solutions to real-world problems of significant size and to generate a routing plan for the collection of waste from the entire city of Oslo. However, due to long run times, this plan was generated using a mere 20 iterations. The results from the other instances suggest that as the problem size grows, more iterations are needed to reach a point of convergence for the fitness score. To reach close-to-optimal solutions for an instance of the size of Oslo, one would need far more than 20 iterations.

The purpose of the EA is to solve such large-scale waste collection problems. Even though the EA has problems regarding exploration of the solution space, the most pressing issue is the long run time of the algorithm, as the algorithm is far from reaching convergence for the largest instances due to run time limitations.

Chapter 9

Further Research

During the process of writing this thesis, several areas arose as potential topics further research. First and foremost, the primary focus in the literature on EAs for routing purposes is on solving small, theoretical benchmark instances rather than practical applications. Thus, we suggest that practical routing applications should be emphasized going forward, as such applications may require other characteristics of an EA than the benchmark instances prevalent in the literature do. As an extension of this, we suggest that more complex benchmark instances containing the coordinates of nodes or heuristic distances between nodes should be developed, to exploit efficient heuristic methods in the algorithms. Such benchmark sets would enable for a better assessment of the performance of algorithms targeted at real-world applications.

Additionally, several aspects should be considered to improve the proposed algorithm. A functional mutation procedure would benefit the algorithm by countering the issue of premature convergence. However, as the algorithm converges slowly for larger instances, combining a mutation operator with a local search procedure could improve the performance of the algorithm.

Lastly, the EA could be altered to better represent the real-world scenario of waste collection. Such improvements include accounting for uncertainty. Several parameters, such as waste volumes and traversal times, are stochastic, and an improved algorithm should reflect this trait. Furthermore, by extending the algorithm to account for multiple waste management facilities, the EA could be adapted to even more realistic scenarios.

Appendix A

Solution of the RW3 instance

A.1 Suggested schedule

Vehicle	Monday, 1. shift	Monday, 2. shift	Tuesday, 1. shift	Tuesday, 2. shift	Wednesday, 1. shift	Wednesday, 2. shift	Thursday, 1. shift	Thursday, 2. shift	Friday, 1. shift	Friday, 2. shift
1	Shift 1	Shift 3	Shift 5	Shift 7	Shift 9	Shift 11	Shift 13	Shift 15	Shift 17	Shift 19
2	Shift 2	Shift 4	Shift 6	Shift 8	Shift 10	Shift 12	Shift 14	Shift 16	Shift 18	Shift 20

Table A.1: Suggested schedule for the RW3 solution

A.2 Illustrations of the shifts

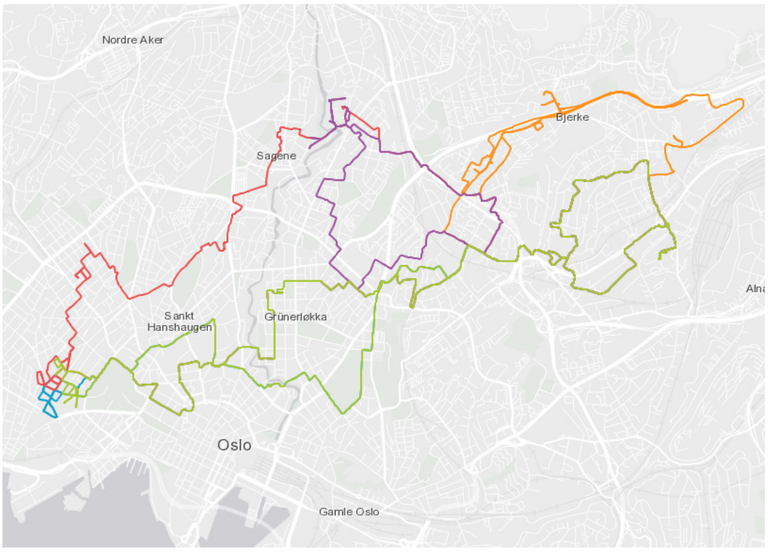


Figure A.1: Shift 1

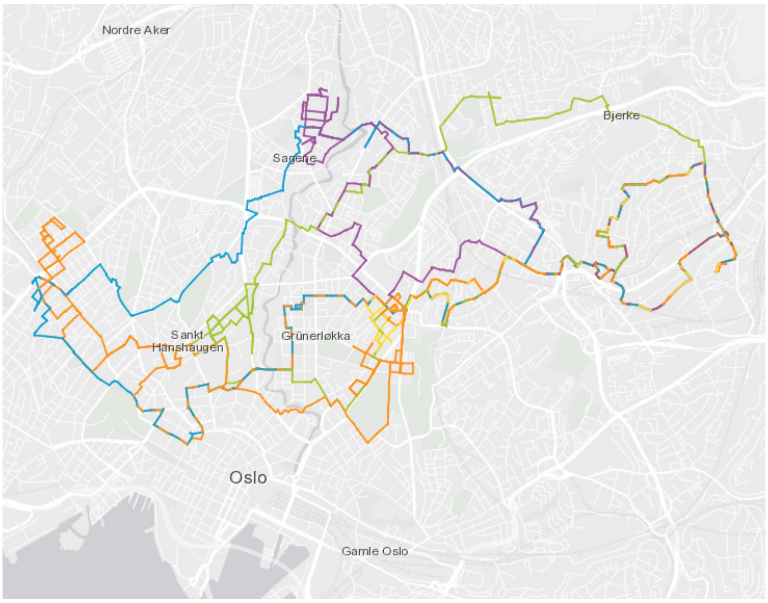


Figure A.2: Shift 2

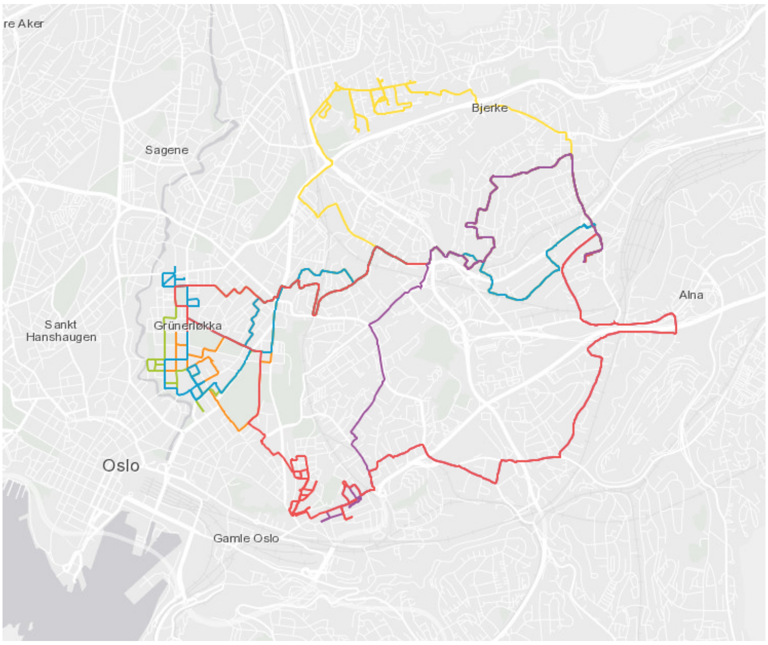


Figure A.3: Shift 3

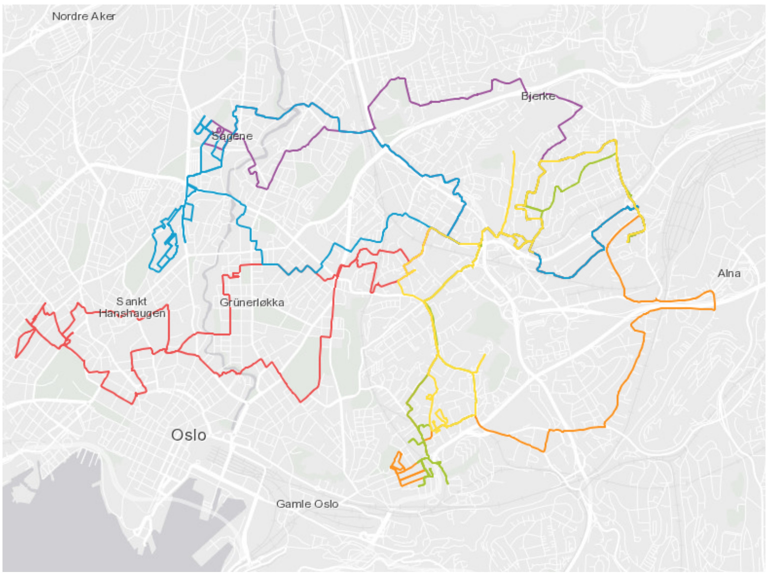


Figure A.4: Shift 4

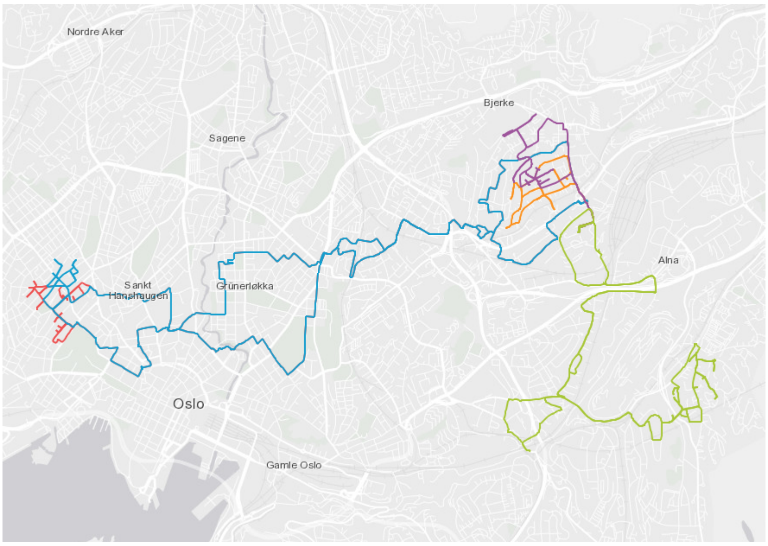


Figure A.5: Shift 5

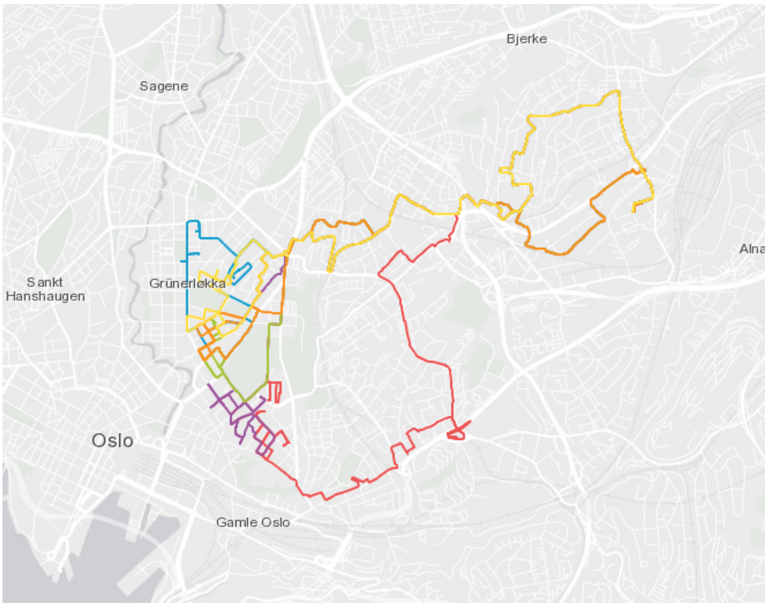


Figure A.6: Shift 6

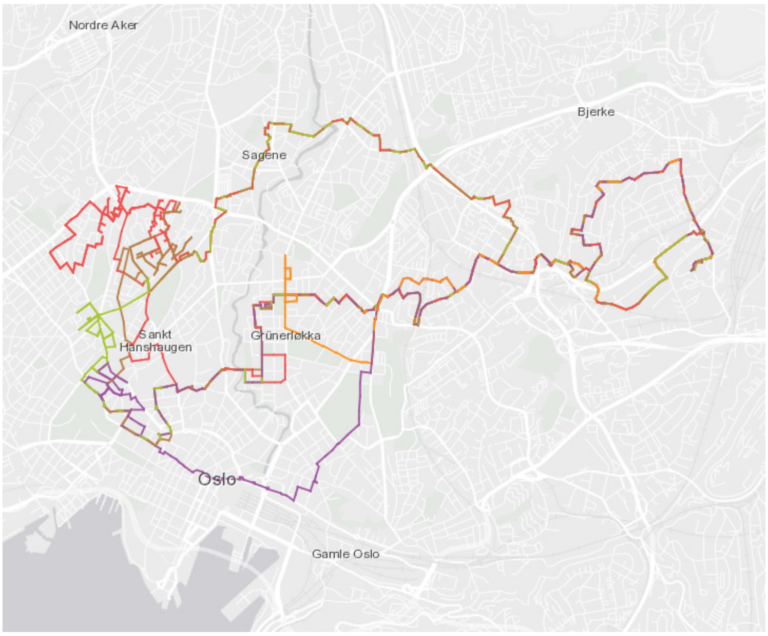


Figure A.7: Shift 7

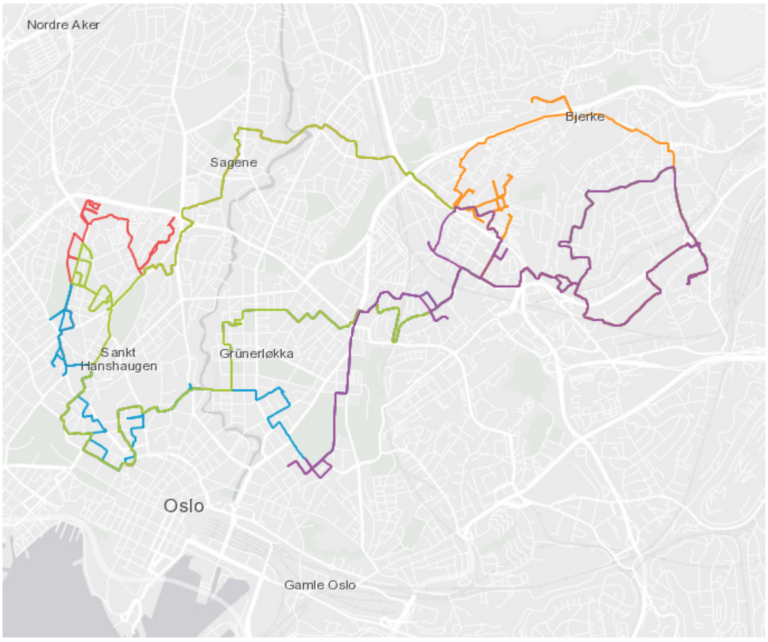


Figure A.8: Shift 8

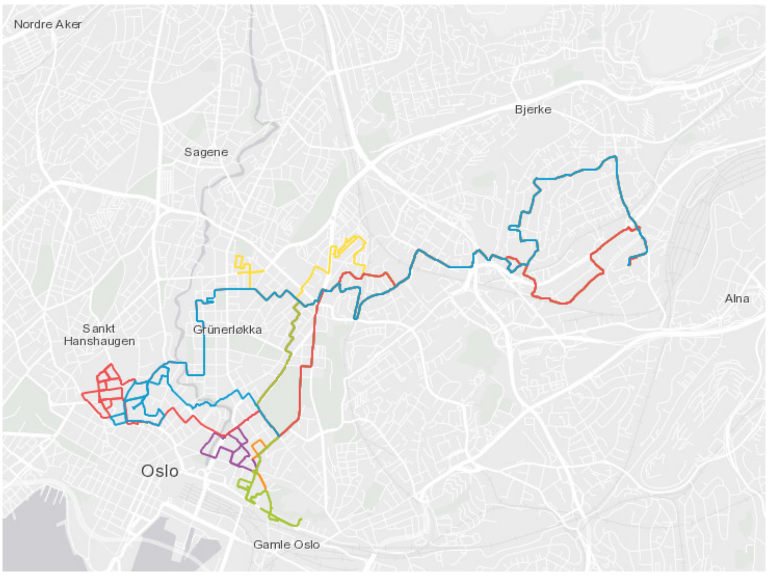


Figure A.9: Shift 9

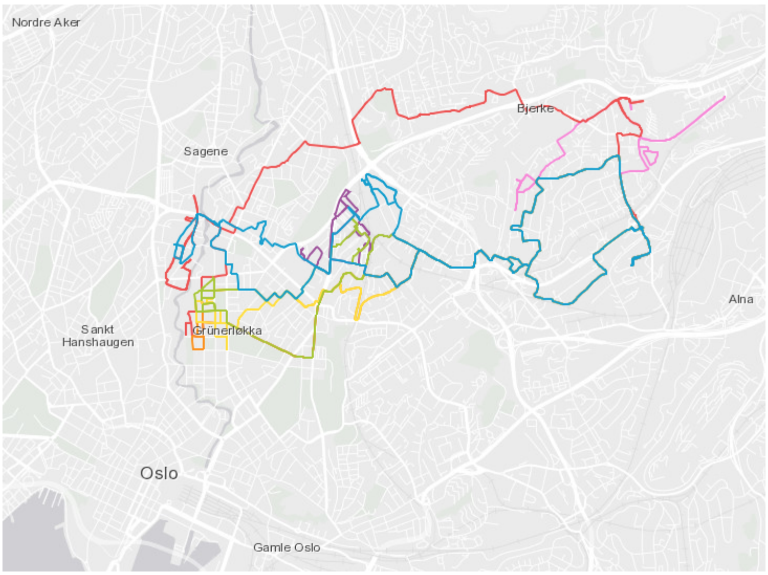


Figure A.10: Shift 10

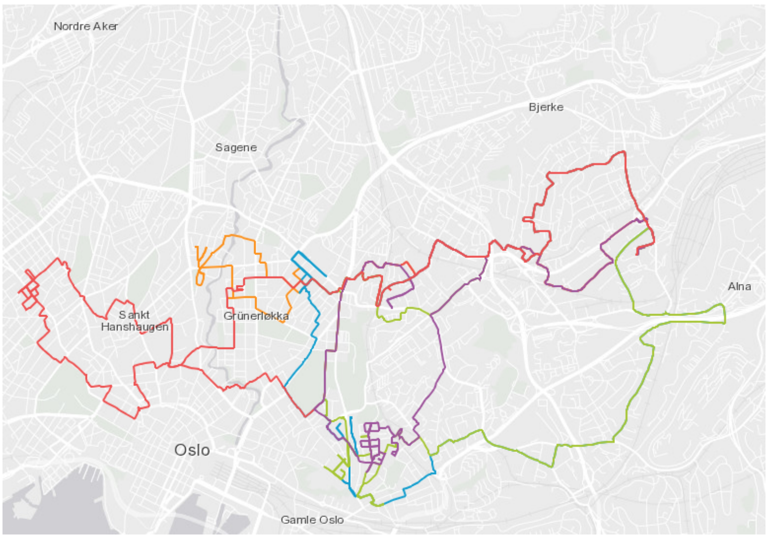


Figure A.11: Shift 11

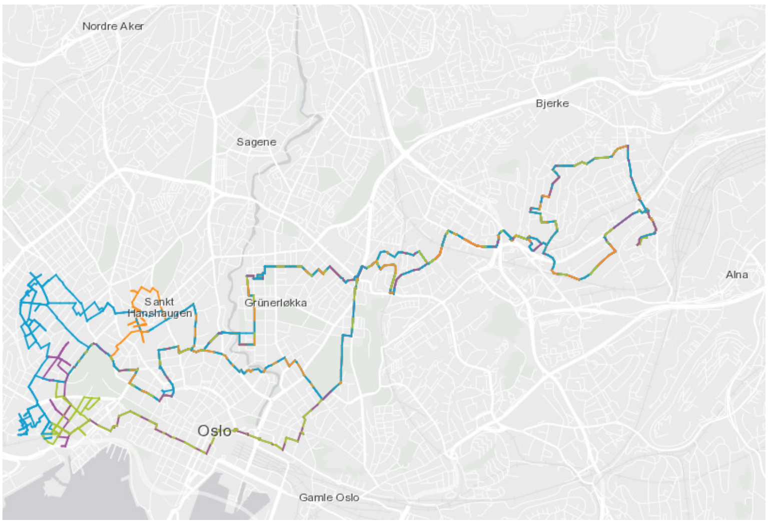


Figure A.12: Shift 12

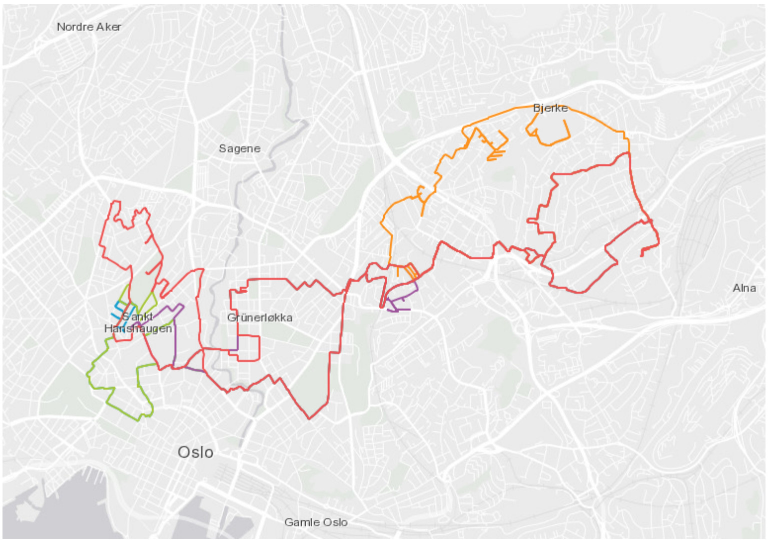


Figure A.13: Shift 13

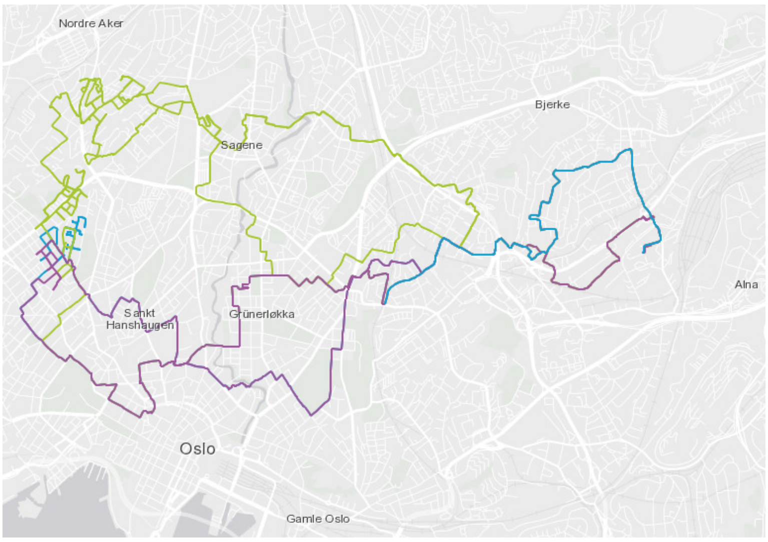


Figure A.14: Shift 14

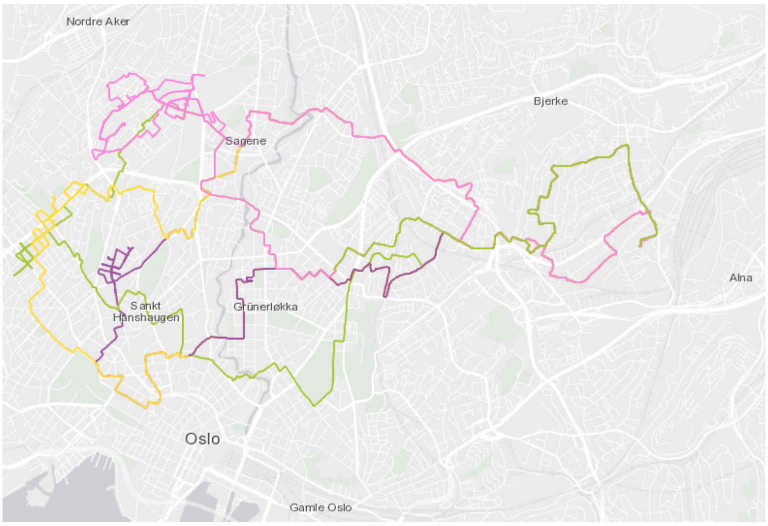


Figure A.15: Shift 15

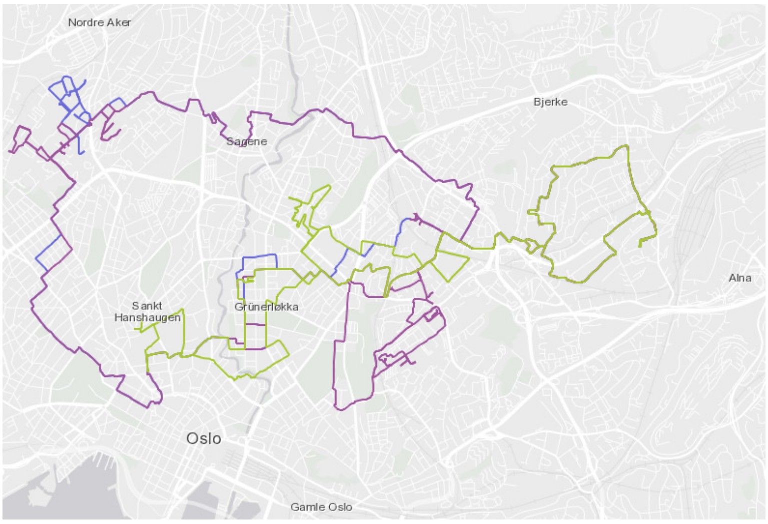


Figure A.16: Shift 16

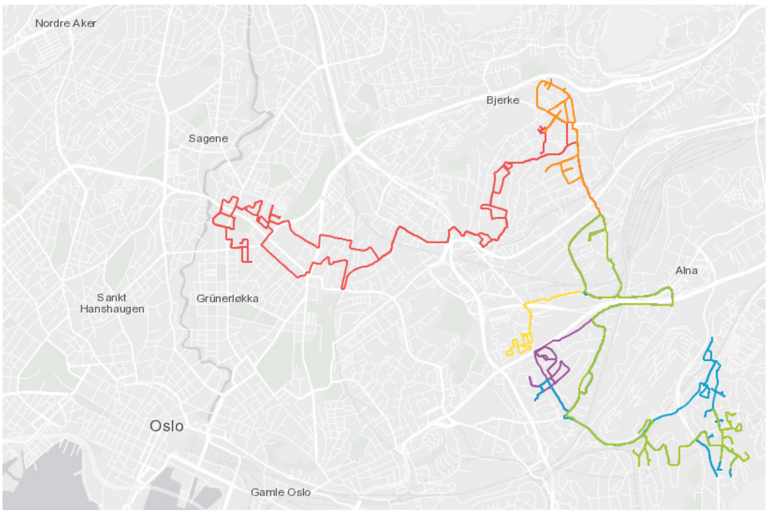


Figure A.17: Shift 17

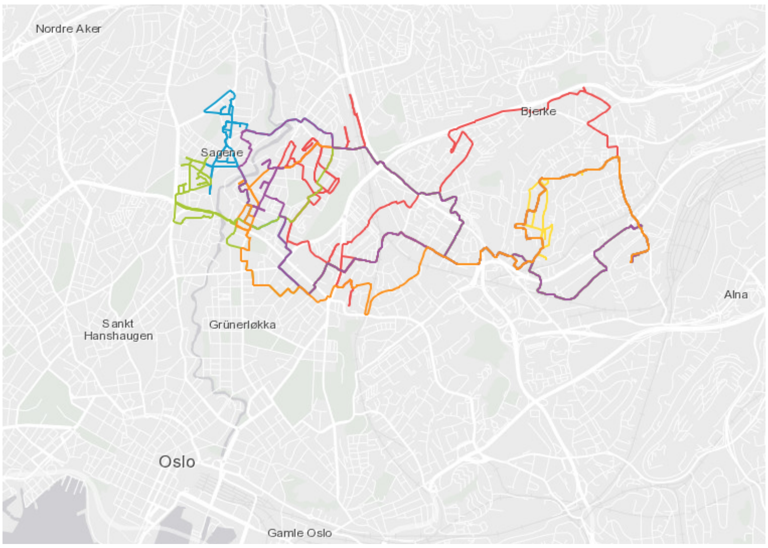


Figure A.18: Shift 18

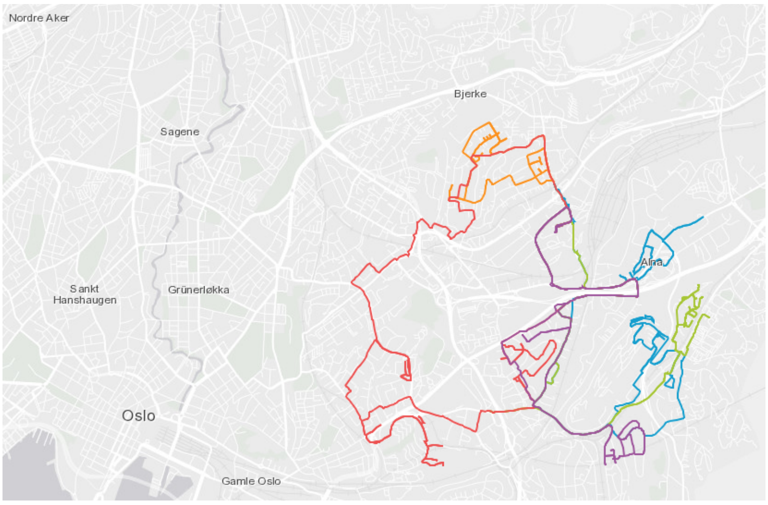


Figure A.19: Shift 19

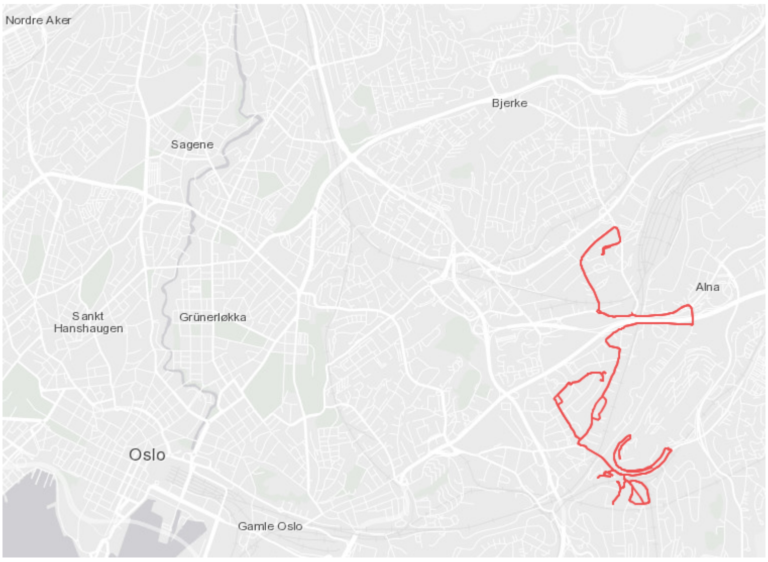


Figure A.20: Shift 20

Bibliography

Aftenposten (2017). Kommunefakta oslo.

Amberg, A., Domschke, W., and Voß, S. (2000). Multiple center capacitated arc routing problems: A tabu search algorithm using capacitated trees. *European Journal of Operational Research*, 124(2):360–376.

Archetti, C., Speranza, M. G., and Hertz, A. (2006). A tabu search algorithm for the split delivery vehicle routing problem. *Transportation science*, 40(1):64–73.

Baker, B. M. and Ayechew, M. (2003). A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800.

Baldacci, R. and Maniezzo, V. (2006). Exact methods based on node-routing formulations for undirected arc-routing problems. *Networks*, 47(1):52–60.

Belenguer, J. M. and Benavent, E. (2003). A cutting plane algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 30(5):705–728.

Beliën, J., De Boeck, L., and Van Ackere, J. (2012). Municipal solid waste collection and management problems: a literature review. *Transportation Science*, 48(1):78–102.

Benavent, E., Campos, V., Corberán, A., and Mota, E. (1992). The capacitated arc routing problem: lower bounds. *Networks*, 22(7):669–690.

Beullens, P., Muyldermans, L., Cattrysse, D., and Van Oudheusden, D. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147(3):629–643.

Blanton Jr, J. L. and Wainwright, R. L. (1993). Multiple vehicle routing with time and capacity constraints using genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 452–459. Morgan Kaufmann Publishers Inc.

Brandão, J. and Eglese, R. (2008). A deterministic tabu search algorithm for the capacitated arc routing problem. *Computers & Operations Research*, 35(4):1112–1126.

Chen, Y., Hao, J.-K., and Glover, F. (2016). A hybrid metaheuristic approach for the capacitated arc routing problem. *European Journal of Operational Research*, 253(1):25–39.

Christofides, N. (1973). The optimum traversal of a graph. *Omega*, 1(6):719–732.

Corberán, A. and Prins, C. (2010). Recent results on arc routing problems: An annotated bibliography. *Networks*, 56(1):50–69.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271.

Dorigo, M., Birattari, M., and Stutzle, T. (2006). Ant colony optimization. *IEEE computational intelligence magazine*, 1(4):28–39.

Dror, M. (2012). *Arc routing: theory, solutions and applications.* Springer Science & Business Media.

Eglese, R. W. (1994). Routeing winter gritting vehicles. *Discrete applied mathematics*, 48(3):231–244.

Fleury, G., Lacomme, P., and Prins, C. (2004). Evolutionary algorithms for stochastic arc routing problems. In *Workshops on Applications of Evolutionary Computation*, pages 501–512. Springer.

Fleury, G., Lacomme, P., Prins, C., and Ramdane-Cherif, W. (2005). Improving robustness of solutions to arc routing problems. *Journal of the operational research society*, 56(5):526–538.

Fung, R. Y., Liu, R., and Jiang, Z. (2013). A memetic algorithm for the open capacitated arc routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 50:53–67.

Gendreau, M. and Potvin, J.-Y. (2005). Metaheuristics in combinatorial optimization. *Annals of Operations Research*, 140(1):189–213.

Ghiani, G., Improta, G., and Laporte, G. (2001). The capacitated arc routing problem with intermediate facilities. *Networks*, 37(3):134–143.

Ghiani, G., Laganà, D., Laporte, G., and Mari, F. (2010). Ant colony optimization for the arc routing problem with intermediate facilities under capacity and length restrictions. *Journal of Heuristics*, 16(2):211–233. Copyright - Springer Science+Business Media, LLC 2010; Document feature - ; Tables; Equations; Last updated - 2012-07-24.

Golden, B. L., DeArmon, J. S., and Baker, E. K. (1983). Computational experiments with algorithms for a class of routing problems. *Computers & Operations Research*, 10(1):47–59.

Golden, B. L. and Wong, R. T. (1981). Capacitated arc routing problems. *Networks*, 11(3):305–315.

Hertz, A., Laporte, G., and Mittaz, M. (2000). A tabu search heuristic for the capacitated arc routing problem. *Operations research*, 48(1):129–135.

Hertz, A. and Mittaz, M. (2001). A variable neighborhood descent algorithm for the undirected capacitated arc routing problem. *Transportation science*, 35(4):425–434.

Hirabayashi, R., Saruwatari, Y., and Nishida, N. (1992). Tour construction algorithm for the capacitated arc routing-problems. *Asia-Pacific Journal of Operational Research*, 9(2):155–175.

Kiilerich, L. and Wøhlk, S. (2018). New large-scale data instances for carp and new variations of carp. *INFOR: Information Systems and Operational Research*, 56(1):1–32.

Kiuchi, M., Shinano, Y., Hirabayashi, R., and Saruwatari, Y. (1995). An exact algorithm for the capacitated arc routing problem using parallel branch and bound method. In *Spring national conference of the operational research society of Japan*, pages 28–29.

Labadi, N., Prins, C., and Reghioui, M. (2008). An evolutionary algorithm with distance measure for the split delivery capacitated arc routing problem. In *Recent advances in evolutionary computation for combinatorial optimization*, pages 275–294. Springer.

Lacomme, P., Prins, C., and Ramdane-Chérif, W. (2001). A genetic algorithm for the capacitated arc routing problem and its extensions. *Applications of evolutionary computing*, pages 473–483.

Lacomme, P., Prins, C., and Ramdane-Cherif, W. (2004a). Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131(1):159–185.

Lacomme, P., Prins, C., and Sevaux, M. (2003). Multiobjective capacitated arc routing problem. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 550–564. Springer.

Lacomme, P., Prins, C., and Sevaux, M. (2006). A genetic algorithm for a bi-objective capacitated arc routing problem. *Computers & Operations Research*, 33(12):3473–3493.

Lacomme, P., Prins, C., and Tanguy, A. (2004b). First competitive ant colony scheme for the carp. In *International Workshop on Ant Colony Optimization and Swarm Intelligence*, pages 426–427. Springer.

Laporte, G., Musmanno, R., and Vocaturo, F. (2010). An adaptive large neighbourhood search heuristic for the capacitated arc-routing problem with stochastic demands. *Transportation Science*, 44(1):125–135.

Letchford, A. N. and Oukil, A. (2009). Exploiting sparsity in pricing routines for the capacitated arc routing problem. *Computers & Operations Research*, 36(7):2320–2327.

Li, L. Y. and Eglese, R. W. (1996). An interactive algorithm for vehicle routeing for winter—gritting. *Journal of the Operational Research Society*, 47(2):217–228.

Li, L. Y. O. (1992). *Vehicle routeing for winter gritting.* PhD thesis, University of Lancaster.

Liu, M., Singh, H. K., and Ray, T. (2014). Application specific instance generator and a memetic algorithm for capacitated arc routing problems. *Transportation Research Part C: Emerging Technologies*, 43:249–266.

Longo, H., De Aragao, M. P., and Uchoa, E. (2006). Solving capacitated arc routing problems using a transformation to the cvrp. *Computers & Operations Research*, 33(6):1823–1837.

Mourão, M. C. and Pinto, L. S. (2017). An updated annotated bibliography on arc routing problems. *Networks*, 70(3):144–194.

Mullaseril, P. A. (1997). Capacitated rural postman problem with time windows and split delivery.

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

Oliver, I., Smith, D., and Holland, J. R. (1987). Study of permutation crossover operators on the traveling salesman problem. In *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms: July 28-31, 1987 at the Massachusetts Institute of Technology, Cambridge, MA*. Hillsdale, NJ: L. Erlhaum Associates, 1987.

Pearn, W.-L., Assad, A., and Golden, B. L. (1987). Transforming arc routing into node routing problems. *Computers & operations research*, 14(4):285–288.

Polacek, M., Doerner, K. F., Hartl, R. F., and Maniezzo, V. (2008). A variable neighborhood search for the capacitated arc routing problem with intermediate facilities. *Journal of Heuristics*, 14(5):405–423.

Potvin, J.-Y. and Bengio, S. (1996). The vehicle routing problem with time windows part ii: genetic search. *INFORMS journal on Computing*, 8(2):165–172.

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

Prins, C. and Bouchenoua, S. (2005). A memetic algorithm solving the vrp, the carp and general routing problems with nodes, edges and arcs. In *Recent advances in memetic algorithms*, pages 65–85. Springer.

Prins, C., Labadi, N., and Reghioui, M. (2009). Tour splitting algorithms for vehicle routing problems. *International Journal of Production Research*, 47(2):507–535.

Reuters (2018). Key facts about norway.

Santos, L., Coutinho-Rodrigues, J., and Current, J. R. (2010). An improved ant colony optimization based algorithm for the capacitated arc routing problem. *Transportation Research Part B: Methodological*, 44(2):246–266.

SSB (2018). Kommunefakta oslo.

Tang, K., Mei, Y., and Yao, X. (2009). Memetic algorithm with extended neighborhood search for capacitated arc routing problems. *IEEE Transactions on Evolutionary Computation*, 13(5):1151–1166.

Tavares, G., Zsigraiova, Z., Semiao, V., and Carvalho, M. d. G. (2009). Optimisation of msw collection routes for minimum fuel consumption using 3d gis modelling. *Waste Management*, 29(3):1176–1185.

Teixeira, J., Antunes, A. P., and de Sousa, J. P. (2004). Recyclable waste collection planning—-a case study. *European Journal of Operational Research*, 158(3):543–554.

the World Bank Group (2012). What a waste : A global review of solid waste management.

Ulusoy, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22(3):329–337.

Whitley, D. (1994). A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85.

Willemse, E. J. and Joubert, J. W. (2016). Constructive heuristics for the mixed capacity arc routing problem under time restrictions with intermediate facilities. *Computers & Operations Research*, 68:30–62.

Wøhlk, S. (2006). *Contributions to arc routing*. Citeseer.

Wøhlk, S. (2008). A decade of capacitated arc routing. *The vehicle routing problem: latest advances and new challenges*, pages 29–48.

Worldometers (2018).

Xing, L., Rohlfshagen, P., Chen, Y., and Yao, X. (2010). An evolutionary approach to the multidepot capacitated arc routing problem. *IEEE Transactions on Evolutionary Computation*, 14(3):356–374.

Zhang, Y., Mei, Y., Tang, K., and Jiang, K. (2017). Memetic algorithm with route decomposing for periodic capacitated arc routing problem. *Applied Soft Computing*, 52:1130–1142.