# Forside

**Eksamensinformation**
B580004102 - B580004102: Bachelorprojekt, Magnus Jørck-Thomsen

**Besvarelsen afleveres af**
Magnus Jørck-Thomsen
magnj22@student.sdu.dk

**Eksamensadministratorer**
Mona Maoued
mao@sam.sdu.dk
+4565503373

**Bedømmere**
Marco Chiarandini
Eksaminator
marco@imada.sdu.dk
+4565504031

Mette Gamst
Censor
mettegamst.dk@gmail.com

**Besvarelsesinformationer**

**Titel:** Optimering af Indsamling af Farligt Affald
**Titel, engelsk:** Optimization of Hazardous Waste Collection
**Må besvarelsen bruges til undervisning:** Nej

University of Southern Denmark, Odense
Department of Mathematics and
Computer Science

Date: 19.05.2025
Number of keystrokes: 67.366

# Optimization of Hazardous Waste Collection
# Bachelor Project

Author:
Magnus Jørck-Thomsen
14032002, Mathematics-Economics
6. semester

Supervisor:
Marco Chiarandini
Associate Professor
Department of Mathematics
and Computer Science

## Tro og loveerklæring

Det erklæres herved på tro og love, at undertegnede egenhændigt og selvstændigt har udformet denne rapport. Alle citater i teksten er markeret som sådanne, og rapporten eller dele af den har ikke tidligere været fremlagt i anden bedømmelsessammenhæng.

Magnus Jørck-Thomsen

## Abstract

Roadside collection of hazardous and textile waste is a time-consuming and recurring task, that must be completed as fast as possible, to minimize the risk of removal or opening by unauthorized persons. The collections are currently planned and carried out manually, and with local knowledge, by the utility company Nyborg Forsyning & Service A/S in Eastern Funen.

In this project it is attempted to accelerate the collections by modelling the waste collection problem as a multi-compartment capacitated vehicle routing problem. The project is focused on delivering a solution to the utility company, that can produce efficient collection routes, by using exsisting open-source software and solvers, to reduce the time spent on the collections.

It is shown that the solver PyVRP produces better solutions than other comparable open-source and commercial solvers across four synthetically generated datasets based on real-life data. A rough estimate is given showing that the company can save a half to a full day of time on the collection of the largest dataset, if the proposed solution is used. Future research should investigate the possibilities of implementing automatic timetabling for the collections, and explore whether the time it takes to complete the collections can be estimated more accurately.

## Resumé

Husstandsindsamling af farligt affald og tekstiler er en tidskrævende og gentagende opgave, der skal gennemføres hurtigst muligt for at minimere risikoen for, at det fjernes eller tilgås af uvedkommende. Indsamlingerne planlægges og udføres på nuværende tidspunkt manuelt, og på baggrund af lokalkendskab, af forsyningsvirksomheden Nyborg Forsyning & Service A/S på Østfyn.

I dette projekt forsøges det at accelerere indsamlingerne ved at modellere affaldsindsamlingsproblemet som et multi-compartment capacitated vehicle routing problem. I projektet fokuseres der på at levere en løsning, der kan producere effektive indsamlingsruter til virksomheden, ved hjælp af eksisterende open-source software og problemløsere og herved afkorte indsamlingstiden.

Det vises at løseren PyVRP leverer bedre løsninger end andre sammenlignelige open-source og kommercielle løsere på fire kunstigt fremstillede dataset, der er baserede på data fra det virkelige liv. Der gives et overslag på, at virksomheden kan spare mellem en halv og en hel dag på det største datasæt, hvis den foreslåede løsning anvendes. Fremtidig forskning bør undersøge mulighederne for at implementere automatisk planlægning af indsamlingerne, samt at undersøge om tiden, det tager at gennemføre indsamlingerne, kan estimeres mere præcist.

# Contents

**1 Introduction 1**

1.1 Collection of textile and hazardous waste in Eastern Funen . . . . . . . . . . . . 1

1.2 Real-life data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2

1.3 Synthetic data . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3

**2 Literature Review 7**

2.1 Existing vehicle routing problem models in waste collection . . . . . . . . . . . 7

2.2 Solving vehicle routing problems . . . . . . . . . . . . . . . . . . . . . . . . . 8

**3 Solution Methods 9**

3.1 A Mixed Integer Linear Programming Model . . . . . . . . . . . . . . . . . . . 10

3.1.1 Sets . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

3.1.2 Variables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

3.1.3 Decision variables . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

3.1.4 Model . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 11

3.1.5 Implementation with Gurobi . . . . . . . . . . . . . . . . . . . . . . 11

3.2 Hybrid Genetic Search . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

3.2.1 HGS-CVRP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12

3.2.2 PyVRP . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

3.2.3 Framework . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15

3.3 OR-Tools . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18

3.4 Hexaly . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19

**4 Experimental Results 20**

4.1 Optimization of synthetic datasets . . . . . . . . . . . . . . . . . . . . . . . . 20

4.2 Solution verification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

4.3 Solver comparison . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 25

4.4 Collection Time . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30

**5 Conclusion 32**

# 1 Introduction

As sustainability and environmental awareness have become more prevalent in Europe and Denmark in recent years, the management and recycling of private and commercial waste has also become more important and structured. In 2021, affaldsbekendtgørelsen (The Act of Waste) [1] was amended, standardizing waste management across Denmark. It was decided that household waste must be sorted in 10 waste fractions at household level everywhere in Denmark: residual, food, paper, cardboard, glass, metal, plastics, food and beverage cartons, textile and hazardous waste [1]. The 10 waste fractions are now collected at the vast majority of households in Denmark. The collection frequency of the waste fractions varies between municipalities; however, residual and food waste is often collected weekly or bi-weekly, while hazardous and textile waste is generally collected less often, for example every three months. Given the recurring nature of waste collection, the associated costs and importance in society it is essential that the collection, recycling and disposal processes are efficient and optimized.

## 1.1 Collection of textile and hazardous waste in Eastern Funen

The utility company Nyborg Forsyning & Service A/S (NFS), located in Nyborg, supplies various utilities, including waste management and operates three local recycling centers. The collection of residual, food, paper, cardboard, glass, metal, plastic and food and beverage cartons waste is outsourced to local contractors, while the utility company handles the collection of textile and hazardous waste itself. Hazardous waste must be handled carefully to comply with occupational safety and health regulations. Unlike other waste fractions hazardous waste must be transported in a closed container. Examples of hazardous waste include, but is not limited to, small batteries, fertilizers, paint, pesticides, electronics, cosmetics, medicine, oil, acids/bases, spraycans etc [2].

NFS is currently relying on manual planning and the drivers' experience to collect textile and hazardous waste. At the start of the year NFS schedules four collection weeks for the entire year where textile and hazardous waste is collected. Well in advance of the collection weeks a textile and hazardous waste collection registration form is opened on NFS' website. Customers register to have hazardous waste and/or textiles collected in the following collection week [2]. In 2024 the collections were carried out in week 7, 22, 39 and 47 which in total amounted to 1570 visited addresses. Each collection had a few hundred registrations. The registration form closes five days before the upcoming collection week and the registered customers are sorted into groups based on postcode and area.

The customers are instructed to place the container with hazardous waste and the textiles in a closed bag at the roadside no later than monday 8 am in the collection weeks. When the hazardous waste containers are placed at the roadside, they must be collected as soon as possible to minimize the risk of removal or opening by unauthorized persons. This is because such removal or opening could potentially lead to polution of the environment. The customer groups are driven according to which roads the drivers are familiar with, and the unknown roads are driven by GPS. The trucks start at the local recycling center with 37 empty containers, corresponding to the trucks' max capacities of hazardous waste, as will be decribed in greater detail in section 1.2. At each customer the customer's container of hazarous waste is swapped with an empty container, and if there is any textile waste it is also loaded onto the truck. Whenever a truck is filled to its max capacity, the truck returns to the recycling center to unload the waste and load 37 new empty containers. Any longer staff breaks are taken in continuation of the unloading and loading of the truck. Once any breaks have been taken and the truck is prepared, the truck proceeds to its next addresses. When all registered customers have been visited, the collection is concluded.

This bachelor project investigates the possibilities of optimizing the collection of hazardous waste (and textiles) by NFS. The main goal is to minimize the risk of harmful substances being released into the environment by accelerating the collection process to prevent the hazardous waste containers being opened or removed while they are at the roadside. Further, a central aim is to reduce unnecessary ressource consumption, consequently leading to cost savings for the company by minimizing the distance covered and routes driven.

## 1.2 Real-life data

Having established the importance of effective waste collection and provided an insight into how the collection of hazardous waste and textiles is currently carried out at NFS, this section will describe the available data before continuing to a literature review.

NFS has kindly agreed to provide company data and summary data of last year's collections to use in this project. Unfortunately, it is not possible to use actual registration data from last year's collections because of GDPR. Therefore, synthetic address and demand data, based on the summary data, is generated in section 1.3.

There are two heterogeneous trucks available for the collections. Each truck has got capacity for 37 containers of hazardous waste. The maximum amount of hazardous waste allowed is determined by safety regulations which state that a driver can transport a maximum of 300 kg of hazardous waste, without additional qualifications [3]. Each container can store at most 5,6 kg of hazardous waste; however, the utility company, in consultation with a safety advisor, has decided that no more than 37 containers should be transported at a time. Additionally, one truck can transport approximately 60-80 bags of textile waste in addition to the hazardous waste, while the other truck has got a smaller capacity of 30-40 textile bags. The company notes that they rarely transport more than 20 textile bags at a time. In the collection week one truck currently collects daily from 8:00 to approximately 15:00, while the other truck collects daily from 9-10:00 to 15-16:00 until the collection is concluded. The company estimates that they spend roughly 2 minutes at each customer to swap the hazardous waste containers and load the textile waste. The unloading of waste and loading of new empty containers at the recycling center takes approximately 20 minutes when no breaks are held. Around noon, the truck drivers take a 20-25 minute lunch break in continuation of unloading and loading the trucks.

Table 1 presents the summary data provided by NFS for the collections carried out in February, May, September and November in 2024. The data shows the number of registrations for hazardous and textile waste alongside the total number of registered customers to be visited in each collection.

| Collection\Postal codes | | 5540 (incl. 5550) | 5800 | 5853 | 5871 (incl. 5874) | Total |
|---|---|---|---|---|---|---|
| February (week 7) | Hazardous Waste | 51 | 167 | 35 | 14 | 267 |
| | Textiles | 16 | 61 | 14 | 6 | 97 |
| | **Customers visited** | **55** | **182** | **38** | **15** | **290** |
| May (week 22) | Hazardous Waste | 50 | 249 | 57 | 22 | 378 |
| | Textiles | 18 | 121 | 37 | 5 | 181 |
| | **Customers visited** | **58** | **279** | **66** | **22** | **425** |
| September (week 39) | Hazardous Waste | 75 | 299 | 84 | 35 | 493 |
| | Textiles | 33 | 120 | 35 | 11 | 199 |
| | **Customers visited** | **83** | **322** | **94** | **37** | **536** |
| November (week 47) | Hazardous Waste | 51 | 171 | 42 | 18 | 282 |
| | Textiles | 14 | 68 | 18 | 7 | 107 |
| | **Customers visited** | **56** | **192** | **51** | **20** | **319** |
| | **Total 2024** | **252** | **975** | **249** | **94** | **1570** |
| | | 16,1% | 62,1% | 15,9% | 6,0% | |

Table 1: Summary data provided by NFS for the collections of 2024.

While it is possible for customers to have more than one container of hazardous waste, it is uncommon for them to have multiple containers collected as seen in Table 1. Likewise, most customers have at most one bag of textile waste to be collected. Consequently, it is assumed that customers have at most one container and/or bag of textile waste to be collected in the synthetic datasets.

## 1.3 Synthetic data

Since actual address data from NFS cannot be used, realistic synthetic address data is generated to test the proposed solution. The data is generated by combining the summary data of

Table 1 with address data from OpenStreetMap [4] and distance data from Project OSRM [5]. Efficient filtering, random selection and demand simulation utilities are implemented in Python to manage the data. The following is a detailed outline of the synthetic data generation procedure.

- As a one-time procedure, a master address dataset of Eastern Funen is retrieved from OpenStreetMap with the data mining tool Overpass turbo [6]. This master dataset contains the fundamental address-related data such as street names, house numbers, postcodes and cities. Moreover, the data includes coordinates for all addresses that are necessary to determine the distance between locations with Project OSRM. The dataset also functions as a database, to which real-life customer data is matched to ensure data accuracy and provide coordinates.

The following steps are performed for each generated dataset:

1. The master address dataset is filtered to the postcodes of Table 1.
2. For each postcode, a specified number of addresses is selected randomly from the filtered dataset and the depot is inserted as the first entry in the data.
3. Collection demand is synthesized by random assignment across all postcodes. Initially, all addresses are assigned a collection demand of 0 for all waste types. Afterwards, a specified number of addresses for each waste type is randomly selected for which the collection demand is changed to 1.
4. A distance matrix of the fastest routes between all pairs of addresses in the synthetic dataset is generated with a locally hosted Project OSRM instance. Project OSRM uses a Multi-Level Dijkstra (MLD) algorithm to find the fastest routes between locations [5]. The distance matrix will serve as a cost matrix for optimization.

For analysis four datasets mirroring the four collections summarized in Table 1 is generated as described above. The synthetic datasets are illustrated in Figure 1, Figure 2, Figure 3 and Figure 4. In the figures customer locations are indicated by blue dots and the depot by a black triangle. The data used to illustrate the postal district borders is obtained from Geo Fyn [7].

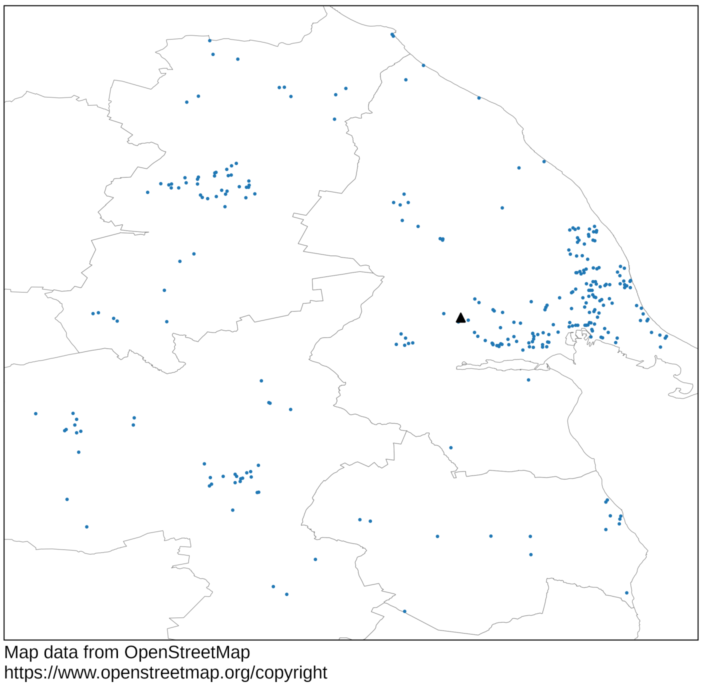
Map data from OpenStreetMap
https://www.openstreetmap.org/copyright

Figure 1: Synthetic Feb. dataset, 290 customers. Locations plotted on map of Eastern Funen.

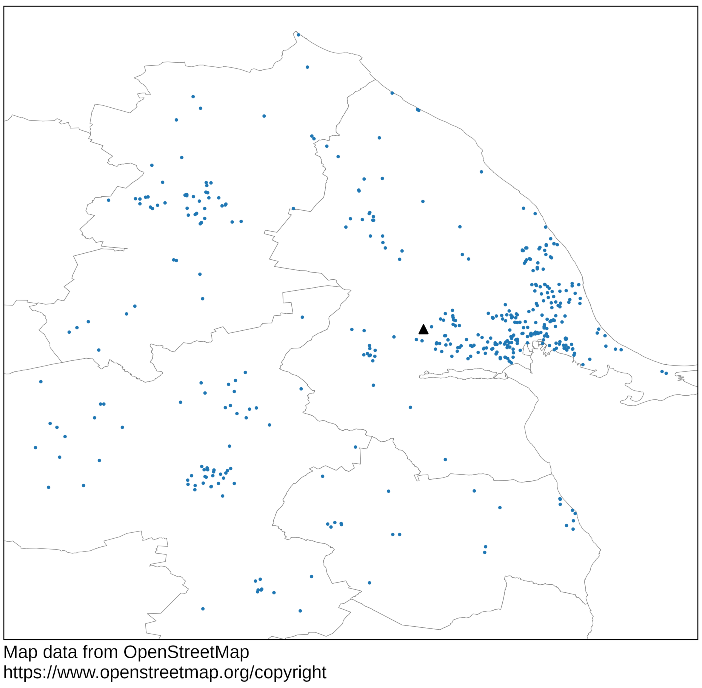
Map data from OpenStreetMap
https://www.openstreetmap.org/copyright

Figure 2: Synthetic May dataset, 425 customers. Locations plotted on map of Eastern Funen.

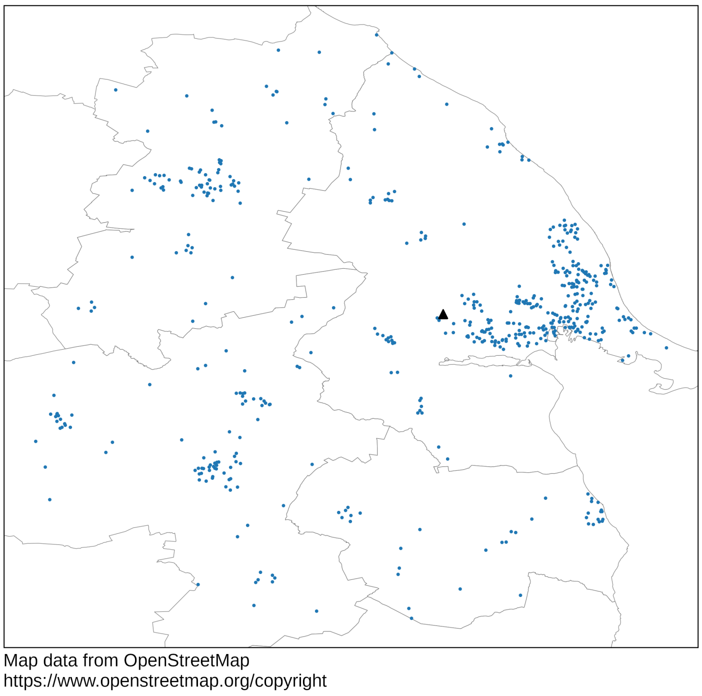
Map data from OpenStreetMap
https://www.openstreetmap.org/copyright

Figure 3: Synthetic Sep. dataset, 536 customers. Locations plotted on map of Eastern Funen.

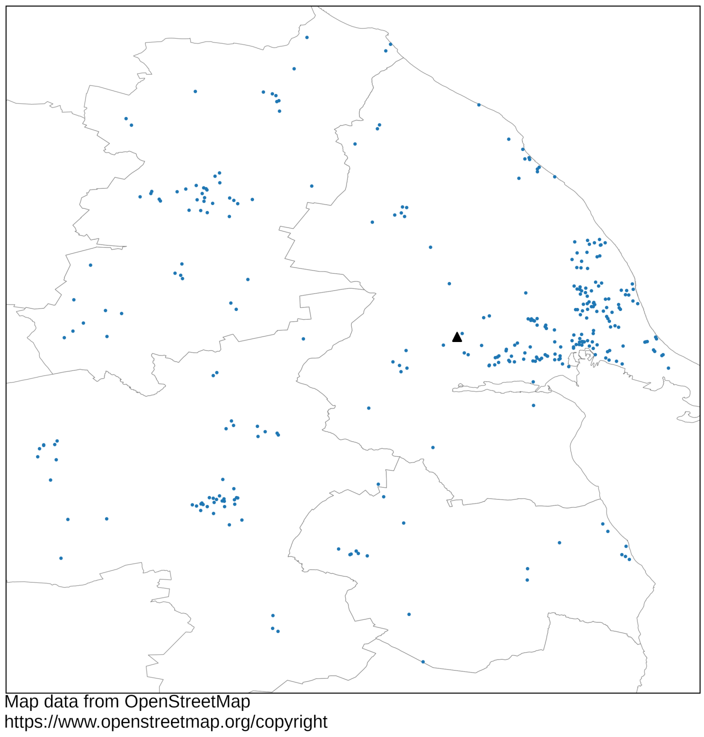
Map data from OpenStreetMap
https://www.openstreetmap.org/copyright

Figure 4: Synthetic Nov. dataset, 319 customers. Locations plotted on map of Eastern Funen.

# 2 Literature Review

A significant amount of literature has been published in the field of Waste Collection Problems (WCPs), with optimization of collection routes being one of the most examined areas; However, it seems that there has been relatively few studies focused on large-scale route optimization of WCPs with geographically dispersed nodes. In this context large-scale refers to problems that requires visiting more than 200-300 nodes. Recent literature on modelling and optimization of Vehicle Routing Problems (VRPs) in the context of WCPs is reviewed in this section. The review is conducted with a focus on how medium and large-scale WCPs are modelled, and further what the best existing open-source solvers or algorithms are that can be applied to solve these problems. It is interesting to investigate freely available solutions, as the field of route optimization in practice often is dominated by commercial solutions such as Cplex [8], Gurobi [9] and Hexaly [10]. The dominance of commercial solutions can hinder accessibility for companies where the need is intermittent and the cost of commercial solutions is not justifiable.

## 2.1 Existing vehicle routing problem models in waste collection

VRPs in the context of WCPs are most often modelled in Mixed-Linear Integer Programming (MILP) terms and come in several variations where different constraints are considered. In [11] they implement a Capacitated Vehicle Routing Problem (CVRP) in combination with a backtracking search algorithm and test it on instances with up to 100 nodes with good results [11]. The CVRP is sometimes refered to as the standard VRP variant, and it was first introduced in 1959 [12]. It is a problem in which some cost associated with the routes travelled, often travel distance or time spent, is minimized while collecting or delivering some goods from/to a set of customers, from a depot using a fleet of vehicles with limited capacities [12]. An elaborate VRP with Time Windows (VRPTW) model featuring multiple disposal facilities, driver rest periods and time windows for customers, depots and facilities is considered in [13]. Metaheuristics based on variable neighborhood and tabu searches are implemented sucessfully for instances up to 2092 nodes and 19 waste disposal facilities [13]. Further, the family of VRPs has also been applied to real-world hazardous waste collection scenarios. In [14] they model the collection of health-care waste (e.g. sharp objects and chemicals) from geographically dispersed hospitals as a CVRP. An adapted evolutionary approach combining genetic and local search operators is shown to outperform another heuristic method [14].

Other studies such as [15] model waste collection as Capacitated Arc Routing Problems (CARPs). In the study by [15] real-world examples of waste collection in six danish municipalities is examined. The collection of general household waste from instances with up to 11.640 nodes is modelled and a constructive CARP heuristic is developed, outperforming a comparable heuristic on speed for large-scale instances [15]; However, CARPs are generally not employed in the context of WCPs with geographically dispersed nodes. CARPs are better suited for another category of WCPs, namely the WCPs where all addresses along the arcs have got waste to be collected [16], such as collection of general household waste as modelled in [15].

Furthermore, there is a vibrant competition scene for the development of better VRP algorithms and solvers which is explored in the following section.

## 2.2 Solving vehicle routing problems

The family of VRPs was formally shown to be NP-hard in 1981 by [17]. The NP-hard nature of VRPs necessitates the use of efficient algorithms to solve VRP models of real-life scenarios, which often can have hundreds of locations to visit. Consequently, medium and large-scale VRPs are predominantly solved with heuristic and metaheuristic methods, as evidenced by the methods employed in the literature reviewed in the preceding section. [18] clarifies that the primary challenge of exact methods is the large numbers of variables and constraints, which requires powerful computers and a lot of time to optimize larger instances. Heuristic methods are often able to deliver satisfactory solutions in a shorter time span [18]. The superiority of heuristic methods is further validated by winning solvers of recent VRP competitions.

In the VeRoLog Solver Challenge of 2019 focused on a variation of the multiperiod VRP [19] the winning solver utilized a combination of heuristic methods joining Large Neighborhood Search (LNS) with Variable Neighboorhood Descent (VND) in an adaptable framework [20]. In the DIMACS Implementation Challenge in 2022 eight different VRP variations including the CVRP were represented [21]. The open-source heuristic Hybrid Genetic Search (HGS) [22, 23, 24] demonstrated a notable prevalence in the CVRP category, by being employed in conjunction with other algorithms in six of the eight finalists' solvers [21, 25, 26, 27, 28, 29, 30]. In particular the algorithm was utilized in the winning solvers [25, 31] of both the CVRP and VRPTW categories [21].

The HGS algorithm was initially proposed for the multidepot VRP (MDVRP), periodic VRP (PVRP) and combinations of these in 2012 [22]. The algorithm is population-based, combining genetic crossovers with neighborhood search techniques [22, 23]. HGS was reimplemented in an improved version known as HGS-CVRP specialized to tackle CVRPs in 2022 [23]. It is shown that the improved version achieves equivalent solution quality to the original implementatation in a substantially reduced time span [23]. In addition, the improved version outperforms a selection of other state-of-the-art algorithms across a range of different instances [23]. As noted, the algorithm is open-source. Wrappers are available for C, Python and Julia [24]. Moreover, the HGS-CVRP based winner in the VRPTW category Router [31] is available as an open-source Python library that handles a selection of VRP variants [32]. The library, which is known as PyVRP, lowers the barriers to entry even further by implementing extra functionality, and it acts as a user-friendly interface by making the performance critical code, implemented in C++, accessible in Python [32]. It is noteworthy that three other finalists' solvers from the

CVRP category of the DIMACS challenge are also open-source, albeit implemented in more challenging programming languages: MDM-HGS in C++ [33, 26], HGSRR in Rust [34, 28] and the AILSII solver implemented in Java [35, 36].

Outside the competitive scene there are well-known open-source solvers such as VRPSolverEasy [37], VROOM [38], Jsprit [39], OptaPlanner [40] and Google's OR-Tools Routing library [41]. VRPSolverEasy is a Python library for the exact solver VRPSolver. It is shown to perform very well for instances with up to 100 customers and even being able to solve some instances with up to 200 customers. It utilizes advanced Branch-Cut-and-Price (BCP) algorithms based on the best existing BCP algorithms [37]. However, VRPSolverEasy appears to only be available for academic usage. VROOM is a C++ based solver developed by the company Verso to power their routing API. It is able to use data from a selection of routing engines such as Project OSRM. Tested on a range of benchmark instances for CVRPs, with sizes ranging from 15 to 1000 locations and number of vehicles from 2 to 207, it is shown to provide solutions with [0%; 9,37%] gaps from the best known solutions [38]. Jsprit and OptaPlanner are both metaheuristic solvers implemented in Java. It is claimed that Jsprit and OptaPlanner are lightweight, flexible and easy-to-use; however, it appears that there is no available benchmark information for the solvers [39, 40]. OR-Tools provides a (meta-)heuristics based VRP solver implemented in C++ with wrappers for Python, C# and Java [41]. In the comprehensive comparison of state-of-the-art solvers by [23] OR-Tools is generally found to be outperformed by all other solvers in the comparison.

Given the documented succes of VRP variations being used to model WCPs, the present problem will be modelled as a CVRP variation known as a Multi-compartment CVRP (MCVRP) in MILP terms. The MCVRP is a CVRP that features different capacities for multiple different types of goods, in this case waste types. In light of the superiority of HGS-based solvers in the competitive environment, HGS-CVRP will be employed through the Python library PyVRP [32] in combination with custom code to provide a framework that can handle optimization of WCPs in the context as described in section 1.1 and section 1.2. In addition, to provide a first-hand impression of how the proposed framework using PyVRP compares to other solvers, the solvers Gurobi [9], OR-Tools [41] and Hexaly [10] will also be applied to the problem.

## 3 Solution Methods

Having gained an overview of the situation, available data and an insight into relevant literature, the following sections first presents the problem formally as a MCVRP model in MILP terms and provides its implementation with Gurobi. Subsequently, the HGS-CVRP algorithm and the solver PyVRP are examined in detail, before introducing the proposed framework and the implementation with PyVRP. Finally, the solvers OR-Tools and Hexaly are briefly explored, and their implementations are presented.

## 3.1 A Mixed Integer Linear Programming Model

The MCVRP model presented in this section is based on the well-known Miller-Tucker-Zemlin-based (MTZ) CVRP formulation, which was initially presented by [42] and subsequently improved and corrected by [43] in 2004. The choice to formulate a MCVRP model based on this model is because it is relatively simple and compact (in relation to the number of constraints). The compactness is beneficial when the model is attempted to be solved with Gurobi on a computer with limited memory in section 4. An alternative less compact model was also tested; however, it was not possible to load this alternative model on the computer because of its size. The MTZ subtour constraints adds a polynomial number of constraints, whereas e.g. the well-known Dantzig-Fulkerson-Johnson (DFJ) subtour elimination constraints adds an exponential number of constraints [44].

### 3.1.1 Sets

Let $V = \{0, 1, 2, \ldots, n\}$ denote the set of all nodes (locations) where node 0 is the depot, n is the number of customers in the collection and the set of customers is denoted by $V\backslash\{0\}$.

Let $G = (V, A)$ be a graph where A is the set of arcs $\text{Arc}(i, j) \in A$.

Let $T = \{1, 2, \ldots, H\}$ be the set of vehicle types where $H$ is the number of vehicle types.

Let $K = \{1, 2, \ldots, P\}$ be the set of available vehicles where $P$ is the number of available vehicles.

Let $K_t = \{k \in K \mid m(k) = t\}$ be the sets of available vehicles of type $t \in T$ where $m(k)$ is a function mapping vehicle $k$ to its type $t$.

Let $W = \{1, 2, \ldots, M\}$ be the set of waste types where M is the number of waste types.

### 3.1.2 Variables

$c_{ij}$ is the distance of the fastest route from node $i$ to node $j$ for $i, j \in V$.

$d_{iw}$ is the demand of customer $i \in V\backslash\{0\}$ for waste type $w \in W$ to be collected.

$Q_{tw}$ is the capacity of vehicle type $t \in T$ for waste type $w \in W$.

$u_{iwt}$ denotes the label setting variables for customer $i \in V\backslash\{0\}$, vehicle type $t \in T$ and waste type $w \in W$.

### 3.1.3 Decision variables

$$x_{ijt} = \begin{cases} 1 & \text{if} \quad \text{a vehicle of type } t \in T \text{ travels over the Arc}(i,j) \in A \\ 0 & \text{otherwise.} \end{cases}$$

### 3.1.4 Model

$$
\begin{aligned}
\min \quad & \sum_{i,j\in V, i\neq j} \sum_{t\in T} c_{ij} \cdot x_{ijt} && && (1)\\
\text{s.t.} \quad & \sum_{j\in V\setminus\{0\}} x_{0jt} \le |K_t|, && \forall t \in T && (2)\\
& \sum_{j\in V, i\neq j} \sum_{t\in T} x_{ijt} = 1, && \forall i \in V\setminus\{0\} && (3)\\
& \sum_{j\in V, i\neq j} x_{ijt} = \sum_{j\in V, i\neq j} x_{jit}, && \forall i \in V\setminus\{0\}, \forall t \in T && (4)\\
& u_{iwt} - u_{jwt} + Q_{tw}\cdot x_{ijt} && && \\
& \quad + \left(Q_{tw} - d_{iw} - d_{jw}\right)\cdot x_{jit} \le Q_{tw} - d_{jw}, && \forall i,j \in V\setminus\{0\}, i\neq j, \forall t\in T, \forall w \in W && (5)\\
& u_{iwt} \le Q_{tw} && && \\
& \quad - \left(Q_{tw} - \max_{j\in V\setminus\{0\}, i\neq j} d_{jw} - d_{iw}\right)\cdot x_{0it} && \forall i \in V\setminus\{0\}, \forall t\in T, \forall w\in W && (6)\\
& \quad - \sum_{j\in V\setminus\{0\}, i\neq j} d_{jw} x_{ijt}, && && \\
& u_{iwt} \ge d_{iw} + \sum_{j\in V\setminus\{0\}, i\neq j} d_{jw}\cdot x_{jit}, && \forall i \in V\setminus\{0\}, \forall t\in T, \forall w\in W && (7)\\
& x_{ijt} \in \{0,1\}, && \forall i,j \in V, i\neq j, \forall t\in T && (8)\\
& u_{iwt} > 0, && \forall i \in V\setminus\{0\}, \forall w\in W, \forall t\in T. && (9)
\end{aligned}
$$

The objective function (1) states that the total distance of the routes should be minimized. Constraints (2) states that no more than the available number of vehicles of each type can leave the depot. Constraints (3) states that each customer is visited by a single vehicle. Constraints (4) states that all vehicles entering a node must leave the node. Constraints (5)-(7) are the subtour elimination constraints and the constraints enforcing the capacity constraints. Constraints (8)-(9) states that the decision variables are binary and that the label variables are positive.

### 3.1.5 Implementation with Gurobi

Gurobi is a commercial mathematical optimization-based solver that utilizes exact methods, such as the simplex method, and a selection of mixed-integer programming-specific heuristic methods [9] to solve linear programming problems. As Gurobi is based on mathematical optimization, the implementation of the MILP model closely resembles the model itself. The implementation of the MCVRP model with Gurobi in Python is shown in Listing 1.

```python
m = gp.Model("MCVRP")
valid_moves = [(i,j,t) for i in locations for j in locations if i!=j for t in vehicle_types]

x = m.addVars(valid_moves, vtype=GRB.BINARY, name="x")
u = m.addVars(customers, wastes, vehicle_types, lb=0, vtype=GRB.CONTINUOUS, name="u")
```

```
6  m.setObjective(gp.quicksum(distance_matrix[i][j]*x[i,j,t] for (i,j,t) in valid_moves),
       ↪ GRB.MINIMIZE)

8  m.addConstrs(gp.quicksum(x[0,j,t] for j in customers if (0,j,t) in x)  <= num_trucks[t] for t
       ↪ in vehicle_types)

10 m.addConstrs(gp.quicksum(x[i,j,t] for j in locations for t in vehicle_types if (i,j,t) in x)
       ↪ == 1 for i in customers)

12 m.addConstrs(((gp.quicksum(x[i,j,t] for j in locations if (i,j,t) in x) ==
       ↪ gp.quicksum(x[j,i,t] for j in locations if (j,i,t) in x)) for i in customers for t in
       ↪ vehicle_types))

14 m.addConstrs(u[i,w,t] - u[j,w,t] + capacities[t][w]*x[i,j,t] + (capacities[t][w] -
       ↪ demands[i][w] - demands[j][w])*x[j,i,t] <= capacities[t][w] - demands[j][w] for i in
       ↪ customers for j in customers for t in vehicle_types if (i,j,t) in x and (j,i,t) in x
       ↪ for w in wastes)

16 m.addConstrs((u[i,w,t] <= capacities[t][w] - (capacities[t][w] - max([demands[j][w] for j in
       ↪ customers if j!=i]) - demands[i][w])*x[0,i,t] - gp.quicksum(demands[j][w]*x[i,j,t] for
       ↪ j in customers if (i,j,t) in x) for i in customers for t in vehicle_types for w in
       ↪ wastes))

18 m.addConstrs(u[i,w,t] >= demands[i][w] + gp.quicksum(demands[j][w]*x[j,i,t] for j in
       ↪ customers if (j,i,t) in x) for i in customers for w in wastes for t in vehicle_types)

20 m.optimize()
```

Listing 1: The MILP model implemented using Gurobi in Python.

In Listing 1, line 1 initializes a model, line 2 defines the valid moves, lines 4-5 defines the decision variables and label setting variables, line 6 defines the objective function, lines 8-12 states that only the available number of vehicles can be used, customers are visited exactly once and that vehicles must leave the nodes they enter, lines 14-18 are the subtour elimination and capacity constraints and finally line 20 invokes the solver to optimize the model.

## 3.2 Hybrid Genetic Search

### 3.2.1 HGS-CVRP

As previously mentioned, HGS-CVRP is a population-based heuristic algorithm combining genetic crossovers with neighborhood search techniques [23]. In this section, the algorithm is explained in a step-by-step fashion, based on the elaborate articles of the algorithm's author [22, 23], to facilitate understanding of its inner workings. The algorithm works by first initializing a start population of $4\mu$ random solutions. The initial solutions are improved by local search methods (as in step 3 below) before being sorted into a feasible and infeasible subpopulation. Each subpopulation has size in the interval $[\mu; \mu + \lambda]$ where $\mu$ is the minimum population size and $\lambda$ is the generation size. After initializing the start population the algorithm enters the following iterative procedure to improve the solutions further [23].

1. Selecting two parent solutions

All solutions in the population are characterized by rank in terms of solution quality and diversity contribution. Each of the two parent solutions are found by randomly selecting two solutions from the population with all solutions having equal probability to be selected. The two selected solutions are compared on their "fitness score" and the most fit solution is chosen to be the parent. This fitness score is calculated as a weighted sum of the solution quality rank and diversity contribution rank [23].

2. Crossing over the parent solutions to produce a new child solution

A child solution is created by crossing over the parent solutions by an Ordered Crossover (OX). In the crossover the child solution inherits a random part of one parent solution and inherits the missing visits from the other parent solution. This crossover operation ignores depot visits and hence does not respect capacity constraints. In order to produce a complete CVRP solution, an efficient algorithm known as SPLIT is employed after each crossover. The SPLIT algorithm optimally reinserts depot visits as delimiters in the child after the crossover operation to produce complete CVRP solutions [23].

3. Improving the child solution by local neighborhood search

The child solution is improved by a local neighborhood search. The search uses RELOCATE, SWAP, SWAP*, 2-OPT and 2-OPT* moves to explore local neighborhoods consisting of geographically close node pairs $(i, j)$ where $i$ is a node and $j$ is its neighbor. The local neighborhood size is controlled by a granularity parameter $\Gamma$ which limits the set of node pairs such that $j$ belongs to the $\Gamma$ closest nodes to node $i$. The search is performed iteratively, and for each node all possible move types are evaluated in random order with its local neighborhood. Any improving move is applied immediately. The search terminates when arriving at a local minimum [23].

The move types can be outlined as follows. The RELOCATE moves relocates node $i$ (and possibly its successor $i + 1$) to behind its neighbor $j$. The SWAP moves swap node $i$ (and possibly its successor $i + 1$) with its neighbor $j$ (and possibly the neighbor's sucessor $j + 1$). If node $i$ and its neighbor $j$ are in the same route the single 2-OPT move replaces Arc$(i, i + 1)$ and Arc$(j, j + 1)$ with Arc$(i, j)$ and Arc$(i + 1, j + 1)$ respectively. If node $i$ and its neighbor $j$ are not in the same route the 2-OPT* moves replaces Arc$(i, i + 1)$ and Arc$(j, j + 1)$ by Arc$(i, j)$ and Arc$(i + 1, j + 1)$ or by Arc$(i, j + 1)$ and Arc$(i + 1, j)$ [22]. Finally, the most advanced move is the SWAP* move. Unlike the standard SWAP move that interchanges nodes in place, SWAP* can insert nodes anywhere in the routes. The move is useable on nodes $i$ and $j$ not in the same route, i.e. node $i$ can be inserted anywhere in the route of its neighbor $j$ and $j$ can similarly be inserted anywhere in the route of node $i$. The computational complexity of searching all SWAP* moves is very high and proportional to $\Theta(n^3)$ hence the algorithm utilizes efficient search techniques and is restricted to explore only those routes that have intersecting polar sectors from the depot. The additional restrictions and efficient search techniques reduces the time to explore the SWAP* moves to a level that is comparable to the exploration time of the other move types [23].

If the solution is infeasible after the local search procedure, the local search procedure is run again with 50% probability, but this time with 10 times larger penalty coefficients on solution infeasibility in an effort to recover a feasible solution. This procedure is known as the REPAIR operation [23].

4. Inserting the resulting child solution into the solution population

The produced child solution is inserted in either the feasible or infeasible subpopulation depending on its status. When a subpopulation reaches its maximum size $\mu + \lambda$ an elimination procedure starts. $\lambda$ solutions are eliminated from the subpopulation iteratively by first removing identical solutions and then the least fit solutions. To ensure the generation of a satisfying number of feasible solutions at the end of the local search procedure, the penalty coefficients that control the solution infeasibility are adjusted throughout the local search process [23].

The algorithm terminates by some stopping criteria, i.e. a maximum runtime or a maximum number of iterations without improving [23].

### 3.2.2 PyVRP

The PyVRP library uses a slightly different version of the HGS-CVRP algorithm comparable to the original HGS-CVRP implementation. PyVRP also employs a genetic algorithm, maintains a population with feasible and infeasible subpopulations and improves its solutions through local search methods in an iterative manner [32].

In PyVRP the parents for the crossover are by default selected through a 2-way tournament based on fitness score as in the original algorithm. The fitness score in PyVRP closely resembles that of the original algorithm. On the contrary, PyVRP employs a Selective Route Exchange (SREX) crossover operator for VRPs [32, 31]. SREX is different from OX in that it combines entire routes from the two parent solutions in the child solution. A selection of routes from the first parent solution is replaced with similar routes from the second parent solution. Since entire routes are combined from two parents, nodes might be present in two routes. One parent is selected to have all duplicate nodes removed from the routes originating from it. Any missing nodes are inserted into the routes to minimize the detour distance from existing routes. Unlike OX, SREX preserves the depots and hence does not need to reinsert the depots [31]. In the local search PyVRP utilizes moves that closely resembles the moves of the original algorithm; however, they have been implemented slightly differently by collecting and combining the moves in fewer functions [32]. To manage the populations PyVRP uses the same approach as the original algorithm. When a subpopulation grows to its maximum size, identical solutions are first eliminated and then the least fit solutions [32]. While the original HGS-CVRP algorithm is entirely implemented in C++ [23], only the performance-critical code in PyVRP is implemented in C++ e.g. the local search, while the rest of the code is implemented in Python [32]. The PyVRP solver has been benchmarked against the orginal HGS-CVRP algorithm and Best-Known Solutions (BKS) across 100 CVRP instances, covering a wide range of characteristics and instance sizes. The orginal HGS-CVRP algorithm achieves a mean gap to BKS of 0,11% across the instances, and slightly outperforms PyVRP that achives a mean gap to BKS of 0,22% [32].

### 3.2.3 Framework

The proposed WCP framework, which is facilitated by the extensive functionality of the PyVRP solver, can be outlined as follows.

1. The synthetic data (described in section 1.3) is loaded and prepared to be used in the model.

2. As the PyVRP solver does not support vehicles making multiple trips and cannot automatically determine the optimal number of vehicles to use, a vehicle set formation function is utilized. The set formation function works by identifying the smallest set of vehicles needed to collect all waste for a given VRP instance, and thus ensures the number of routes is minimized. In practice the function works by adding copies of the unique vehicles to the set of available vehicles $K$, to expand the set of available vehicles for the solvers. Each additional copy of a vehicle reflects that the vehicle is available to take an additional route. The function works as follows. The ratios of demand to capacity for each waste type $w \in W$ is computed by

$$R_w = \frac{\sum_{i \in V \setminus \{0\}} d_{iw}}{\sum_{t \in T} Q_{tw} \cdot |K_t|}$$

if $\max_{w \in W}(R_w) \leq 1$ the number of available vehicles $K$ is feasible and the framework continues to the next step. If $\max_{w \in W}(R_w) > 1$ an iterative procedure is started in which the set of available vehicles $K$ is expanded repeatedly, with copies of the unique vehicles, until $\max_{w \in W}(R_w) \leq 1$. The vehicles are added in a sub-iterative procedure in which the waste types demanding the largest increase in capacity are prioritized. Therefore the vehicles are copied to $K$ in descending order with respect to the capacities of the vehicle types $T$ for the most prioritized waste types.

To illustrate the iterative procedure consider the following small example based on the two vehicles available at NFS. Both vehicles can transport 37 containers of hazardous waste. Vehicle 1 can additionally transport 60 bags of textile waste, and vehicle 2 can transport 30 bags of textile waste in addition to the hazardous waste. Assume that there is demand to have 100 containers of hazardous waste and 120 bags of textile waste collected in total. Initially, the ratios for hazardous and textile waste are 1,35 and 1,33 respectively. Thus the vehicle set formation function will prioritize adding a copy of the vehicle type with the greatest capacity for hazardous waste; however, since both vehicle types have got the same capacity for hazardous waste, the function adds a copy of the vehicle type with the largest capacity for textile waste first, namely vehicle 1. After adding an extra copy of vehicle 1, the ratios for hazardous and textile waste are 0,9 and 0,8 respectively, and the iterative procedure terminates. The vehicle set formation function is summarized in Algorithm 1.

**Algorithm 1:** Vehicle set formation

```
1  $T$ set of vehicle types;
2  $K$ set of vehicles;
3  Compute $R_w$ for $w \in W$;
4  while $\max_{w \in W}(R_w) > 1$ do
5      $w^* = \text{argmax}_w R_w$;
6      $L$ = list of vehicle types $T$ sorted in decreasing order of
         capacity for $w \in W$for decreasing order of $R_w$ for $w \in W$;
7      while $R_{w^*} > 1$ do
8          $\gamma$ = extract first element of L and move it to the end of L;
9          add $\gamma$ to $K$;
10         Compute $R_{w^*}$;
11     end
12 end
```

3. A PyVRP model is created, the found set of vehicles and the customer, demand and distance data previously prepared is added to the model. It is important to note that PyVRP converts all numeric input values to integers [32]. Therefore, the distance data from Project OSRM is multiplied by 10, before being loaded into the model, to ensure that no data is lost. Similarly, the results produced by PyVRP are divided by 10. The framework handles the scaling of values automatically.

4. The PyVRP solver is invoked to optimize the MCVRP. Two stopping criteria are set to make sure the solver does not run for an unnecessarily long time. The solver is invoked with a default max runtime of 900 seconds and a maximum of 100.000 iterations without improvement; however, these criteria can easily be changed by the user. The decision to set a default maximum runtime of 900 seconds is subjectively based on what is considered a reasonable waiting time.

The implementation of the MCVRP in the proposed framework using the PyVRP solver is based on the CVRP example code from the PyVRP documentation [32]. The implementation with PyVRP, excluding any other auxiliary code, is shown in Listing 2.

```
m = Model()

for t in range(0,len(num_trucks)):
    m.add_vehicle_type(num_trucks[t], capacity=capacities[t])

depot = m.add_depot(x=coordinates[0][0], y=coordinates[0][1])

clients = []
for i in range(1,len(coordinates)):
    if len(demands[0]) == 1:
```

```
11          to_deliver = int(demands[i][0])
12      else:
13          to_deliver = []
14          for subdemand in demands[i]:
15              to_deliver.append(int(subdemand))

17      client = m.add_client(x=int(coordinates[i][0]), y=int(coordinates[i][1]),
            ↪ delivery=to_deliver)
18      clients.append(client)

20  locations = [depot] + clients
21  i,j = 0,0
22  for frm in locations:
23      for to in locations:
24          m.add_edge(frm, to, distance=distance_matrix[i][j]*10)
25          j+=1
26      i+=1
27      j=0

29  res = m.solve(seed = seed, stop=MultipleCriteria([MaxRuntime(runtime),
        ↪ NoImprovement(100000)]))
```

Listing 2: The MCVRP implemented in the proposed framework using the PyVRP solver.

In Listing 2, line 1 initializes a model and lines 3-4 adds the vehicles to the model. The vehicles are added to the model by specifying the number of vehicles available of each type and passing lists of each vehicle type's capacities. Line 6 adds the depot to the model. Lines 8-18 creates and adds each customer to the model by specifying the customers' coordinates and passing lists of the customers' demands for each waste type to be collected. In lines 20-27 the edges between all locations are added to the model by specifying the distance from each node to all other nodes. The distances are multiplied by 10 to ensure that no information about the problem is lost, when the distances are converted to integers. Line 29 invokes the PyVRP solver to optimize the problem and sets the stopping criteria.

In addition to the main PyVRP-based implementation, the MILP model described in section 3.1 is implemented in the framework with the open-source Python library PuLP [45] and Gurobi. The MILP model has been implemented in the framework to enable the possibility of employing exact methods. However, as previously explained, this is mostly only possible for smaller problem instances. The MILP model is implemented in a slightly more restricted version by changing constraints (2) from inequalities to equalities. The constraints are changed since the number of needed vehicles, to collect all the waste, is known from the set formation function explained above. Otherwise, the implementation follows the implementation shown in Listing 1 with Gurobi. The implementation for PuLP closely resembles the implementation with Gurobi.

The framework is implemented to utilize Gurobi as the MILP solver if it is available. Otherwise, the framework resorts to PuLP and open-source solvers that are less performant but available to all users.

## 3.3 OR-Tools

In the context of VRPs OR-Tools [41] employs a two-step procedure to produce its solutions. First an initial solution is found with a first solution strategy. The first solution strategy used, in the examples of the OR-Tools documentation, is a cheapest arc heuristic, but other strategies such as the Savings and Christofides algorithms are available. Since the cheapest arc algorithm is used in the examples of the documentation, it will also be employed as the first solution strategy for the comparison in section 4. The cheapest arc algorithm starts at the depot, and then adds the next node that creates the cheapest route segment to the route in an iterative procedure. The iterative procedure continues until the capacity constraints for the vehicle are met. It then proceeds to the next vehicle and so on to create a complete set of routes [41]. Subsequently, OR-Tools applies a local search strategy to improve the initial solution until the time limit is reached. In the documentation of OR-Tools, the recommended local search strategy is Guided Local Search (GLS). GLS is a metaheuristic algorithm capable of escaping local minima. OR-Tools states that GLS generally is the most efficient metaheuristic for VRPs [41], and hence it will be applied as the local search strategy in the comparison.

The implementation of the MCVRP using OR-Tools is based on the CVRP example code from the OR-Tools documentation [41]. The OR-Tools implementation, excluding any auxiliary code, is shown in Listing 3.

```python
manager = pywrapcp.RoutingIndexManager(len(data["distance_matrix"]), data["num_vehicles"],
    ↪ data["depot"])

routing = pywrapcp.RoutingModel(manager)

def distance_callback(from_index, to_index):
    from_node = manager.IndexToNode(from_index)
    to_node = manager.IndexToNode(to_index)
    return data["distance_matrix"][from_node][to_node]

transit_callback_index = routing.RegisterTransitCallback(distance_callback)
routing.SetArcCostEvaluatorOfAllVehicles(transit_callback_index)

def demand_callback_hazardous(from_index):
    from_node = manager.IndexToNode(from_index)
    return data["demands_hazardous"][from_node]

demand_callback_hazardous_index =
    ↪ routing.RegisterUnaryTransitCallback(demand_callback_hazardous)
routing.AddDimensionWithVehicleCapacity(
    demand_callback_hazardous_index,
    0,  # null capacity slack
    data["vehicle_capacities_hazardous"],
    True,  # start cumul to zero
    "Capacity")

def demand_callback_textile(from_index):
    from_node = manager.IndexToNode(from_index)
    return data["demands_textile"][from_node]

demand_callback_textile_index = routing.RegisterUnaryTransitCallback(demand_callback_textile)
routing.AddDimensionWithVehicleCapacity(
31      demand_callback_textile_index,
32      0,  # null capacity slack
33      data["vehicle_capacities_textile_waste"],
34      True,  # start cumul to zero
35      "Capacity")

37  search_parameters = pywrapcp.DefaultRoutingSearchParameters()
38  search_parameters.first_solution_strategy =
        ↪ (routing_enums_pb2.FirstSolutionStrategy.PATH_CHEAPEST_ARC)
39  search_parameters.local_search_metaheuristic =
        ↪ (routing_enums_pb2.LocalSearchMetaheuristic.GUIDED_LOCAL_SEARCH)

41  solution = routing.SolveWithParameters(search_parameters)
```

Listing 3: The MCVRP implemented using OR-Tools.

In Listing 3, lines 1-3 makes the data available to the model and initializes the model with the number of customers, number of vehicles and the depot. Lines 5-11 defines a distance callback function and instructs OR-Tools to use the distances from the distance matrix as the arc costs. Lines 13-15 defines a demand callback function to get the amount of hazardous waste at customers. Lines 17-23 adds the capacity constraints for hazardous waste using the demand callback function and the vehicles' capacities for hazardous waste. Lines 25-35 defines a demand callback function for textile waste and similarly adds the capacity constraints for textile waste. Lines 37-39 instructs OR-Tools to use the cheapest arc algorithm as the first solution strategy and GLS as the local search strategy. Line 41 invokes the solver to optimize the problem.

## 3.4 Hexaly

Hexaly is a commercial closed-source solver, so the methods used in the solver are not publicly known; however, Hexaly is described as a new kind of mathematical optimization solver combining exact methods with heuristic methods such as local-search heuristics [10]. Hexaly uses a different and more mathematical set-based modelling API compared to e.g. Gurobi. Hexaly uses sets, lists, intervals and other mathematical concepts. Hexaly claims to excel in speed for combinatorial problems such as routing, scheduling and packing problems [10]. The implementation of the MCVRP using Hexaly is based on the CVRP example code from the Hexaly documentation [10]. The Hexaly implementation, excluding any auxiliary code, is shown in Listing 4.

```
1   with hexaly.optimizer.HexalyOptimizer() as optimizer:
2       model = optimizer.model

4       customers_sequences = [model.list(nb_customers) for _ in range(nb_trucks)]
5       model.constraint(model.partition(customers_sequences))

7       demands_haz = model.array(demands_data_haz)
8       demands_tex = model.array(demands_data_tex)
9       customer_dist_matrix = model.array(customer_to_customer)
10      depot_to_customers_matrix = model.array(depot_to_customers)
11      customers_to_depot_matrix = model.array(customers_to_depot)

13      dist_routes = [None] * nb_trucks
```

```
14     for k in range(nb_trucks):
15         sequence = customers_sequences[k]
16         c = model.count(sequence)

18         demand_lambda_haz = model.lambda_function(lambda j: demands_haz[j])
19         route_quantity_haz = model.sum(sequence, demand_lambda_haz)
20         model.constraint(route_quantity_haz <= truck_capacities_haz[k])

22         demand_lambda_tex = model.lambda_function(lambda j: demands_tex[j])
23         route_quantity_tex = model.sum(sequence, demand_lambda_tex)
24         model.constraint(route_quantity_tex <= truck_capacities_tex[k])

26         dist_lambda = model.lambda_function(lambda i: model.at(customer_dist_matrix,
               ↪ sequence[i - 1], sequence[i]))
27         dist_routes[k] = model.sum(model.range(1, c), dist_lambda) + model.iif(c > 0,
               ↪ depot_to_customers_matrix[sequence[0]] + [sequence[c - 1]], 0)

29     total_distance = model.sum(dist_routes)
30     model.minimize(total_distance)
31     model.close()
32     optimizer.solve()
```

Listing 4: The MCVRP implemented using Hexaly.

In Listing 4, lines 1-2 initializes a model. Line 4 creates a sequence of customers that can be visited by each truck. Line 5 adds the constraint that each customer must be visited by exactly one truck. Lines 7-11 creates Hexaly arrays of the data. Line 13 defines an empty list of the distances each truck traverses. Lines 14-27 form a for loop that iterates over each truck in the model. Lines 15-16 in the for loop counts the number of customers visited in the trucks' sequence of customers, lines 18-24 adds the capacity constraints for hazardous and textile waste for the truck, line 26 defines a lambda function for the distance covered by the truck and line 27 updates the list of distances each truck traverses with the distance covered by the truck. Finally, lines 29-32 states that Hexaly should minimize the total distance covered, closes the model and invokes the solver to optimize the problem.

## 4 Experimental Results

In this section the proposed framework is first applied to the synthetic datasets, the resulting solutions are verified, and the framework is compared against the solvers Gurobi, OR-Tools and Hexaly. Finally, an estimate of how long the optimized collections will take to complete is given. All experiments in this section are run on a computer with an AMD Ryzen 7 PRO 7840U 3.3 GHz processor with 16 GB of RAM, running Fedora 41.

### 4.1 Optimization of synthetic datasets

The synthetic datasets are optimized with the proposed framework utilizing PyVRP version 0.10.1. Since the PyVRP solver is non-deterministic, the datasets are each optimized three times with different seeds using the default 900 seconds of runtime or 100.000 iterations without improvement. Given that the capacities of the trucks for textile waste are varying, the lower limits of 60 and 30 bags respectively are assumed. The median results of the three runs of each synthetic dataset are reported in Table 2.

| Collection | n | P | Haz | Tex | Obj. Val. | Time (s) | Total runtime (s) |
|---|---|---|---|---|---|---|---|
| February | 290 | 8 | 267 | 97 | 284.826,2 | 821 | 900 |
| May | 425 | 11 | 378 | 181 | 357.390,0 | 886 | 900 |
| September | 536 | 14 | 493 | 199 | 429.976,6 | 900 | 900 |
| November | 319 | 8 | 282 | 107 | 294.350,6 | 50 | 900 |

Table 2: Median results for each synthetic dataset. $n$ is the number of customers. $P$ is the number of vehicles. Haz and Tex denotes the number of hazardous waste containers and bags of textile waste to be collected respectively. Time (s) is the time spent to achieve the best objective value.

The routes of the median results are superimposed on the plots in Figure 5, Figure 6, Figure 7 and Figure 8 for the February, May, September and November collections respectively. Note that the superimposed routes do not reflect the roads driven but the connection between nodes. Each color represents an individual route. Statistics detailing the optimizations of the median results are plotted with PyVRP in Figure 9, Figure 10, Figure 11 and Figure 12. The top diversity panels show the average diversity of the solutions in the feasible and infeasible populations. The middle objective panels show the best and average solution quality in the given population. The bottom iteration runtime panels show the runtime of each iteration of the entire optimization.

Interestingly, a good solution is very quickly found in the optimization of the November dataset. This is evidenced by the objectives plot as seen in Figure 12. The spikes seen in the diversity and objectives plots in Figure 12 occur when the genetic algorithm in the PyVRP solver restarts. The genetic algorithm restarts after 20.000 iterations without improvement [32], which naturally happens quickly as the solution was found early in the search.

In the optimization of the February dataset (Figure 9) the objective value improves very little after the first 10.000 iterations. The optimizations of the larger May (Figure 10) and September (Figure 11) datasets benefit more from running longer. as they improve noticeably right up until the end. These findings suggest that setting a maximum runtime of 900 seconds is not unreasonable, although it may be excessive for smaller datasets. Conversely, the findings indicate that larger datasets might benefit from even longer runtimes, e.g. 1200 seconds.

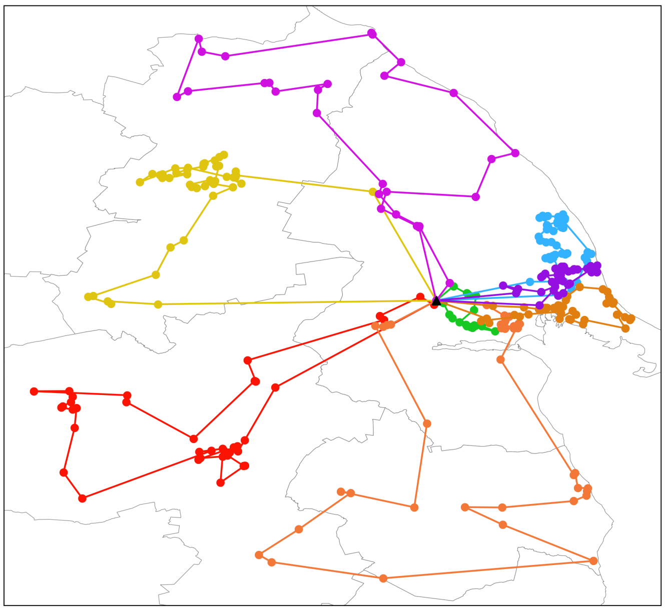
Map data from OpenStreetMap
https://www.openstreetmap.org/copyright

Figure 5: Synthetic Feb. dataset, 290 customers. Routes plotted on map of Eastern Funen.

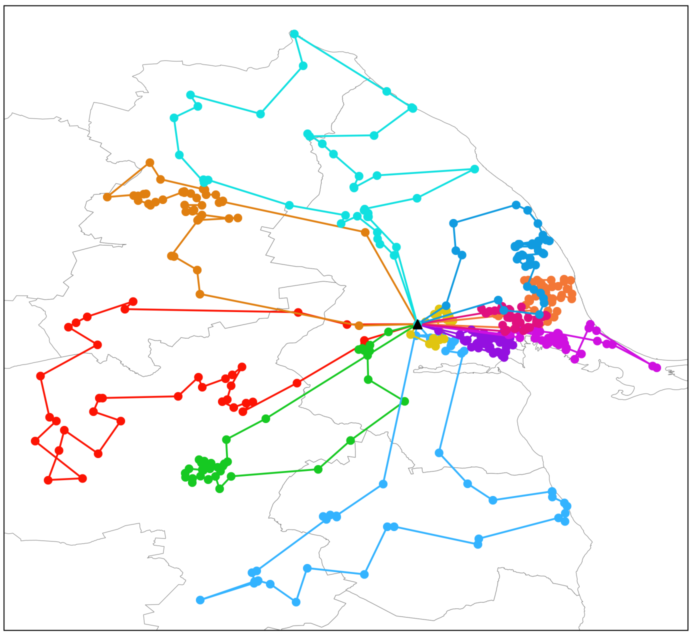
Map data from OpenStreetMap
https://www.openstreetmap.org/copyright

Figure 6: Synthetic May dataset, 425 customers. Routes plotted on map of Eastern Funen.

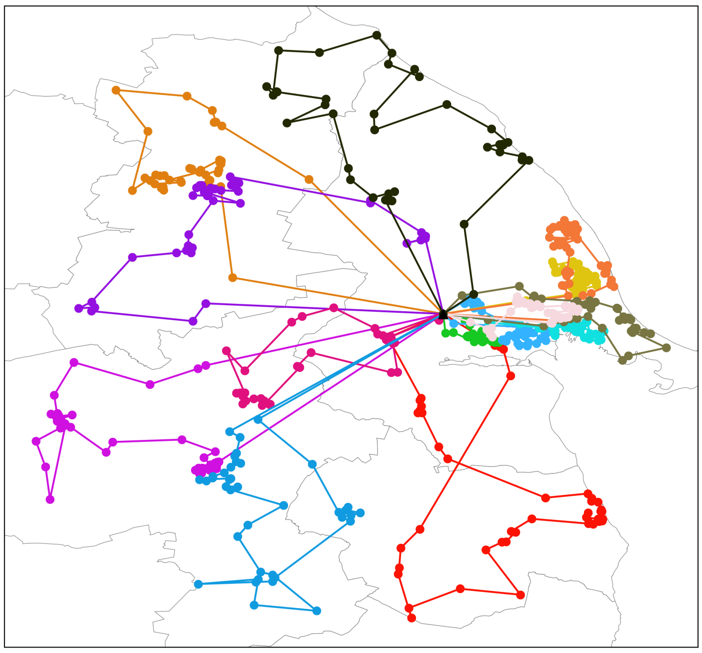
Map data from OpenStreetMap
https://www.openstreetmap.org/copyright

Figure 7: Synthetic Sep. dataset, 536 customers. Routes plotted on map of Eastern Funen.

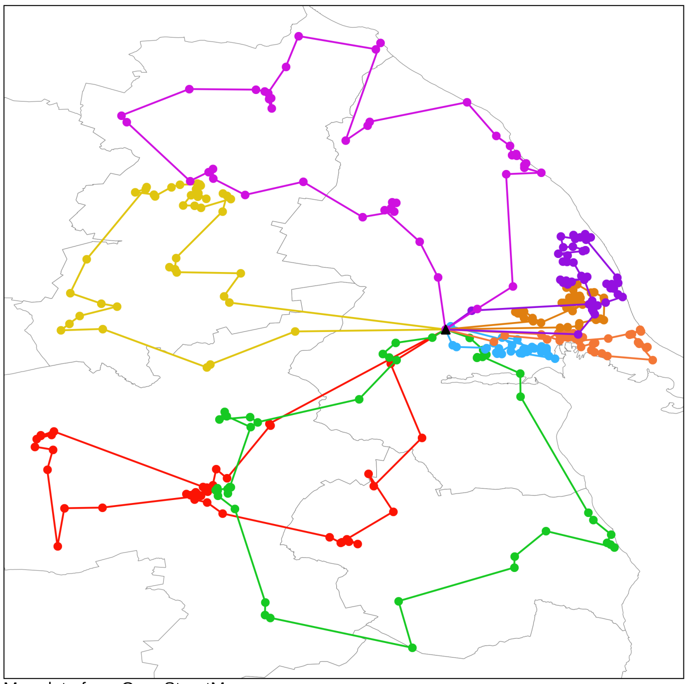
Map data from OpenStreetMap
https://www.openstreetmap.org/copyright

Figure 8: Synthetic Nov. dataset, 319 customers. Routes plotted on map of Eastern Funen.

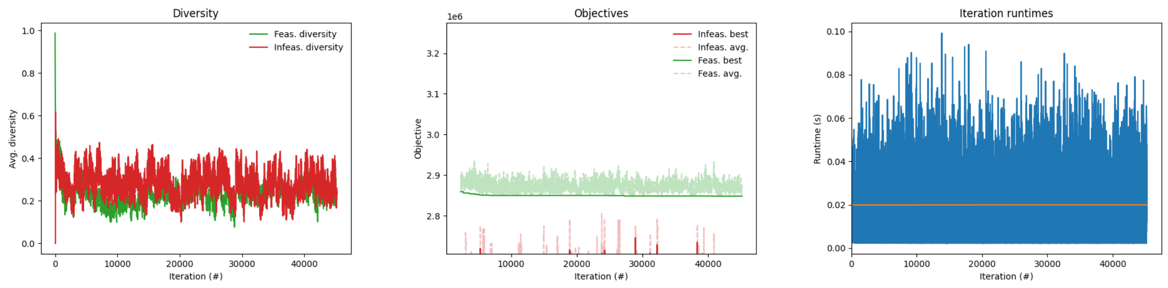

Figure 9: Optimization statistics of diversity, objectives and iterations runtimes for median result of the synthetic Feb. dataset.

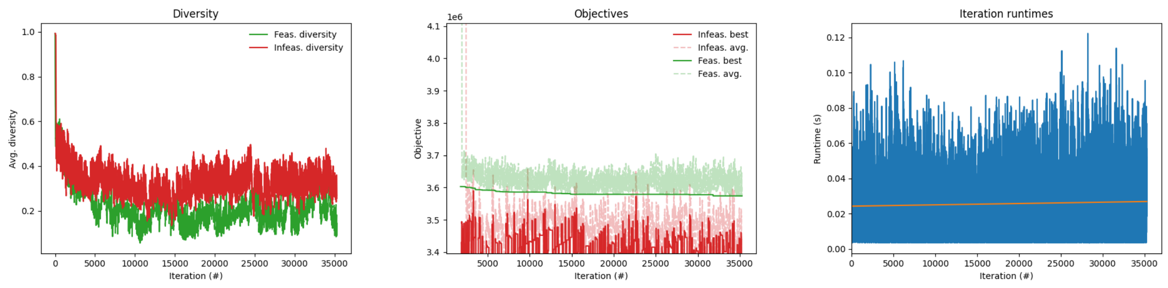

Figure 10: Optimization statistics of diversity, objectives and iterations runtimes for median result of the synthetic May dataset.

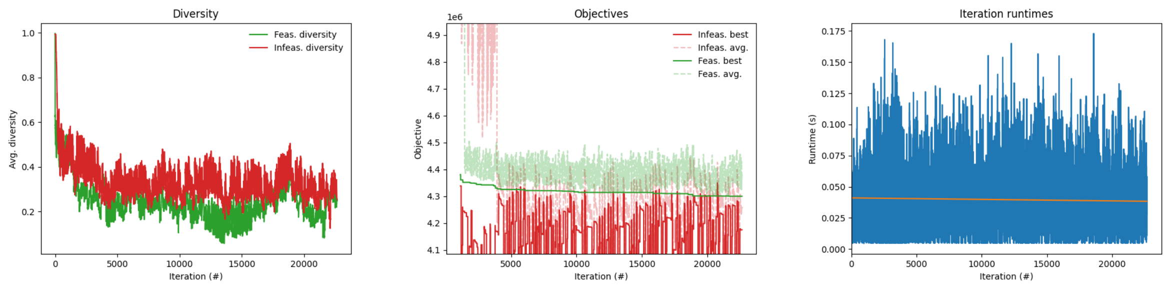

Figure 11: Optimization statistics of diversity, objectives and iterations runtimes for median result of the synthetic Sep. dataset.

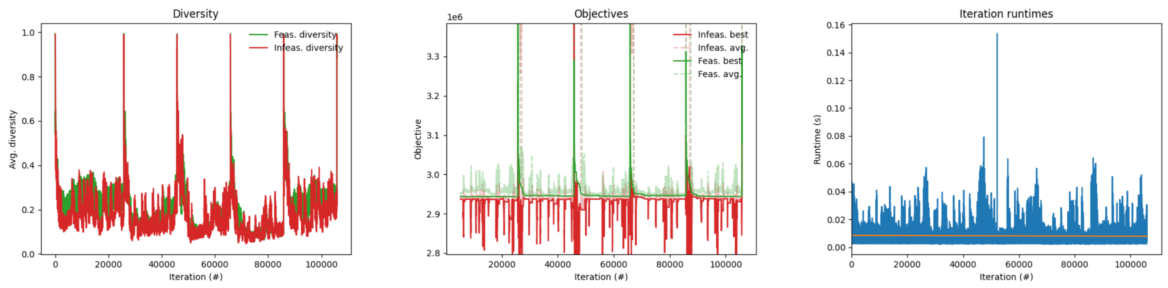

Figure 12: Optimization statistics of diversity, objectives and iterations runtimes for median result of the synthetic Nov. dataset.

### 4.2 Solution verification

The found solutions are verified by thoroughly examining each individual route of a solution. The routes are inspected visually and with a Python script to ensure no subtours are present, that all capacity constraints are respected and that all nodes are visited exactly once.

The verification script works by loading the produced routes, the demand data and the set of available vehicles. The script runs through each individual route keeping track of which nodes have been visited and which nodes have yet to be visited. The capacities are taken into account by summing up the demands for the nodes in each route and verifying that the total demands of the route are less than or equal to the assigned vehicle's capacities.

All solutions are found to be valid solutions satisfying the aforementioned requirements.

### 4.3 Solver comparison

In order to provide a first-hand impression of the differences in performance between various solvers, a comparison is made between the proposed framwork using PyVRP, the MILP model implemented with Gurobi, heuristics-based OR-Tools and the commercial solver Hexaly. Altough it has previously been established that OR-Tools is generally outperformed for VRPs by most other heuristic-based solvers [23], OR-Tools is included in the comparison as it is a very popular and easily accessible solver, whereas most other heuristic based solvers are more challenging to set up. The MILP model implemented in Gurobi represents what can be achieved with a relatively simple and compact MILP formulation solved by a state-of-the-art mathematical optimization-based solver. It is important to note that the results produced by Gurobi only represent the specific MILP formulation used, and that other formulations might produce different results. Finally, the solver Hexaly is included to represent the group of commercial solvers specifically designed for VRPs and other combinatorial problems.

The MILP model is implemented with Gurobi as described in section 3.1 and section 3.2.3. In an effort to improve performance, Gurobi is run with the following non-default settings:

1. MIPFocus is set to 1 to find feasible solutions faster.
2. Heuristics is set to 0,8 to produce more and better feasible solutions.
3. Nodefilestart is set to 0,5 to limit memory usage.
4. SoftMemLimit is set to 15 to exit the optimization gracefully if memory usage is higher than 15 GB.

Additionally, three optimizations are performed with different values for NoRelHeurTime. Using this parameter enables a NoRel heuristic which searches for high-quality feasible solutions. The intended purpose of the NoRel heuristic is to assist in cases where the root relaxation is difficult; however, it is also helpful for combinatorial problems [9] such as VRPs. In this case, the idea is to leverage the NoRel heuristic to achieve a better intial solution and to speed up finding a first inital solution that the solver can improve upon. The values of NoRelHeurTime used are 0%, 15% and 100% of the total runtime to see how the different runtimes of the NoRel heuristic affects the solution quality.

To further assist Gurobi, vehicle sets formed from a single vehicle type are optimized in addition to the standard vehicle sets formed from both vehicle types. This effectively halves the size of the MILP problem. To ensure that the routes produced are still valid, when the company collects the waste using both trucks, the single vehicle type is assumed to be the smallest truck type with capacity for 37 boxes of hazardous waste and 30 bags of textile waste. Using vehicle sets formed from only the smallest truck type is possible based on the routes found with PyVRP. It is observed that more than 30 textile bags are never allocated to any of the trucks on any of the routes. Furthermore, when the ratios of demand to capacity are calculated for vehicle sets formed from only the smallest truck type, it is seen that the number of trucks needed to collect all waste is the same regardless of whether the vehicle sets are formed from both truck types or only the smallest truck type. Note that this does not hold in general. It is also important to note that it might be sub-optimal to form the vehicle sets from only the smallest truck type, as there is no proof that the optimal solutions will have less than 30 bags of textile waste allocated to all routes, but it seems likely based on the routes found with PyVRP.

The ratios of demand to capacity for the vehicle sets, formed from both truck types and a single truck type respectively, are presented in Table 3, showing that the vehicle sets formed from the single truck type are also feasible.

| | | | | Both truck types | | Single truck type | |
|---|---|---|---|---|---|---|---|
| Collection | P | Haz | Tex | $R_{\text{hazardous}}$ | $R_{\text{textile}}$ | $R_{\text{hazardous}}$ | $R_{\text{textile}}$ |
| February | 8 | 267 | 97 | 0,90 | 0,27 | 0,90 | 0,40 |
| May | 11 | 378 | 181 | 0,93 | 0,35 | 0,93 | 0,55 |
| September | 14 | 493 | 199 | 0,95 | 0,32 | 0,95 | 0,47 |
| November | 8 | 282 | 107 | 0,95 | 0,30 | 0,95 | 0,45 |

Table 3: Ratios for the vehicle sets formed from both truck types and the smallest truck type respectively. Haz and Tex denote the number of hazardous waste and textile waste to be collected respectively.

The OR-Tools solver is setup as described in section 3.3. Since it is not possible to set a seed in OR-Tools, and Gurobi is a deterministic solver, these are only run once for each dataset and set of settings. Hexaly is setup as described in section 3.4. Hexaly does support setting a seed and uses randomness in its search. Hexaly is therefore also run three times using the same seeds as PyVRP. Hexaly also appears to convert numerical values to integers, so the distance data is scaled in the same way as for PyVRP. The Hexaly results reported are the median results of the three runs.

All the solvers are set up to use the same vehicle sets, formed by the set formation function described in section 3.2.3, except for Gurobi that also performs optimizations with the vehicle sets formed from only the smallest truck type. The solvers are run with a time limit of 900 seconds yielding the results in Table 4, Table 5 and Figure 13.

| | $NRHT$ = 0% | | $NRHT$ = 15% | | $NRHT$ = 100% | |
|---|---|---|---|---|---|---|
| Collection | Obj. Val. | Lb. | Obj. Val. | Lb. | Obj. Val. | Lb. |
| **Panel A: Vehicle set formed from both truck types** | | | | | | |
| February | - | 196.852,9 | 863.431,4 | 196.845,8 | 359.663,2 | 164.431,6 |
| May | - | 260.288,5 | - | 260.301,8 | - | 213.892,4 |
| September | - | 289.564,1 | - | 289.517,98 | - | 236.127,2 |
| November | - | 211.531,5 | - | 211.544,5 | 672.542,6 | 171.730,5 |
| **Panel B: Vehicle set formed from the smallest truck type** | | | | | | |
| February | 311.443,0 | 198.433,8 | 310.079,1 | 199.511,6 | 295.095,8 | 195.430,3 |
| May | - | 263.528,2 | 1.412.838,4 | 265.002,7 | 551.490,5 | 260.475,2 |
| September | - | 291.542,9 | - | 293.258,7 | 1.952.214,6 | 289.572,3 |
| November | - | 210.665,9 | 359.204,7 | 214.522,1 | 391.081,4 | 210.023,0 |

Table 4: Gurobi optimizations of the MILP formulation using vehicle sets formed from both truck types (panel A) and the smallest truck type (panel B). Results of using 0%, 15% and 100% of the total runtime in the NoRel heuristic, denoted by NoRelHeurTime (NRHT), are reported. Obj. Val. denotes the best objective value found. Lb. denotes the lower bound on the best possible objective. "-" indicates that no solution was found.

The results reported in Table 4 demonstrates that the reduced model size, achieved by forming the vehicle sets from only the smallest vehicle type, yields more and better results compared to forming the vehicle sets from both vehicle types. Additionally, the results indicate that applying the NoRel heuristic for a greater percentage of the total runtime generally leads to better solutions. The best solutions are found when the NoRel heuristic is run for 100% of the runtime, except for the November dataset where the best solution is found with the NoRel heuristic running for 15% of the runtime. However, as seen in the table, using the NoRel heuristic for more of the runtime comes at the cost of worse lower bounds since Gurobi spends more time trying to find good solutions.

| | | | | | | | Gurobi* | | Hexaly | |
|---|---|---|---|---|---|---|---|---|---|---|
| Collection | n | P | Haz | Tex | PyVRP | OR-Tools | Obj. Val. | Lb. | Obj. Val. | Lb. |
| February | 290 | 8 | 267 | 97 | **284.826,2** | 286.497,8 | 295.095,8 | 195.430,3 | 285.202,5 | *283.553* |
| May | 425 | 11 | 378 | 181 | **357.390,0** | 376.308,2 | 551.490,5 | 260.475,2 | 358.393,0 | *351.976* |
| September | 536 | 14 | 493 | 199 | **429.976,6** | 464.695,4 | 1.952.214,6 | 289.572,3 | 431.034,8 | *415.486* |
| November | 319 | 8 | 282 | 107 | **294.350,6** | 305.245,8 | 391.081,4 | 210.023,0 | 295.013,1 | *292.122* |

Table 5: PyVRP and Hexaly results are the median results of three runs. *Gurobi results are the results using a vehicle set formed from the smallest truck type and $NHRT = 100\%$. Obj. Val. and Lb. denotes the best objective value found and the lower bounds respectively for Gurobi and Hexaly. The best objective values for each instance are highlighted in bold, and the best lower bounds are italicized. $n$ is the number of customers. $P$ is the number of vehicles. Haz and Tex denotes the number of hazardous waste containers and bags of textile waste to be collected respectively.

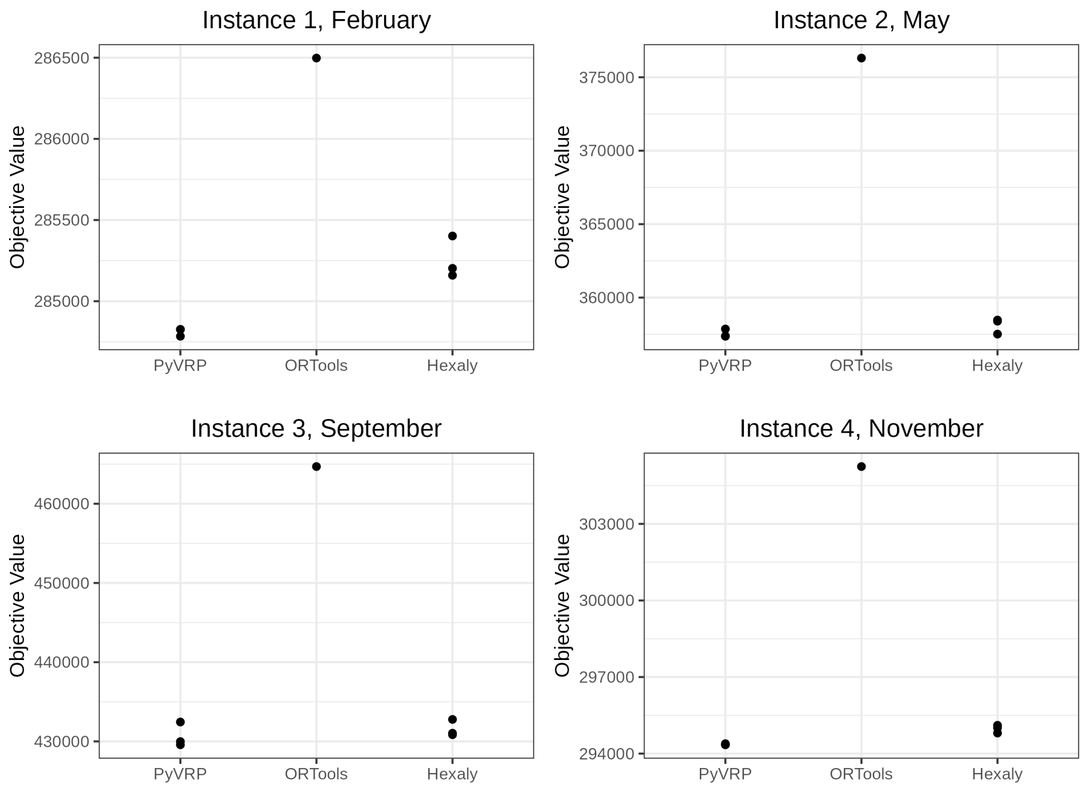

Figure 13: The objective value after 900 seconds of runtime for each run of the algorithms excluding Gurobi.

The results reported in Table 5 and seen in Figure 13 show that PyVRP and Hexaly are superior in solution quality compared to OR-Tools and Gurobi. PyVRP and Hexaly produces very similar objective values with PyVRP delivering marginally better results. OR-Tools yields results that are slightly worse than PyVRP and Hexaly on the smaller February and November datasets. However, the results of OR-Tools are notably worse on the larger May and September datasets. Gurobi is only competitive on the smallest February dataset and performs significantly worse on the other datasets. The lower bounds produced by Hexaly indicates that the PyVRP and Hexaly solutions are very good.

For the solutions of each solver, the percentage-wise difference relative to the lower bounds from Hexaly and the best solutions from PyVRP is calcuated as follows,

$$\text{\%-increase over lb.} = \frac{\text{Obj. Val.} - \text{lb.}}{\text{lb.}}, \quad \text{\%-increase over best sol.} = \frac{\text{Obj. Val.} - \text{Best Obj. Val.}}{\text{Best Obj. Val.}}$$

The percentage-wise differences are reported in Table 6.

| | | | PyVRP | | OR-Tools | | Gurobi | | Hexaly | |
|---|---|---|---|---|---|---|---|---|---|---|
| Collection | Lower Bound | Best Obj. Val. | Lb. | Obj. Val. | Lb. | Obj. Val. | Lb. | Obj. Val. | Lb. | Obj. Val. |
| February | 283.553 | 284.826.2 | 0.45% | 0.00% | 1.04% | 0.59% | 4.07% | 3.61% | 0.58% | 0.13% |
| May | 351.976 | 357.390.0 | 1.54% | 0.00% | 6.91% | 5.29% | 56.68% | 54.31% | 1.82% | 0.28% |
| September | 415.486 | 429.976.6 | 3.49% | 0.00% | 11.84% | 8.07% | 369.86% | 354.03% | 3.74% | 0.25% |
| November | 292.122 | 294.350.6 | 0.76% | 0.00% | 4.49% | 3.70% | 33.88% | 32.86% | 0.99% | 0.23% |

Table 6: The percentage-wise differences over the lower bounds, denoted by Lb., and the best solutions, denoted by Obj. Val. The percentage values show how much larger the found solutions are compared to the lower bounds and best objective values produced by Hexaly and PyVRP respectively.

Lastly, the solvers' progress over time is reported for each dataset in Figure 14 for PyVRP, Hexaly and Gurobi. OR-Tools does not output clear progress information over time and is excluded. The progress is measured by the objective value after 1%, 5%, 10%, 20%, 50%, 75% and 100% of the total runtime. If no progress information is available at the exact measurement point the closest reported value is used. For PyVRP and Hexaly the line plots are based on the median results of the seeded runs. The Gurobi results are the results using a vehicle set formed from the smallest truck type and applying the NoRel heuristic for 100% of the total runtime.

Figure 14 indicates that Hexaly provides higher-quality solutions than PyVRP, in the early stages of the optimization, with the exception of the November dataset where PyVRP provides a better solution from the start. In the optimizations of the February and May datasets, PyVRP has found better solutions than Hexaly after approximately 20% of the total runtime. In the case of the larger September dataset PyVRP finds solutions that are better than those found by Hexaly after approximately 75% of the total runtime. In the November dataset the PyVRP solution quality temporarily degrades when the genetic algorithm is restarted. Gurobi is slow to find solutions and yields very poor solutions at the start of the optimization.

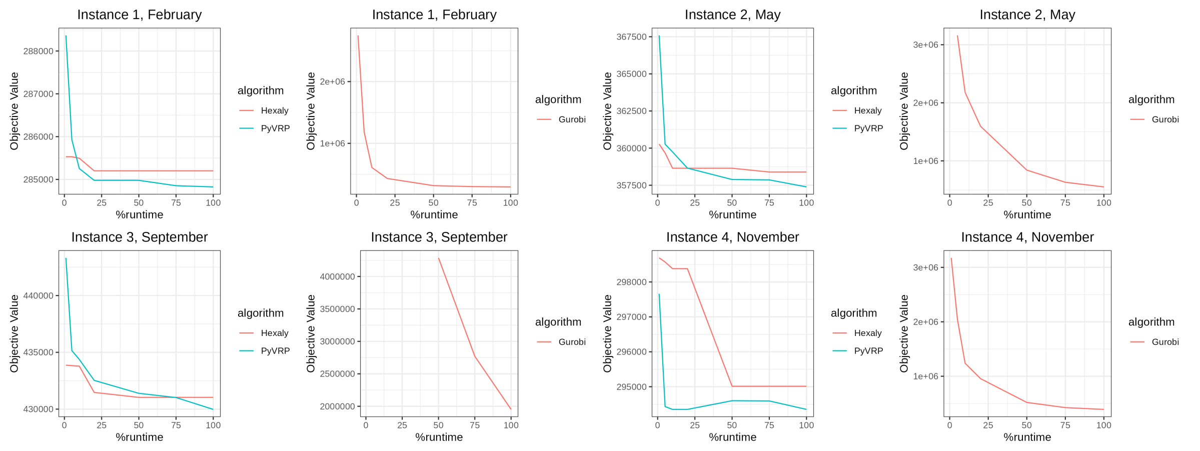

Figure 14: PyVRP, Hexaly and Gurobi progress over time measured by objective value at 1%, 5%, 10%, 20%, 50%, 75% and 100% of the runtime.

## 4.4 Collection Time

This section attempts to provide an estimate of the time required to complete a collection. The estimate is based on the September dataset and the median solution of the PyVRP-based framework. As described in section 1.2 the utility company estimates that they spend approximately 2 minutes at each customer to swap the hazardous waste containers and load the textile waste. Since no data is available about the average driving speed, it is assumed that the trucks are driven at an average speed of $50\frac{km}{h}$.

As an example, the time spent on driving and collecting hazardous and textile waste for the first route is calculated as follows. The first route is 45,1 km long with 42 customers. The time spent for the first route is estimated as,

$$\begin{aligned}\text{Time spent} &= \text{Avg time per customer} \cdot n + \frac{\text{Distance}}{\text{Avg. speed}} \\ &= 2m \cdot 42 + \frac{45{,}0632km}{50\frac{km}{h}} \cdot 60\frac{m}{h} = 138{,}08m\end{aligned}$$

The estimated time to collect all waste from the first route is 138,1 minutes. The routes of the PyVRP solution are summarized in Table 7 with the allocated vehicle types, the number of customers, the amount of hazardous and textile waste to be collected, the distances of the routes and the estimated time to complete the routes computed as above.

| Route | Vehicle Type | n | Haz | Tex | Dist (meters) | Est. time (minutes) |
|---|---|---|---|---|---|---|
| 1 | Large | 42 | 37 | 22 | 45.063,2 | 138,1 |
| 2 | Large | 37 | 34 | 11 | 28.014,0 | 107,6 |
| 3 | Large | 38 | 37 | 12 | 49.221,1 | 135,1 |
| 4 | Large | 38 | 37 | 14 | 21.956,7 | 102,3 |
| 5 | Large | 40 | 37 | 12 | 17.719,5 | 101,3 |
| 6 | Large | 44 | 37 | 15 | 26.458,5 | 119,8 |
| 7 | Large | 39 | 37 | 18 | 53.898,6 | 142,7 |
| 8 | Small | 41 | 37 | 12 | 45.465,9 | 136,6 |
| 9 | Small | 45 | 36 | 16 | 17.755,0 | 111,3 |
| 10 | Small | 37 | 37 | 17 | 16.606,6 | 93,9 |
| 11 | Small | 41 | 37 | 15 | 14.219,5 | 99,1 |
| 12 | Small | 16 | 16 | 8 | 8.199,8 | 41,8 |
| 13 | Small | 38 | 37 | 14 | 43.834,5 | 128,6 |
| 14 | Small | 40 | 37 | 13 | 41.563,7 | 129,9 |

Table 7: Summary of the routes of the median solution produced by PyVRP with the estimated time it takes to complete the routes. $n$ is the number of customers. Haz and Tex denotes the containers of hazardous waste and the number of textile bags to be collected respectively.

Based on the estimated time to complete the routes, the information about the working hours and the loading and unloading of the trucks described in section 1.2, a rough suggestion of how the collection week could be planned is presented in Table 8. It is assumed that the large truck is available for the collection all days from 8:00-15:00 and the small truck from 9:00-16:00. For simplicity it is assumed that the lunch break is held every day at 12:00, and that the trucks need to be prepared every morning.

The time column of the timetables, in Table 8, indicates the start times of the tasks that are to be completed on the given day. The timetables are generous in that there are considerable time gaps around most of the tasks, which provides some flexibility in case of any delays. For example, in the plan for Monday in Timetable A, the tasks at 8:30 are allocated 3,5 hours to be completed; however, Route 3 incl. unloading and preparation for the next trip is estimated to be completed in approximately 2,5 hours, leaving 1 hour until the lunch break.

As seen in Table 8 it is estimated that the large September dataset with 536 customers could be collected in 3,5 days; however, it is important to note that the estimates of time spent on the routes calculated in this section are rather naive and is an obvious point for improvement. As such, this suggested plan should be taken with a grain of salt.

| Time | Monday | Tuesday | Wednesday | Thursday | Friday |
|---|---|---|---|---|---|
| **Timetable A: Routes driven by the large truck** | | | | | |
| 08:00 | Prep (30 m) | Prep (30 m) | Prep (30 m) | Prep (30 m) | |
| 08:30 | Route 3 (135,1 m)<br>Unload/prep (20 m) | Route 1 (138,1 m)<br>Unload/prep (20 m) | Route 7 (142,7 m)<br>Unload/prep (20 m) | Route 4 (102,3 m)<br>Unload/prep (20 m) | |
| 12:00 | Break (25 m) | Break (25 m) | Break (25 m) | Break (25 m) | |
| 12:25 | Route 5 (101,3 m)<br>Unload (20 m) | Route 6 (119,8 m)<br>Unload (20 m) | Route 2 (107,6 m)<br>Unload (20 m) | | |
| 15:00 | | | | | |
| **Timetable B: Routes driven by the small truck** | | | | | |
| 09:00 | Prep (30 m) | Prep (30 m) | Prep (30 m) | | |
| 09:30 | Route 9 (111,3 m)<br>Unload/prep (20 m) | Route 13 (128,6 m)<br>Unload/prep (20 m) | Route 10 (93,9 m)<br>Unload/prep (20 m) | | |
| 12:00 | Break (25 m) | Break (25 m) | Break (25 m) | | |
| 12:25 | Route 14 (129,9 m)<br>Unload (20 m) | Route 11 + 12 (140,9 m)<br>Unload (20 m) | Route 8 (136,6 m)<br>Unload (20 m) | | |
| 16:00 | | | | | |

Table 8: Suggested timetables of how the collection week could be planned for the September collection. Timetable A shows the timetable for the large truck. Timetable B shows the timetable for the small truck.

NFS does not record the exact time spent to complete the collections, but NFS states that it took approximately 4,5 days to complete the September collection in real-life. However, it is important to note that the number of hazardous waste containers collected in real-life exceeded the number of registered containers, since NFS also ended up collecting some containers placed at the roadside that were not registered. With this in mind, a conservative estimate is that NFS could have saved between 0,5 and 1 day to complete the collection using the rough plan generated from the routes produced with the proposed framework using PyVRP.

# 5 Conclusion

In this bachelor project the collection of textile and hazardous waste by Nyborg Forsyning and Service A/S has been examined and modelled successfully as a multi-compartmental vehicle routing problem in MILP terms.

A framework using the open-source state-of-the-art solver PyVRP has been implemented to optimize four synthetically generated datasets based on summary data of real-life collections carried out in Eastern Funen. In this study the open-source solver is shown to perform very well, slightly outperforming a commercial state-of-the-art solver on solution quality across the four datasets. The results are noteworthy, as they indicate that open-source solvers can be a viable alternative to commercial solvers in the context of WCPs similar to the one considered in this project.

It has been demonstrated how the proposed framework can contribute to Nyborg Forsyning & Service A/S reaching their goal, of minimizing the risk of harmful substances being released into the environment, by accelerating the collection process to prevent the hazardous waste containers being opened or removed while they are at the roadside. Further, the proposed framework helps the company achieve their second goal, of reducing unnecessary ressource consumption, by minimizing the distance covered and the routes driven. However, it is important to acknowledge the presence of some potentially inaccurate assumptions such as the time spent at each customer and the average driving speed when estimating how long the collections will take to complete. Consequently, it is recommended for future research to investigate the possibilities of modelling these variables with greater precision using methods from probability theory.

Another intriguing research opportunity is to explore the integration of timetabling into the proposed framework. It would be a natural extension to efficiently schedule the optimized routes in the collection weeks, which would further help the company spend less time. This could for example be done by modelling the timetabling as a bin packing problem.

For Nyborg Forsyning & Service A/S it is recommended that they extend their registration form, where customers sign-up for the collections, with data fields for the number of containers of hazardous waste and textile bags to be collected. Recording the exact amount of waste to be collected at each customer will provide more accurate routes when optimizing future collections, if any customers have more than one container or bag of hazardous waste or textile waste respectively. It is also recommended that the company records more data about their business processes as this will ease any future optimization projects, work and research.

In conclusion, this bachelor project has investigated a waste collection problem, modelled it as a multi-compartmental vehicle routing problem and applied relevant methods to deliver a solution to Nyborg Forsyning & Service A/S that can contribute to the company reaching its goals.

## References

[1] Miljø- og Ligestillingsministeriet, "Bekendtgørelse om affald." [Online]. Available: https://www.retsinformation.dk/eli/lta/2021/2512, 2021, Accessed: Feb. 17, 2025.

[2] Nyborg Forsyning & Service A/S, "Tilmeld afhentning af farligt affald." [Online]. Available: https://www.nfs.as/nyheder/tilmeld-afhentning-af-farligt-affald/, 2025, Accessed: Feb. 17, 2025.

[3] Transportministeriet, "Bekendtgørelse om vejtransport af farligt gods." [Online]. Available: https://www.retsinformation.dk/eli/lta/2017/828, 2017, Accessed: Feb. 17, 2025.

[4] OpenStreetMap contributors, "https://www.openstreetmap.org/copyright. Data is available under the Open Database License." [Online]. Available: https://www.openstreetmap.org/, 2025.

[5] Project OSRM contributors, "Project OSRM." [Online]. Available: https://project-osrm.org/, 2025, Accessed: Mar. 26, 2025.

[6] Overpass turbo contributors, "Overpass turbo, a web based data mining tool for OpenStreetMap." [Online]. Available: https://overpass-turbo.eu/, 2025, Accessed: Mar. 26, 2025.

[7] Geo Fyn, "Geo Fyn - Sammen om data." [Online]. Available: https://www.geofyn.dk/, 2025, Accessed: Apr. 28, 2025.

[8] IBM, "IBM ILOG CPLEX Optimization Studio." [Online]. Available: https://www.ibm.com/products/ilog-cplex-optimization-studio, 2025, Accessed: Apr. 27, 2025.

[9] Gurobi, "The Leader in Decision Intelligence Technology." [Online]. Available: https://www.gurobi.com/, 2025, Accessed: Apr. 27, 2025.

[10] Hexaly, "Hexaly." [Online]. Available: https://www.hexaly.com/, 2025, Accessed: Apr. 27, 2025.

[11] M. Akhtar, M. Hannan, R. Begum, H. Basri, and E. Scavino, "Backtracking search algorithm in CVRP models for efficient solid waste collection and route optimization," *Waste Management*, vol. 61, pp. 117–128, 2017, Accessed: Mar. 6, 2025.

[12] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959, Accessed: Feb. 28, 2025.

[13] A. M. Benjamin and J. E. Beasley, "Metaheuristics for the waste collection vehicle routing problem with time windows, driver rest period and multiple disposal facilities," *Computers & Operations Research*, vol. 37, no. 12, pp. 2270–2280, 2010, Accessed: Mar. 6, 2025.

[14] N. Ouertani, H. Ben-Romdhane, I. Nouaouri, H. Allaoui, and S. Krichen, "On Solving the Hazardous Health-Care Waste Transportation Problem: a Real Case Study," in *2020 International Multi-Conference on: "Organization of Knowledge and Advanced Technologies" (OCTA)*, 2020, pp. 1–5, Accessed: Mar. 3, 2025.

[15] S. Wøhlk and G. Laporte, "A fast heuristic for large-scale capacitated arc routing problems," *The Journal of the Operational Research Society*, vol. 69, no. 12, pp. 1877–1887, 2018, Accessed: Mar. 6, 2025.

[16] E. Babaee Tirkolaee, M. Alinaghian, M. Bakhshi Sasi, and M. M. Seyyed Esfahani, "Solving a robust capacitated arc routing problem using a hybrid simulated annealing algorithm: A waste collection application," *Journal of Industrial Engineering and Management Studies*, vol. 3, no. 1, pp. 61–76, 2016, Accessed: Mar. 7, 2025.

[17] J. K. Lenstra and A. H. G. R. Kan, "Complexity of vehicle routing and scheduling problems," *Networks*, vol. 11, no. 2, pp. 221–227, 1981, Accessed: Mar 7, 2025.

[18] A. G. Soroka, G. V. Mikhelson, A. V. Mescheryakov, and S. V. Gerasimov, "Smart Routes: A System for Development and Comparison of Algorithms for Solving Vehicle Routing Problems with Realistic Constraints," *Automation and remote control*, vol. 85, no. 3, pp. 309–319, 2024, Accessed: Mar. 7, 2025.

[19] Verolog, "Verolog challenge 2019." [Online]. Available: https://verolog2019.ortec.com/, 2019, Accessed: Feb. 28, 2025.

[20] B. Graf, "Adaptive large variable neighborhood search for a multiperiod vehicle and technician routing problem," *Networks*, vol. 76, no. 2, pp. 256–272, 2020, Accessed: Mar. 8, 2025.

[21] DIMACS, "DIMACS :: Implementation Challenge: Vehicle Routing." [Online]. Available: http://dimacs.rutgers.edu/programs/challenge/vrp/, 2022, Accessed: Feb. 28, 2025.

[22] T. Vidal, T. G. Crainic, M. Gendreau, N. Lahrichi, and W. Rei, "A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems," *Operations Research*, vol. 60, no. 3, pp. 611–624, 2012, Accessed: Feb. 28, 2025.

[23] T. Vidal, "Hybrid genetic search for the CVRP: Open-source implementation and SWAP* neighborhood," *Computers & Operations Research*, vol. 140, 2022, Accessed: Feb. 28, 2025.

[24] T. Vidal, "HGS-CVRP: A modern implementation of the Hybrid Genetic Search for the CVRP." [Online]. Available: https://github.com/vidalt/HGS-CVRP, 2022, Accessed: Feb. 28, 2025.

[25] S. Jiang, Z. He, W. Lin, F. Ma, and Z. Lü, "FHCSolver: Fast Hybrid CVRP Solver," Presented

at DIMACS. [Online]. Available: http://dimacs.rutgers.edu/programs/challenge/vrp/papers-videos/, 2022, Accessed: Feb. 28, 2025.

[26] M. R. d. H. Maia, A. Plastino, and U. d. S. Souza, "An improved hybrid genetic search with data mining for the cvrp," *Networks*, vol. 83, no. 4, pp. 692–711, 2024, Accessed: Feb. 28, 2025.

[27] E. Queiroga and R. Sadykov, "POP-HGS: Hybrid Genetic Search as a subsolver in a POPMUSIC algorithm," Presented at DIMACS. [Online]. Available: http://dimacs.rutgers.edu/programs/challenge/vrp/papers-videos/, 2022, Accessed Feb. 28, 2025.

[28] M. Simensen, G. Hasle, and M. Stålhane, "Hybrid Genetic Search With Ruin-and-Recreate," Presented at DIMACS. [Online]. Available: http://dimacs.rutgers.edu/programs/challenge/vrp/papers-videos/, 2022, Accessed: Feb. 28, 2025.

[29] J. Zheng, X. Zheng, Y. Wei, and K. He, "MAESN: Solver Description," Presented at DIMACS. [Online]. Available: http://dimacs.rutgers.edu/programs/challenge/vrp/papers-videos/, 2022, Accessed: Feb. 28, 2025.

[30] S. Voigt, M. Frank, P. Fontaine, and H. Kuhn, "Hybrid Large Neighborhood Search for the Capacitated Vehicle Routing Problem and the Vehicle Routing Problem with Time Windows," Presented at DIMACS. [Online]. Available: http://dimacs.rutgers.edu/programs/challenge/vrp/papers-videos/, 2022, Accessed: Feb. 28, 2025.

[31] W. Kool, J. Olde Juninck, E. Roos, K. Cornelissen, P. Agterberg, J. van Hoorn, and T. Visser, "Hybrid Genetic Search for the Vehicle Routing Problem with Time Windows: a High-Performance Implementation," Presented at DIMACS. [Online]. Available: http://dimacs.rutgers.edu/programs/challenge/vrp/papers-videos/, 2022, Accessed: Feb. 28, 2025.

[32] N. A. Wouda, L. Lan, and W. Kool, "PyVRP: a high-performance VRP solver package," *INFORMS Journal on Computing*, vol. 36, no. 4, pp. 943–955, 2024, Accessed: Feb. 28, 2025.

[33] M. Maia, "MDM-HGS-CVRP." [Online]. Available: https://github.com/marcelorhmaia/MDM-HGS-CVRP, 2024, Accessed: Feb. 28, 2025.

[34] M. Simensen, "HybridCVRP." [Online]. Available: https://github.com/martsime/hybridcvrp, 2024, Accessed: Feb. 28, 2025.

[35] V. R. Máximo, J.-F. Cordeau, and M. C. V. Nascimento, "AILS-II: An Adaptive Iterated Local Search Heuristic for the Large-Scale Capacitated Vehicle Routing Problem," *INFORMS Journal on Computing*, vol. 36, no. 4, pp. 974–986, 2024, Accessed: Mar. 1, 2025.

[36] V. R. Máximo, J.-F. Courdeau, and M. C. V. Nascimento, "AILS-II: An Adaptive Iterated Local Search Heuristic for the Large-scale Capacitated Routing Problem." [Online]. Available: https://github.com/INFORMSJoC/2023.0106, 2024, Accessed: Mar. 1, 2025.

[37] N. Errami, E. Queiroga, R. Sadykov, and E. Uchoa, “VRPSolverEasy: A Python Library for the Exact Solution of a Rich Vehicle Routing Problem,” *INFORMS Journal on Computing*, vol. 36, no. 4, pp. 956–965, 2024, Accessed: Apr. 8, 2025.

[38] J. Coupey, J.-M. Nicod, and C. Varnier, “VROOM v1.14, Vehicle Routing Open-source Optimization Machine,” [Online]. Available: https://github.com/VROOM-Project/vroom, Verso (https://verso-optim.com/), Besançon, France, 2024, Accessed: Apr. 8, 2025.

[39] GraphHopper, “Jsprit,” [Online]. Available: https://github.com/graphhopper/jsprit, 2025, Accessed: Apr. 8, 2025.

[40] G. De Smet and open source contributors, “OptaPlanner,” [Online]. Available: https://www.optaplanner.org, 2025, Accessed: Apr. 8, 2025.

[41] V. Furnon and L. Perron, “OR-Tools Routing Library,” [Online]. Available: https://developers.google.com/optimization/routing/, Google, Accessed: Apr. 8, 2025.

[42] M. Desrochers and G. Laporte, “Improvements and extensions to the Miller-Tucker-Zemlin subtour elimination constraints,” *Operations research letters*, vol. 10, no. 1, pp. 27–36, 1991, Accessed: May. 13, 2025.

[43] I. Kara, G. Laporte, and T. Bektas, “A note on the lifted Miller-Tucker-Zemlin subtour elimination constraints for the capacitated vehicle routing problem,” *European journal of operational research*, vol. 158, no. 3, pp. 793–795, 2004, Accessed: May. 13, 2025.

[44] R. Roberti and P. Toth, “Models and algorithms for the Asymmetric Traveling Salesman Problem: an experimental comparison,” *EURO journal of transportation and logistics*, vol. 1, pp. 113–133, 2012, Accessed: May. 13, 2025.

[45] I. Dunning, S. Mitchell, and M. O’Sullivan, “PuLP: A Linear Programming Toolkit for Python,” [Online]. Available: https://optimization-online.org/2011/09/3178/, 2011, Accessed: May. 19, 2025.