Geographical Information Management and Applications

OPTIMIZING MUNICIPAL SOLID WASTE COLLECTION WITH SENSOR DATA

Student: Grigory Nedaev (Solis-ID 5960681); g.nedaev@students.uu.nl

Supervisors:
Rolf A. de By (ITC, University of Twente)
Parya Pasha Zadeh Monajjemi (ITC, University of Twente)

Responsible Professor: prof. dr. M.J. Kraak

External advisor: Marlex de Jong (GIS Specialisten)

Universiteit Utrecht
TU Delft
UNIVERSITY OF TWENTE.
WAGENINGEN UR

Abstract

This study is about optimizing the municipal solid waste collection: collection scheduling and garbage truck routing. We argue that the use of sensors for monitoring the amount of waste in containers can improve the prediction of accumulation levels and make scheduling more efficient: fewer overflows and fewer unnecessary visits. The benefits that such optimization can bring about are substantial: a reduction in air pollution and traffic and a decrease in operational costs. At the same time, these changes imply that each container will not have a fixed collection frequency anymore but will be collected as late as possible without letting it overflow. Dynamic scheduling will inevitably require dynamic routing: the routes will be defined based on the set of containers chosen for the given date. We will discuss the benefits and the potential drawbacks that these floating schedule and routing may bring about. We approach the problem from a computational and algorithmic perspective and use methods from the fields of combinatorial optimization and operations research to solve the problem. We review some of the exact and heuristic methods and draw our conclusion based on the literature. Finally, we develop and present a minimalistic software kit that consists of an application for receiving and storing sensor data and a QGIS plugin for scheduling and routing.

Contents

Glossary

API	Application Programming Interface
BGT	Basisregistratie Grootschalige Topografie
(C)VRP(P)	(Capacitated) Vehicle Routing Problem (with Profits)
GIS	Geographical Information Systems
GLS	Guided Local Search
FOSS	Free-and-Open-Source Software
IoT	the Internet of Things
LoRaWAN	Long-Range Wide-Area Network
NWB	Nationaal Wegenbestaand
TSP	Traveling Salesman Problem
TTN	The Things Network

1 Introduction

This chapter starts by introducing the research problem's context (1.1), proceeds by defining the project scope and some important constraints (1.2) and mentioning how this work is related to the company called GIS Specialisten and the municipality Almere was chosen as a study area for this project (1.3). Finally, we list the research objectives (1.4) and questions (1.5).

1.1 Problem context

Municipal solid waste management is an integral element of city management and may take up a significant part of the municipal budget. Some studies indicate that the expenses may reach 50% of the municipal budget in the developing countries (Mamun et al., 2015). In the 'developed' countries, this share is presumably lower, but is still considerable. The largest part of these expenses is incurred in waste transportation: approximately 70% on average (Faccio, Persona, & Zanin, 2011; Silva, 2016). Reducing the garbage truck mileage can thus help save the municipal budget and also diminish the environmental impact.

Municipal garbage containers are normally emptied by specialized waste collection vehicles. Traditionally, this is done on a fixed schedule that is based on long-term observations or some statistical estimations (Nuortio, Kytöjoki, Niska, & Bräysy, 2006). For example, using the population count and the average waste disposal rate per person. Most of the time, however, this only gives a rough estimate. Another practice is tasking the collectors with registering the fill level for each container they empty (Lopes, Ramos, & Barbosa-Póvoa, 2015). This manual way of data collection may help improve the accuracy of estimations, but it increases the time the crew spends at each container and the results depend on the collector's ability to visually gauge the amount of waste in it.

Replacing human-made records with sensor data may increase the prediction accuracy of the waste accumulation levels, thus improving the scheduling so that both too frequent and too rare visits could be prevented. Too frequent visit result in unnecessary expenses incurred by operating a vehicle (fuel, wages), and some authors also mention the increase in air pollution as one of the negative effects (Faccio et al., 2011). Too rare visits impact the sanitary conditions: overfull bins may results in waste lying around and contaminating the environment (Likotiko, Nyambo, & Mwangoka, 2017; Nuortio et al., 2006). Some research shows that up to 60% of the containers are emptied prematurely, with the waste collection vehicle travelling extra distance to collect minor amounts of waste (Ramos, de Morais, & Barbosa-Póvoa, 2018b). The use of real-time data about the amount of waste in containers will allow haulers to cut down the costs, reduce pollution caused by vehicle operation and avoid delays in emptying the containers. This logic goes in line with the Smart City approach in which the urban environment is optimized using information technology and the data is collected automatically by specialized sensors (Likotiko et al., 2017).

There are various data communication technologies; among them, Long-Range (LoRa) has gained popularity for its low energy consumption and long range (up to 1 km in an urban environment) that allows to save on building potentially costly infrastructure such as cell towers. Devices that operate on LoRa have a long lifespan: at least one year for the sensor we use in this study, as promised by the manufacturer. This comes at the expense of low data rates, that is, a limited payload that a sensor can send (Centenaro, Vangelista, Zanella, & Zorzi, 2016). Since several bytes, at most, are enough to report on the amount of waste in a waste container, this latter limitation does not, generally, have a strong impact on the problem in this project.

It is easy to imagine a situation in which the haulers come to work in the morning, check which containers are going to overflow on that day and go and empty them all. In reality, however, the availability of such real-time data does not yet allow to discard scheduling. The amount of work must be balanced throughout the week to allow planning the working hours. Planning is also important for choosing the size of the vehicle fleet and the number of employees: drivers and collectors. Thus, in this study we attempt to develop a solution that does not simply decide which containers to empty today, based on their current fill level, but rather tries to look forward and schedule each container for the date after which it is expected to overflow. Once the planning is done, a set of one or more routes, each corresponding to a single vehicle, should be defined through the scheduled containers such that the overall costs of commute are minimized. These costs can be the travel time, the route distance, or a combination of the two. The objective of finding an optimal set of routes through a set of locations for a fleet of vehicles implies that the problem involves spatial data and spatial analysis. More specifically, it involves network analysis for routing through the street network and can thus be put into a broad category of Vehicle Routing Problems (VRP).

1.2 Problem scope and limitations

Waste collection is one part of waste management. It involves the transportation of waste from public waste disposal facilities (waste containers) to landfills or waste processing stations. Further waste treatment and recycling are beyond the scope of this study. We use the term 'waste' to refer to *municipal solid waste (MSW)*, which can also be called *garbage, trash* or *rubbish*. The potential difference between the various MSW categories from the recycling perspective is not investigated. Other waste types such as sewage sludge or agricultural waste are also beyond the scope of this study. There exist different types of waste containers, with no strict regulations as to what to call each type. The terminology also varies between the countries and regions. In this study, we use '(waste) container' as an umbrella term for all of these. The waste containers with underground storage space are sometimes simply referred to as 'underground (waste) containers'.

This study aims to design a solution that could be equally used by the municipalities and the private sector, although because we test our implementation for the municipality of Almere, the results may better address the municipal services. Due to the time and cost restrictions, this implementation is be minimalistic and subject to hardware and software availability. There may be better solutions, and the study does not set a goal of exploring and comparing all possible options. Optimally, the system should be tested by installing a number of sensors in actual waste containers. This, however, is beyond our capabilities, and we use mock data instead. Finally, although this study focuses on using sensors for data collection, it can, with minor adjustments, be used with data coming from any other source. In fact, as stated above, we ourselves test the system without the actual sensor data.

1.3 Organizational context and case study

This study follows in the footsteps of the work done by Grigory Nedaev at GIS Specialisten within the context of one of the company's pilot projects. The project is experimental and investigates the opportunity of using ultrasonic sensors to monitor the fill levels of waste containers and using those records to improve the collection scheduling and routing. The municipality of Almere expressed their interest in this idea at the outset of the project (winter 2018/2019), and we decided to use this opportunity and carry out a case study on Almere. We conducted a preliminary interview on the 29th of January, in which the representatives of Stadsreiniging Almere expressed their interest in sensor-aided waste collection and shared some thoughts about a previous experience that they had with another contractor. According to them, they were not entirely satisfied with the result of that project and were looking for a new contractor, GIS Specialisten being on the candidates. Later, however, Almere put this collaboration on hold while this study was already in progress. We were unable to reach them in March and April having received no response to the emails we sent. Having tried to get in touch through GIS Specialisten, we received a response that Almere needed more time to decide whether they would like to further collaborate with the company. The time it would take them to decide was not announced. At the time of finishing this study, this collaboration is still on hold.

At that point, we decided to keep Almere as the study area but essentially abandon the case study because we had no access to the municipal data or information. For example, we would not be able to compare our results, such as total truck mileage, with theirs. We had to resort to the existing open data sources, such as Basisregistratie Grootschalige Topografie (BGT) for container locations, for example, but we believe that those may be less accurate in some respects and provide fewer details. As a consequence, we would be unable to make the case study realistic. Instead, we decided to focus more on the methods and algorithms for solving the problem and on the development of an example software application that would be able to receive data from sensor and do scheduling and routing.

We keep this study related to the company's project but independent in the sense that it is not driven by the company's objectives. We use some of the company's hardware: namely, a sensor and a gateway. GIS Specialisten provided us with one ultrasonic LoRa sensor, one The Things Network (TTN) gateway and a corporate TTN account. The abovementioned is agreed and guaranteed by the authors and the external advisor at the company, Marlex de Jong.

1.4 Research Objectives

The *first objective* of this study is to design an algorithmic solution for scheduling the emptying of waste containers based on automatically transmitted sensor data about their fill level, and on finding an optimal set of routes for a fleet of waste collection vehicles. The optimality of a solution is defined by the time a fleet of garbage trucks spends underway to meet its waste collection obligations (let no containers overflow). To meet this objective, we formulate a mathematical optimization problem and review the existing exact methods and heuristics for solving it.

The *second objective* is to build a software prototype that would implement this algorithm: receive and process sensor data, perform scheduling and routing, and visualize the results.

1.5 Research Questions

To help meet the first objective, the following research questions are formulated:

- What is the mathematical programming formulation of the problem?
- How can the operational costs be calculated?
- Which exact methods can be used to solve it and is it possible to use these methods in the given use case considering the size of the problem (number of containers)?
- Which heuristics can be used instead? To what extent do applied heuristics affect optimality of found solutions?

The second objective brings up the following questions:

- What technology can be used to implement and solve an optimization problem of the kind defined in the project?
- If there are several components required, how can they be integrated to work with real-time data input?

2 Related research

Despite the emphasis on using sensors, a certain part of the underlying solution does not depend on how the data is obtained. What matters is how this input is processed. This makes the studies that do not use sensors but still solve the vehicle routing problem in application to waste collection no less valuable for our project. Section 2.2 presents both variations, starting from the older work and proceeding chronologically. Section 2.3. finalizes this chapter with a review of the studies that focus on the hardware rather than on the route optimization. First, however, we briefly review the class of problems that this project aims to solve (2.1).

2.1 Vehicle routing problems

Garbage truck routing in municipal solid waste collection is commonly solved as a Vehicle Routing Problem (VRP) (Archetti & Speranza, 2014; Faccio et al., 2011; Lopes et al., 2015; Nuortio et al., 2006; Ramos et al., 2018b; Silva, 2016). A VRP asks "What is the optimal set of routes for a fleet of vehicles to traverse in order to visit a given set of locations?". It is a generalization of the well-known Travelling Salesman Problem (TSP) to the case of multiple vehicles (Golden & Wasil, 2008). Splitting the set of locations in several parts and solving the TSP for each set independently is not, generally, the same as solving a VRP: optimality of each TSP solution cannot guarantee the optimality of their combination. This makes solving a VRP a more complex task than solving a TSP, and since the TSP is NP-hard, the VRP, as its generalization, is at least as hard (Archetti, Speranza, & Vigo, 2014).

There are many variants of the Vehicle Routing Problem, and some authors use more specific terms such as "Capacitated Vehicle Routing Problem" (CVRP) when the constraints on the vehicle capacity are introduced, or "Vehicle Routing Problem with Time Windows" when the vehicles can only visit the locations within certain time periods. For a comprehensive survey of Vehicle Routing problems, we refer the reader to Golden, Raghavan & Wasil (2008). The diversity of the VRP variants suggests that VRP should be regarded as a *class of problems* that share the goal of finding an optimal combination of routes through a set of locations. The optimality of such a set of routes is most commonly measured in terms of their overall length, travel duration or a combination of the two.

It is important, however, to draw a line between the cases where the set of visit locations is a given, and the ones where it must first be subsetted from a larger set. The latter class of problems is commonly referred to as "Vehicle Routing Problems with Profits", or VRPP. For a comprehensive survey on this specific variety, see Archetti, Speranza & Vigo (2014). Other possible terms include "Stochastic Vehicle Routing Problems" (Nuortio et al., 2006) and the "Team Orienteering Problem" (C. Archetti, Feillet, Hertz, & Speranza, 2009; Archetti, Hertz, & Speranza, 2007). In these problems, multiple possible partitions of the location set must be compared to identify the most "profitable" one. A number of studies on smart waste collection report better results using the "with profits" approach (Nuortio et al., 2006; Ramos et al., 2018b; Silva, 2016). The solutions they present do not simply choose all waste containers that exceed a certain fill level threshold, but are capable, for example, of observing that it is cheaper to collect waste from a half-empty container today if it is located close to the full containers that are being visited, rather than drive to visit that single container when it fills up.

The notions of profit and cost are abstract and should not always be taken literally. The VRPs are commonly solved using network analysis and graph theory. Profits are associated with visit locations, but depending on the solution, can be assigned to the road segments that those locations belong to. Profits may denote the actual monetary gains from visiting a location, e.g. in case of food delivery, but are not restricted to such semantics. In the case of waste collection, profits generally refer to the amounts of waste collected from waste containers, although given waste selling prices, the amounts may be converted into a monetary value (Ramos, de Morais, & Barbosa-Póvoa, 2018a). Costs are generally incurred by traversing the edges (also called arcs), i.e. road segments, and are assigned to those. However, nodes (also called vertices) can also be assigned costs: in waste collection, that could be the time spent emptying the given container. Costs assigned to arcs often represent the length of the road segment or the time required to traverse it. Given a cost function, these values can be converted to the monetary values.

Vehicle Routing Problems are NP-hard which means that the time it takes to solve it to optimality (that is, using an *exact method)* rises exponentially with the number of destinations, and these time cost may not be feasible for the given use case. Therefore, heuristics and *metaheuristics* are commonly used instead. The past decades saw a growing amount of research on metaheuristics, - algorithms that control the behavior of one or more heuristics to solve the given problem within a reasonable time. They are not guaranteed to return an optimal solution, but given enough time, they are expected to find a solution sufficiently close to the global optimum. Golden, Raghavan & Wasil (2008) and Archetti, Hertz & Speranza (2007) present an overview of metaheuristics for VRPs.

2.2 VRP in municipal waste collection

Nuortio et al. (2006) present one of the most comprehensive works on the topic and describe a case study about two municipalities in Eastern Finland. They do not use sensors, but instead approximate the accumulation rates using a combination of historical records and statistical data. This is useful in the situation where the system must be designed and tested before the deployment of sensors is possible. The authors use the fact that a truck is weighed each time it arrives at the waste disposal to estimate the amount of waste collected from each container. The mean average does not necessarily represent a realistic value, however, so Nuortio et al. complement this data with statistical indicators, such as population density and GDP per capita for the given neighborhood. In addition, they use regression analysis to study temporal trends: seasonal, monthly and other variations in waste disposal rate. Besides, they calculate the travel time between the containers using historical data which means that they collect their own traffic data.

The conceptual design is highly detailed, kept as close to reality as possible, imposing a number of restrictions. The truck fleet consists of identical trucks with a capacity of 26 tons each. The different types of waste are collected separately, and each vehicle can only collect one type during a single route. There are several different types of containers, of which underground containers must be collected separately. Collection can only be done on workdays and within the following time windows: between 6 AM and 10 PM. The disposal site, where the trucks deliver the collected waste to, is open from 6 AM on Monday to 7 PM on Friday. The 8-hour working day includes a 30-minute lunch break and two 15-minute coffee breaks, and can be exceeded, in which case an overtime pay is incurred. The authors also recognize the need to allocate containers to scheduled dates prior to solving the VRP itself, and hence call the problem a *Periodic Vehicle Routing Problem*, or an *allocation-routing* VRP. They present a software implementation that calculates optimal routes based on any of the three objectives: distance, duration and a combination of the two.

The authors run tests to compare the results with the then-current situation. Importantly, Nuortio et al. do not attempt to optimize the number of required routes but rather minimize their combined length. In doing so, they argue that the number of routes corresponds to the number of vehicles available, and in reality, the haulers would not like to change the fleet size or let some of the trucks stay unused; instead, they would like to know how to spread the workload between them more evenly and potentially reduce the average travel time of each truck. They first test whether the order of locations in the existing routes can be optimized by solving the Travelling Salesman Problem (TSP) on each single route. At this stage they already report an improvement of on average 12% per route in terms of distance. Next, they run the complete algorithm that includes schedule optimization. This time they report a 46% improvement from the current situation. The authors mention, however, that the imperfections in the road dataset and the approximations made in estimating the baseline data must be accounted for.

Faccio, Persona & Zanin (2011) present a use case of municipal solid waste collection in an Italian city with 100,000 inhabitants. As the paper's title suggests, they develop their model to be used with sensor data and build it in such way that it can be optimized for several different parameters: time spent, distance travelled, and the number of vehicles used. Having no opportunity to deploy actual sensors, they test their hypotheses on historical data provided by the municipality. They do not, however, provide details about the contents and accuracy of this data. The authors present a detailed description of the hardware setup: containers are equipped with volumetric sensors, GPRS transceivers and electronic tags that uniquely identify each container (RFID tags). The vehicles are equipped with GPS modules to track their location in real time and electronic devices that read the RFID tags installed in the containers. The containers hourly report on their fill level. Besides, every time a vehicle empties a container, it reads and reports its unique ID to the system. Following either of these two events, the collection routes are regenerated, i.e. the routes are not only generated before the dispatch but are also recalculated at collection time when any of the containers report a change in fill level. This continuous re-routing distinguishes this study from many others although it is not clear how convenient it is for the drivers. Furthermore, it also means that the model does not involve any scheduling, but rather operates "on-the-fly" and is limited to the present day.

The authors present several variants of the solution, one for every objective. All of them, however, are deterministic in the sense that the selection of the containers is based on a fixed threshold. None of them allows scheduling. Besides, it is not entirely clear how the authors find an optimal combination of routes. Despite discussing several heuristics, they seem to solve the problem by constructing the routes sequentially using the Nearest Neighbour method, i.e. a greedy algorithm. If so, this suggests that the solution is not likely to be optimal. Strong points of this project are a feasibility study and a cost-benefit analysis. The authors present the costs of the vehicles and devices, fuel and maintenance expenditures, wages, etc., and do so for each scenario. According to the figures presented, it will take between three and five years until the costs of sensor deployment are earned back.

Likotiko, Nyambo & Mwangoka (2017) attempt to solve this problem via agent-based modelling using NetLogo. Containers are equipped with sensors, but the authors do not say of what type those are. At a certain interval, sensors report on the fill levels that get stored in a database. At the start of each day, the application selects all containers with fill level above a certain threshold and solves a TSP on them. The authors claim to use the "Dijkstra's algorithm as implemented in NetLogo", and although that refers to a well-known shortest path finding algorithm, it is not exactly the same as solving the TSP. Altogether this study seems to present a relatively simple deterministic model with no scheduling.

Ramos and her colleagues at Lisbon University have published a series of papers on waste collection optimization (Lopes et al., 2015; Ramos et al., 2018b, 2018a; Ramos, Gomes, & Barbosa-Póvoa, 2014). In their research, they often refer to the papers that we have described above. Two of their works (Ramos et al., 2018b, 2018a) describe a case study about a local waste collection company: there is a homogeneous vehicle fleet and a homogeneous container set. The vehicle capacity, fleet size and the maximum tour length are the key constraints in the model. As Nuortio et al., Ramos and co-authors also use the company's actual data both as an input for the model and as a baseline to evaluate the results. The haulers provided them with the actual fill level records for each container: upon emptying a container, the collectors record the observed fill levels on an ordinal scale "<25%", "25-50%", "50-75%", "75-100%".

The authors set an objective of maximizing the amount of waste collected per unit distance travelled. They solve it as a node routing problem but do not use the road network. Instead, they use Euclidian distance between the containers; they multiply this distance by a constant (1.53), as they believe it represents the average ratio between the road and the straight-line distances in their study area. They do not, however, describe how they arrived at this figure. From our perspective, such modelling might oversimplify the solution and impact the results, so in our study we use an actual road network instead.

Ramos et al. formulate three different approaches to sensor-enabled waste collection. In *the first approach*, only the containers whose fill level exceed a certain value are emptied. This selection procedure is run every morning before dispatching the vehicles, and only covers the current day. This significantly reduces the implementation complexity but makes it somewhat short-sighted since the accumulation rates are likely to vary among the containers and the chosen threshold may not be suitable for all of them. Besides, there is no planning for the rest of the week or beyond.

The second approach improves on the first one by calculating a projected fill level for each container using its accumulation rate. The model allows to choose the lowest admissible service level, so a certain number of containers are allowed to overflow up to a certain point if such scenario yields better results. The solution may be optimal for the given day as it guarantees the required service level, but the nature of municipal solid waste collection may still require a longer-term schedule that fixes the collection dates for the coming week, month or a similar calendar period (at least long enough to define the working hours and the required number of employees and vehicles).

In *the third approach*, the model also takes the lowest allowed service level, and finds the number of containers that are allowed to overflow. Now, for each container, the closest overflow date is calculated, and if on a given day the number of such containers exceeds the allowed value, the model schedules the container's emptying for that date. Once all the containers have been scheduled, the algorithm solves the VRP for each of the days, in this way planning the containers and the route ahead for the given time horizon (Ramos et al. use 30 days as such).

While the first two approaches yielded worse results than the company's baseline indicators, the third approach showed an improvement in the distance travelled per waste collected ratio. Lowering the service level to 99% resulted in even larger savings, a compromise that may or may not be acceptable in a real situation. Overall, the authors argue that their results (specifically, the third approach) prove the benefits of using fill level sensors, although we suggest this method should first be enhanced by using a real road network as the graph base. Finally, the authors do not provide the formula that they used to arrive at a constant factor of 1.53.

In one of the most recent studies on the topic, Ferrer & Alba (2018) present a use case of Algeciras (Spain). They develop a model similar to ours and set the objective of minimizing the distance travelled by the fleet. They also calculate waste accumulation rates for each container based on the previous measurements by the sensor. One merit of their model is that it can consider a heterogeneous fleet and a heterogeneous set of containers.

The software setup is divided in four modules: Data Management, Intelligent Decision, Routes Generation and Visualization. The data management module is a database in which all containers with their locations and attributes are stored, as well as their historical fill level records and a cost matrix specifying the commute cost between each pair of containers, in distance and in time. They use Google Maps API to generate the cost matrix, one with distances (constant over time) and one with travel times (computed anew every time, using Google's traffic data). Finally, they use the same database to store the solutions. The intelligent decision module is a set of algorithms from the field of machine learning, as authors describe them, which use historical data to make future predictions. They do not specify which exact method they use, however, unless they simply use an average of historical accumulation rates. The routing module is built around what they call an "evolutionary algorithm (1+1)". The solution can generate routes for different dates, thus facilitating scheduling. Finally, the visualization module is a web page that provides a simple interface for requesting routes and displaying them on a map. At the time of writing this section, it is available online at http://mallba3.lcc.ua.es/binct/.

The authors run two experiments. In the first one, they predict future fill levels for a number of containers and later compare their prediction with factual observations. They use 217 communal paper waste containers in the city and an eleven-month archive of historical fill levels. In this case, the historical data is the records made by waste collectors on the spot when emptying the container (the authors do not specify how the collectors do it). The authors derive the accumulation rate function using three methods: linear regression, Gaussian process and support vector machine for regression (SMOreg). They repeat the test 30 times, and the results show that the mean absolute error is the lowest when using Gaussian process (3.83% compared to 7.41% for linear regression and 9.52% for SMOreg).

In the second experiment, they predict the fill levels for a future date, generate routes and then when the date arrives, compare their results with the hauler's route in terms of distance and duration. In their calculation, they tune the algorithm to mark all containers that exceed the 80% fill level as mandatory for emptying and the ones between 50% and 80% as optional. The algorithm picks 77 containers, which is more than the haulers schedule for that day (64) and divides them between two vehicles. The results show the overall distance and time for each of the routes and the average of both values per container visited. Based on these figures, the authors conclude that their solution reduces the overall mileage by 20% (even though more containers are visited), and the mileage per unit waste collected by 33%. These results are obtained using 10,000 iterations of the intelligent selection algorithm. The authors do not comment, however, on the optimality of this solution in the long run, i.e. in combination with other dates. The reported results may save costs on the given day, but whether they increase the results over a longer time period, remains unclear. Overall, however, this project presents an illustrative study with a high level of detail about the algorithm used (Ferrer & Alba, 2018).

2.3 Research on using sensors in waste collection

Our study focuses on the data processing rather than on the hardware used to acquire the data. We still do, however, discuss the hardware part, hence we present here a brief review of the literature devoted more specifically to that side of the problem.

Mamun, Hannan, Hussain & Basri (2015) develop and test a prototype in which they use a combination of an ultrasonic sensor, an accelerometer, a magnetic proximity sensor and a microprocessor to monitor the state of a container. Their results show that in their case sensors reduce the waste collection expenses by 10–20% because of a 26% lower fleet mileage. They do not, however, describe the routing algorithm they use and focus mostly on the hardware details: the sensors they use, e.g. each module's energy consumption, price, etc., which allows to accurately estimate the costs of deploying sensors. They install sensors in a few containers and run a series of performance tests. Among the results are the 5 – 10% error rate for the fill level measurements. To increase the battery life, they set the fill level sensor in sleep mode by default. When the lid where the accelerometer is installed is opened or closed, or the container is tipped over during emptying, the combination of the acceleration dynamics and the status of the lid (open/closed) reported by the proximity sensor define the status of the container (full / not full). Additionally, they use install weight sensors under the container. Finally, the authors present some valuable information on the costs of solid waste collection in developing countries. According to their information, waste management takes up to 50% of municipal budget, and about 85% of these costs are incurred in waste collection and transportation.

Rovetta et al. (2009) describe a setup tested in Pudong, China: 240-liter wheeled bins were equipped with two sets of devices: one on the inside of the lid and the other one on the bottom. The lid set included an optical camera, four LED lights, an ultrasonic sensor, an electronic thermometer and a humidity sensor. The LED lights illuminate the inside of the container when the camera takes snapshots. The authors used a combination of image-processing with the ultrasonic ranging to determine the fill level. Unfortunately, the authors do not elaborate on the method they use. Although other papers also mention how cameras can help validate and improve the data obtained by ultrasonic ranging (see Mamun, 2015), the combination of the two may be an expensive solution. Now, the bottom set was comprised of a pressure sensor to estimate the content's weight and an electronic thermometer. Both sets of devices were connected to a processing unit and a GPRS modem for data transmission. An interesting fact is that the authors use the weight and fill level measurements to calculate the density of the bin's contents with the aim of defining the type of waste inside the bin. We have not come across this idea in the other studies we have reviewed, and although the goal itself is generally understood, the method suggests that the authors arrive at the average density, and if the contents are heterogeneous, this might provide little help in defining the constituting parts. Finally, the authors use an optimization engine to schedule and define routes via VRP, but do not detail the implementation. The present the value for the computed routes, such as distance and duration, but do not compare them with the actual data, apparently due to unavailability, so they rather illustrate that the system is generally capable of defining routes and scheduling pick-up dates (Rovetta et al., 2009).

Hannan, Arebey, Begum & Basri (2011) describe a system very similar to the one developed by Mamun et al. (2015) but use optical cameras to take images of the container's inside before it is emptied by the vehicle. They use an image of an empty container as a default and compare it with pre-emptying images by subtracting the pixel values of the two images (converted to their greyscale versions). In their case, the container's walls are generally represented by the darker shades while the waste shows up as lighter shades. By using some decision algorithm, the details of which they do not provide, a binary image is derived in which all pixels have values either 0 (no waste) or 1 (waste). The ratio of these two categories allows the authors to estimate the amount of waste in the container. Additionally, the authors use the pixel intensity in the grayscale images to estimate the amount of waste in the vertical direction. Hannan et al. mention that this method was developed empirically by running tests with different amounts of waste in the container and comparing the pixel value histograms of the obtained images (Hannan, Arebey, Begum, & Basri, 2011).

There have been many more studies on the use of sensor in municipal solid waste collection, but it is beyond our capabilities to present them all here. In general, it appears that ultrasonic rangers are used more often than cameras since image analysis techniques are generally more complex than deriving the fill level from the measurements obtained by ultrasonic sensors. Furthermore, ultrasonic rangers send their records in a few bytes whereas each image may take up several megabytes, and it means more storage is required and processing may take longer time. Ultrasonic sensors do, however, one important vulnerability: the waste inside the container must be spread evenly.

3 Methodology

We use the municipality Almere as a study area. Our choice fell on Almere because they collaborated with GIS Specialisten in the pilot project described in section 1.3. Almere is a mid-to-large-sized municipality with just over 200,000 inhabitants in which communal containers with underground storage space are used for waste collection at apartment blocks. An interview with the representatives of Stadsreiniging Almere (see appendix A) showed that they were supportive of the idea of using sensors to optimize the way they perform waste collection and transportation. Almere already had a similar project before (2017 – 2018, carried out by VConsyst). The experience was not entirely positive, and Stadsreiniging was looking for another contractor (D. Puzic, G. Kalkhoven, G. Chandler, personal communication, 29th of January 2019).

3.1 Sensors

Ultrasonic rangers are a type of sensors commonly used for waste level monitoring. The one at our disposal is a "Smart Waste Bin Detector DF702 LoRaWAN" (fig.1) produced by CNDingtek (Beijing, China).

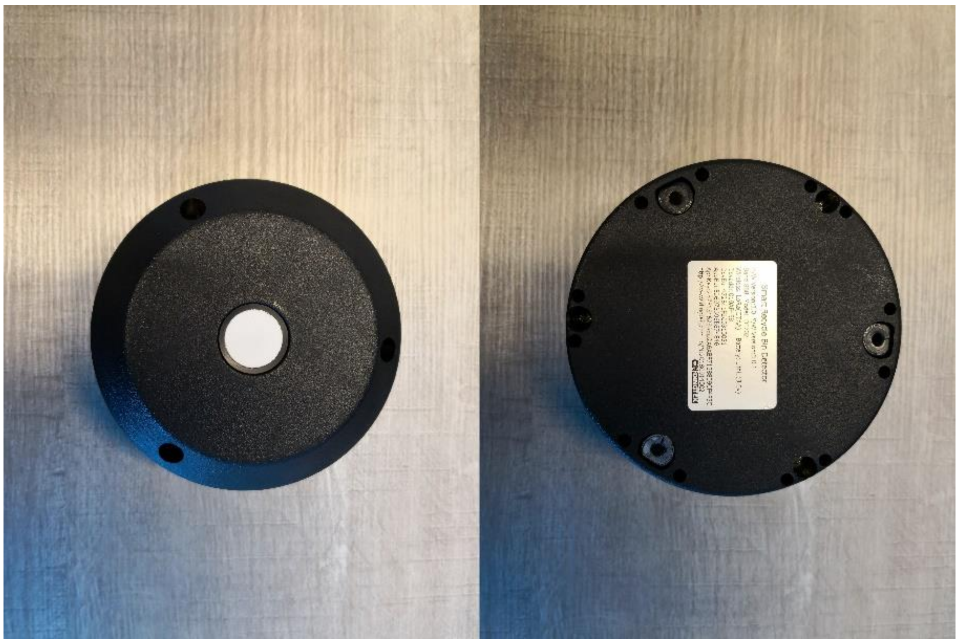


Figure 1. The sensor used in this study.

It is housed in a protective case and mounted on the inside of the container's lid. From there it sends a downward signal at equal time intervals. The signal bounces back off either the garbage or the container bottom and comes back to the sensor. The sensor built-on chop converts the signal's travel time into the distance between the current waste level and the top of the container (fig.2). This, and another binary value that says whether the container is full (threshold set to 90%), are then sent to whatever the sensor is connected to, in our case The Things Network.

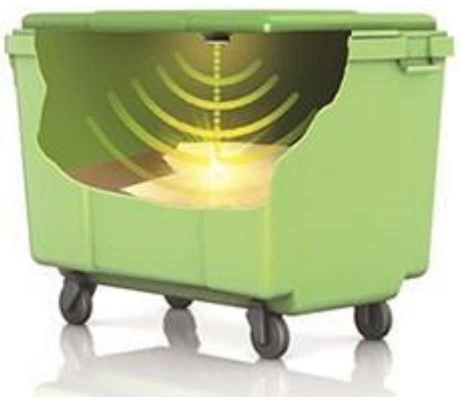


Figure 2. The work principle of the sensor.

The sensor is also supplied with a thermometer that registers whether the container has caught fire (it sends 0 (false) or 1 for 'true'). Additionally, it measures the orientation of the container. Should it tilt, fall on the side or turn over, the sensor will send another '1' value in its payload, otherwise '0'. Finally, if the battery is running out, it will also send a binary value in a separate bit. In this case, however, it will not send the rest of the values in order to save the battery power. All measurements are taken concurrently (CNDigitek, 2019). In this study, however, we only use the fill level value, leaving the rest of functionality for the future work. The sensor is wired to work on LoRa, which operates at 868Mhz in the EU.

Ultrasonic rangers do have certain vulnerabilities. They will give erroneous records if the waste inside the container is spread unevenly or any other obstacles come in the signal's way. For example, the problem in Almere was the bulging of the trash bag that lines the container from the inside. With the parts of the bag getting in the sensor's line of sight, the recorded fill levels were often exaggerated. Ultrasonic rangers are not the only available type of sensors. Mamun et al. (2015) and Faccio et al. (2011) both review the use of different sensors for waste monitoring. Common alternatives are optical or infrared cameras that produce images that can be processed by feature-recognition algorithms to determine the fill level. In section 2.3, we have reviewed some studies that use other sensor or compare the sensors between each other.

3.2 Research problem model

The objective of finding an optimal set of routes for a fleet of vehicles to visit a given set of locations defines a Vehicle Routing Problem (VRP), a generalization of the Travelling Salesman Problem (TSP) to the case of multiple vehicles. Since in our case the set of locations to visit on any given day must first be selected from the superset of all registered containers based on fill level, the problem turns into a VRP with Profits (VRPP). Thus, there are two major steps: selection and routing. Fig. 3 presents an overview of these steps.

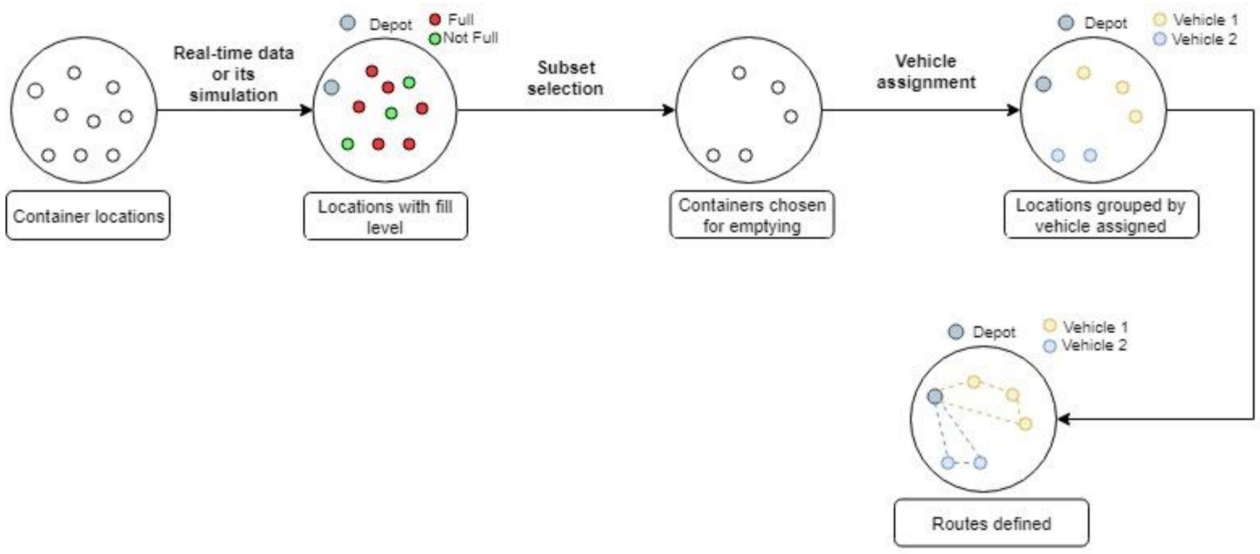


Figure 3. Algorithm workflow overview.

There is a homogeneous fleet of garbage trucks and a heterogeneous set of waste containers that are served by the fleet. There may be several types of containers, each with its own capacity, height and service time (the time it takes to empty it). Each container is equipped with an ultrasonic ranger that reports on the container's fill level at equal interval (in this study, we use one time a day). The objective is to minimize the amount of time the fleet spends underway, aggregated over the dates for which the containers have been scheduled (time horizon). No containers are allowed to overflow (100% service level). Time series of fill level records obtained from sensors are then differences, i.e. the daily increases are computed, and these are then averaged over a certain period of time. In this study, we do not set a fixed limit on age of the records that can be used to calculate the predicted accumulation levels because we believe that it strongly depends on the use case. The scheduling and route planning algorithms are designed to be run once for each time horizon: for each container it finds the date when it is expected to overflow, based on recorded history. If service level is set 100% (default), the algorithm schedules the container's emptying for the day prior to becoming full. Additionally, it ensures that it is a working day.

3.3 Cost function

The optimality of a route is most commonly expressed in two metrics: distance and travel time. The latter can also be called the route duration. In the Euclidean space, the shortest path is always the fastest. In our case, however, paths obey the road network with its line segments, their speed limits, allowed travel direction, etc. It is well known that the shortest path for a vehicle on a road network is not always the same as the fastest path. A longer route on a highway can be faster than a shorter route through regular roads, because of the speed limit difference, traffic lights, etc. In a case study, one would likely want to estimate the improvement brought about by the routing optimization in the monetary value as well. We suppose that this monetary value can be represented as a function of both the route duration and its distance. Two major variable costs of waste collection are: labor, i.e. driver and collector wages, and fuel (Greco, Allegrini, Del Lungo, Gori Savellini, & Gabellini, 2015). Wages are paid per hour whereas the fuel consumption is given in L/km. Maintenance costs have a mixed nature. Many parts, e.g. engine, wear out with distance traveled and are, as such, variable costs. Other maintenance costs, like periodic vehicle inspections or changing between summer and winter tires, are fixed. Generally, the variable part of the maintenance costs is a function of mileage (e.g. engine life). Hence the variable costs $C(r_i)$ for any given route r_i can be approximated as a function of both its duration $T(r_i)$ (h) and its length $L(r_i)$ (km):

$$C(r_i) = T(r_i) \cdot (w_{dr} + n_{col} \cdot w_{col}) + L(r_i) \cdot (f \cdot p + m) \qquad (1)$$

where:

- w_{dr} is the driver's hourly wage, €
- w_{col} is the collector's hourly wage, €
- n_{col} is the number of collectors per vehicle
- p is the fuel price per liter for the fuel type used, €
- f is the fuel consumption of the garbage trucks used in Almere, L/km
- m is the average maintenance costs per km travelled, €

A deeper insight shows that the real relation between the variables is much more complex: fuel consumption can vary greatly depending on the vehicle's weight, and the weight of a garbage truck underway steadily increases as the it collects more waste. Moreover, the age of a vehicle and its speed dynamics (acceleration, deceleration) must be considered as well. Maintenance can be a factor of both time and mileage. Without any information from the hauler, we lack a firm basis for estimating the fuel consumption and the maintenance costs. We believe, however, that the fastest route is usually the cheapest one as well (an assumption that can perhaps be challenged by the reader). Therefore, we make the travel duration an objective in the minimization problem that we formulate in this study. We encourage the reader, however, to challenge our assumptions and develop a more complex and realistic cost function, especially if the required technical details are available.

For this study, we reduce the cost function of a route to a function of its duration:

$$C(r_i) = T(r_i) \qquad (2)$$

And the total cost of a solution can be calculated as a sum of the costs of the routes $R = \{r_i, \dots, r_{K_{max}}\}$:

$$C(R) = \sum_{k=1}^{k} r_i \qquad (3)$$

3.4 Input data: road dataset

One popular source of road data is OpenStreetMap. Since it is open for anyone to make edits, it may require a quality check, although openness can also mean more frequent updates. Nevertheless, we decided to use the official national road dataset, *Nationaal Wegenbestaand (NWB),* instead. The comparison between OpenStreetMap and NWB is beyond the scope of this study. NWB is open data and it can be downloaded from the website of Rijkswaterstaat along with the accompanying documentation (Rijkswaterstaat, 2019). For further processing, we clipped the dataset to the administrative boundaries of Almere using the corresponding QGIS tool (fig.4):

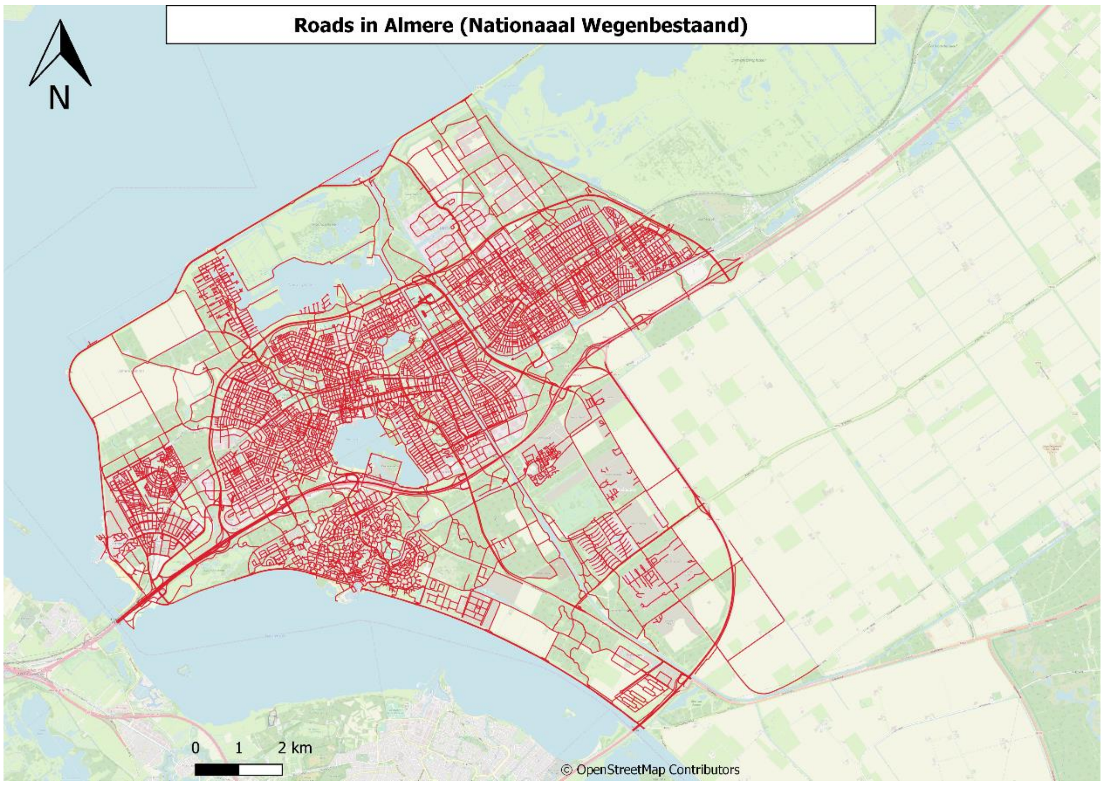

Roads in Almere (Nationaaal Wegenbestaand)
N
0 1 2 km
© OpenStreetMap Contributors

Figure 4. Representation of NWB for the territory of Almere (in red).

NWB does not specify speed limits or any other metric that can be used to calculate the time it takes to traverse any given road segment. Since such metric is required for solving the problem, we augmented NWB as follows. First, the length of each segment was added as a separate attribute (use '$length' operator in QGIS). Next, we added a new attribute to store the speed limit for each road segment. The best guess about its value can be made based on the road type expressed by its 'routeletter' attribute. It is 'A', 'N' or 'E' for highways and *NULL* (empty) for other roads. On highways, trucks are limited to 80 km/h; on other roads, the limit is 50 km/h for the built-up areas and 80 km/h otherwise (ANWB, 2019). We are unaware of any datasets that would demarcate the built-up areas for the given territory. Since Almere is mostly urbanized, we consider all its area to be built-up and define the speed limits as follows:

Table 1. *Speed limits approximation for NWB in Almere.*

Routeletter	*Speed limit, km/h*
NOT NULL (A, N or E)	80
NULL	50

Speed limit is an extreme approximation for the expected travel speed. We therefore multiplied it by a constant factor x. For this study, we arbitrarily chose $x = 0.7$. For real-world applications, we recommend using traffic data instead, if possible. Now, the estimated travel duration t_{ij} for any given line segment (edge) e_{ij} that connects junctions i and j, can be calculated by simply dividing the expected travel speed on it by the segment's length l_{ij}. Finally, we converted it from hours into minutes. Now this attribute represents the 'cost' of the traversing the given edge, or, in graph theory terms, the weight of the edge. The formula for the above procedure is presented below:

$$t_{ij} = \frac{l_{ij}}{x * speed_{ij}^{max}} \qquad (4)$$

where $speed_{ij}^{max}$ denotes the speed limit for edge e_{ij}.

Another limitation of NWB is that the allowed travel directions (both ways / one way) are only specified for highways (routes classified as 'A', 'N' or 'E'). We are unaware of any open sources that could augment NWB in this respect. Therefore, we simply assume all road segments for which the direction is not specified (NULL) to be traversable in both directions. If the direction is specified as 'H' (forward only) for a given segment e_{ij}, we set $t_{ji} = 10^6$. That is, travel duration for the reverse edge is set to a large value that by far exceeds any real travel durations in the given network and will make the optimization algorithm always prefer other road segments.

3.5 Input data: container locations

Basisregistratie Grootschalige Topografie (BGT), developed by Kadaster (Kadaster, 2019), is an open spatial dataset that contains miscellaneous objects, from buildings and roads to street furniture and waste containers. We chose it over OpenStreetMap for the same reasons as described in the previous section. Waste containers in BGT (4854 features in total) belong to the category 'Bak' ('container', IMGeo, 2019) and are divided into three types:

1. 'afval apart plaats' (recycling bin with underground collection space)
2. 'container' (larger containers without underground collection space)
3. 'afvalbak' (smaller street bin)

A closer look at the data, however, revealed significant discrepancies between BGT and the information presented on the website of the municipality of Almere (Gemeente Almere, 2019). For example, there are only 34 'afval apart plaats' items in BGT, and the majority of the actual recycling bins (as presented on the municipality's website) are missing. Since we were not able to obtain any data from Almere, we use random selections, 100 and 500, from the 4854 containers stored in BGT for the territory of Almere. Fig. 5 & 6 show these selections:

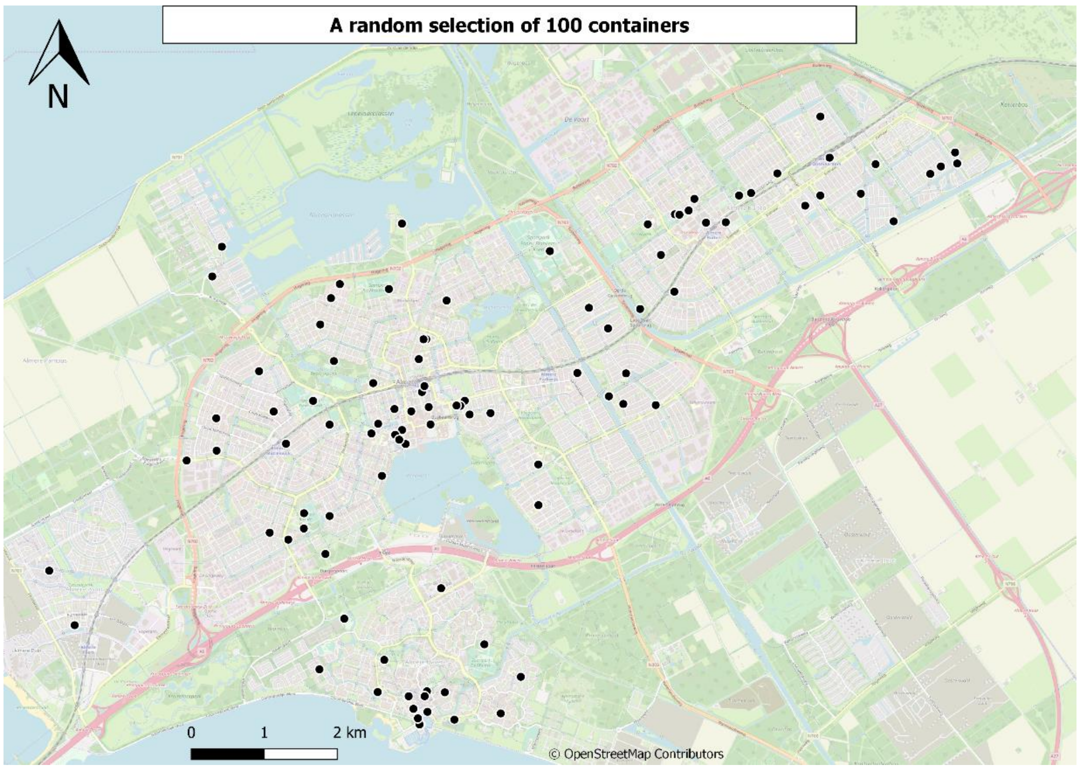
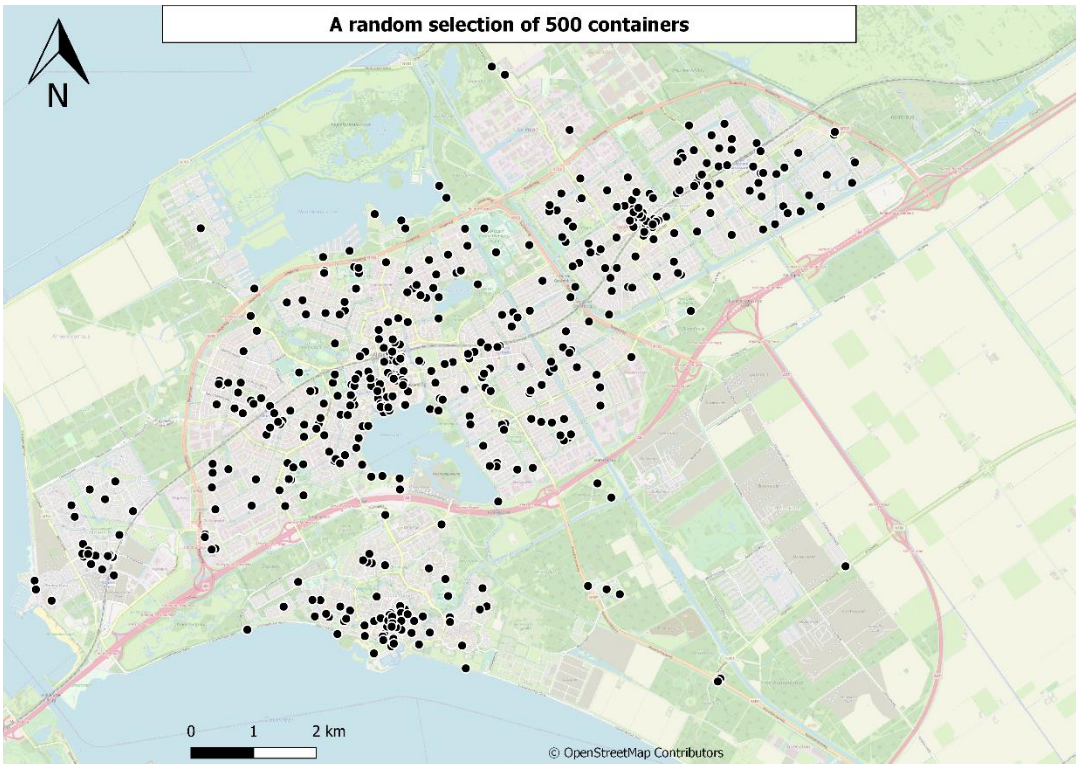

A random selection of 100 containers
N
0
1
2 km
© OpenStreetMap Contributors

Figure 5. 100 randomly selected waste containers from BGT in Almere.


A random selection of 500 containers
N
0
1
2 km
© OpenStreetMap Contributors
Figure 6. 500 randomly selected waste containers from BGT in Almere.

For the implementation and testing, we also needed the type of waste, volume and height for each container. In section 4.2., we describe how we use these parameters to predict accumulation rates. For now, let us say that BGT does not provide any of them, and we had to mock them up. Almere separates municipal solid waste into five categories (Gemeente Almere, 2019):

- Paper and cardboard
- Plastic, tins and packaging (PMT)
- Glass (mixed)
- Organic (groenafval)
- Unsorted (restafval)

We randomly assigned an equal number of containers in each of our two selection to each type of waste, e.g. 20 containers to each category for the 100-container selection. For the height, we use 2000 mm, 2 meters (the maximum range of our sensor), for *all* containers in both selections. For the volumes, we used two values, $3m^3$ and $4m^3$, and split each selection in half among those two. This mock data is not backed by any observation and may deviate from reality, but as we say in this paper, without access to the actual data, we focus more on the prototype development and we believe that should the reader use our prototype for a case study, our simulated data can be easily replaced with the actual data.

3.6 Converting road network into a complete graph

Solving a (C)VRP(P) requires knowing the commute cost between each pair of destinations (waste container locations in our case) and each destination and the depot. In reality, however, the most commonly available input data are road datasets, e.g. *Nationaal Wegenbestaand (NWB)* in the Netherlands. Such datasets represent a *road network,* made of *road segments* and the junctions between them. Road segments can be represented as single or multilines (multilines in NWB). They are usually relatively short, so most roads consist of multiple road segments. By using a road network itself to solve a VRP, one would find a set of tours that visit all the *junctions*, but we are interested in *containers* instead. This brings up a problem of *converting a road network (a sparse graph) into a complete graph.*

A graph is complete if all pairs of its vertices are connected by edges. Therefore, such transformation requires finding the *shortest (or fastest) path* between each pair of containers and between each container and the depot. Finding the shortest path between two vertices is a common task in the field of network analysis and most GIS applications feature at least one tool to solve it. GRASS GIS has a tool for finding all shortest path for a set of locations, for each pair of them. The algorithm is called *v.net.allpairs* and we use it via QGIS because of the extra GUI convenience it provides. However, it can also be executed in the command prompt. Our choice fell on this tool because, firstly, GRASS is FOSS, and, secondly, it does not require extra scripting to batch the execution (repeat it for all pairs of vertices). The algorithm description can be found the official GRASS GIS documentation (Bundala, Bergenheim, & Metz, 2009).

This tool does, however, have its drawbacks, the main one being that it returns each shortest path by segment (each segment as a separate record), therefore the output requires further aggregation to obtain single geometry for every pair of vertices. We use QGIS native *Aggregate* tool for this purpose. We will comment on the performance of this combination in more detail in chapter 6.

3.7 Other procedures & methods

- Interviews with municipalities Almere (in person, Appendix A) and Houten (Appendix B)
- Experimental software development (Appendix C)

4 Methods and algorithms for solving the CVRP(P)

This chapter presents the answers to the research questions formulated for the *first objective* (see section 1.5). We break down the VRPP into the subset selection procedure (selecting which containers to empty) and the VRP itself, which due to the prominence of the vehicle capacity constraint, we call the Capacitated Vehicle Routing Problem (CVRP), in line with other related research. The chapter starts with formulating the CVRP as a mathematical optimization problem, proceeds with the exact methods for solving it, after which it presents some relevant heuristics and metaheuristics.

4.1 CVRP: Problem formulation

Let $G(V, E)$ be a complete directed graph where $V = \{v_0, v_1, \dots, v_n\}$ is the set of vertices, v_0 represents the depot and $\{v_1, \dots, v_n\}$ represent the waste containers, and $E = \{e_{ij} : i, j \in V, i \neq j\}$ is the set of edges, representing the shortest paths between each pair of vertices. Each vertex v_i has an associated demand $q_i \geq 0$ that represents the amount of waste to be collected from that container in kg. Each edge e_{ij} has an associated cost c_{ij} (in minutes) that is in our case represented by the time it takes to traverse it (travel duration), but can also represent cost of a different type, e.g. monetary cost, depending on the nature of the underlying real-world problem. Since the shortest paths e_{ij} follow the road network, and do not, generally, represent the Euclidean shortest paths, the costs of traveling between any to vertices are not, generally, equal: $c_{ij} \neq c_{ji}$. This implies that the graph is directed and the CVRP is directed.

Let K_{max}denote the maximum number of vehicles available. The fleet is homogeneous, i.e. all vehicles have the same capacity Q_{max} and a maximum tour duration T_{max} that is imposed by the working hours. A solution is defined as a set of routes $R = \{r_1, \dots, r_{K_{max}}\}$ where some of the routes can be empty ($r_k = \emptyset$). Each non-empty route $r_k \subseteq E$ is an ordered array of edges $r_k = [e_{0i}, \dots, e_{j0} : v_i, v_j, v_0 \in V]$. For convenience, let us write $e_{ij} \in r_k$ if edge e_{ij} is part of route r_i and $v_i \in r_k$ if any edge e_{ix} or e_{xi}, $\forall\, v_x \in V$, makes part of route r_i. In a 'classical' VRP, each route is a Hamiltonian cycle, i.e. a graph in which each vertex v_i, $deg^-(v_i) = deg^+(v_i) = 1$ (considering the graph is directed), but we assume that this requirement may be unsatisfiable, e.g. due to the presence of dead-end streets, and is not necessary for solving the problem. Each route r_k can be measured in terms of its total duration $T(r_k) = \sum_{e_{ij} \in r_k} c_{ij}$ and the total amount of waste collected $Q(r_k) = \sum_{v_i \in r_k} q_i$. A route is considered feasible if starts and ends at the depot, contains no cycles that do not include the depot, and the vehicle capacity and tour duration constraints are satisfied: $T(r_k) \leq T_{max}$ and $Q(r_k) \leq Q_{max}$. A solution is considered feasible if all its routes are feasible. We will use these conventions throughout this chapter, so their summary is presented in table 2**:**

Table 2. Main variable names that we use in this chapter.

$v_i \in V$	vertex (container)
$v_0 \in V$	depot
$e_{ij} \quad v_i, v_j \in V$	edge (shortest path between v_i and v_j)
$q_i \quad v_i \in V$	amount of waste in container v_i (q_0 = 0) in kg
$c_{ij} \quad v_i, v_j \in V$	edge cost (travel duration) in min
Q_{max}	vehicle capacity
T_{max}	maximum route duration
K_{max}	maximum number of vehicles (routes)
$r_k \in R$	route (tour)
$T(r_k)$	route duration

4.2 Reducing the VRPP to the VRP: container selection

In the definition presented in the previous section, container set V is a given and represents, in fact, a subset of all containers that was selected for emptying on the given day t_x, $(x \in \mathbb{Z})$. How does this selection happen? The daily updates coming from the sensors suggest the selection can be done 'on-the-fly', on the same day the collection is performed and for that day only, and the day after the same procedure will be repeated with up-to-date fill levels from the sensors. This system would be the opposite of the traditional approach where there was only planning via forecasting and minimal to no (near-)real-time information. An example of such system is described in Faccio, Persona & Zanin (2011).

Most researchers, however, believe that planning cannot be discarded, but should rather be improved by using sensor data (Ferrer & Alba, 2018; Lopes et al., 2015; Nuortio et al., 2006; Ramos et al., 2018b). The role of sensors here is to provide accurate measurements automatically and at regular intervals, so that planning can be based on actual fill levels rather than statistical approximations or long-term observations. This is where the problem acquires a temporal dimension. We assume that the time grain for waste collection planning is one day: the obtained solution consists of a set of routes that covers the whole day. For the next day, another set of routes must be found. We also assume that any given container is emptied at most once a day. Our goal is to keep the *service level* at 100%, i.e. let no containers overflow.

Let $V^* = \{v_1, v_2, \dots, v_n\}$ denote the set of all containers serviced by our system. All of them are equipped with sensors that report the fill level once a day. These records make up time series of the form $[\dots, h_i(t_{x-1}),\ h_i(t_x),\ h_i(t_{x+1}), \dots], v_i \in V^*$ and t_x the x-th day of observation, $h_i(t_x)$ is the distance between the lid of the container and the waste body (mm) for any container $v_i \in V^*$. It can be converted into a fill level given the total height of the inside of the container H_i using the following formula:

$$\gamma_i = 1 - \frac{h_i}{H_i} \quad [0,1], \qquad v_i \in V^* \quad (5)$$

Fill level can, in turn, be converted into weight, the measure in which the garbage trucks saturate their capacity. Given that the waste type and the volume for each container is known from the manufacturer, we only need to obtain the average density for that waste type. Considering that the categories for waste containers are relatively broad ('paper and cardboard', 'organic waste', etc.), the average density may deviate from the actual measurement. The precise estimation of these indicators is beyond the scope of this study and the capabilities of the authors, so we use third-party resources (EPA Victoria, 2018; Stimular, 2018):

Table 3. Average material density for the different types of waste.

Waste type	*Average density (kg/m^3)*
Paper and cardboard	120
Plastic, tins and packaging (PMT)	70
Glass (mixed)	300
Organic (groenafval)	300
Unsorted (restafval)	50

Using this information, the actual amount of waste q_i (kg) in each container can be calculated:

$$q_i = \gamma_i * \omega * \rho, \qquad \forall\, v_i \in V^* \quad (6)$$

Here, ω is the container's volume (m^3) and ρ is the material's density (kg/m^3).

Now, to be able to forecast, we need to calculate *accumulation rates* for each container. The absolute fill levels should therefore be differenced to arrive at time series of the form $[\ldots, \Delta_i(t_x), \Delta_i(t_{x+1}), \Delta_i(t_{x+2}), \ldots]$ where $\Delta_i(t_x)$ is the difference between day t_x and the preceding day for any given $v_i \in V^*$. The simplest way would be to compute the mean of the increases over a certain period $(t_1, \ldots, t_n)$:

$$a_i = \frac{\sum_{x=1}^{n} \Delta_i(t_x)}{n} \quad [0,1], \qquad \forall\, v_i \in V^* \qquad (7)$$

There may be, however, some cyclicity in the increases, for example weekly (similarity in waste disposal on the same day of the week in consecutive weeks) and annual patterns (similarity in waste disposal on the same calendar date in consecutive years) (Johnson et al., 2017; Kannangara, Dua, Ahmadi, & Bensebaa, 2017). These patterns may be location-specific, however. *Autocorrelation* can be used to detect such patterns. However, we do not implement it for the reasons that the discuss in chapter 6.

The expected overflow date $t_i^{overflow}$ can be found for any container $v_i \in V^*$ by adding the number of days until its expected overflow to the current date $t^{current}$:

$$t_i^{overflow}(v_i) = t^{current} + \left\lceil \frac{1-\gamma_i}{a_i} \right\rceil \qquad (8)$$

This formula must, of course, obey the 'date arithmetic'.

Now, $\mu_{overflow}(v_i) - 1$ defines the latest possible collection date for the given container. Besides maintaining a 100% service level, it is desirable to avoid collecting waste more frequently than needed, as it causes extra costs and air pollution. Therefore $\mu_{overflow}(v_i) - 1$ would be the optimal collection date, by default. However, it must be a workday: Saturdays, Sundays and bank holidays must be excluded. It must be mentioned that the work schedules in waste management may differ between the countries, regions, etc., and the exact rules as to which days to exclude should be defined considering the local principles.
If a given date is not a workday, the collection must be shifted. Since shifting to a later date would violate the required service level, it can only be rescheduled to an earlier date. The algorithm does it one day at a time and stops once it has found a valid calendar date.

4.3 Exact methods for solving the CVRP

Exact methods for solving the VRP include Mixed Integer Linear Programming and Branch-and-Bound that are described below but are not limited to those two. For a comprehensive overview of the exact methods, see Laporte (1992) and Archetti, Speranza & Vigo (2014).

4.3.1 Mixed Integer Linear Programming

This formulation is based on Archetti, Speranza & Vigo (2014) and Ramos, de Morais & Barbosa-Povoa (2018b). Given a fixed number of available vehicles K_{max}, a set of tours $R = \{r_1, \ldots, r_{K_{max}}\}$ should be defined, such that:

- each non-empty tour r_k starts and ends at the depot v_0 (some tours may be empty)
- the duration of any tour cannot exceed the limit
- the number of tours cannot exceed the maximum available number of vehicles
- the total amount of waste collected on any tour cannot exceed the vehicle capacity

Let us define two decision variables:

- y_i^k, a binary variable equal to 1 if $v_i \in V$ is visited on route r_k and 0 otherwise
- x_{ij}^k, a binary variable equal to 1 if $e_{ij} \in \mathrm{E}$ is traversed on route r_k and 0 otherwise

Now, here is the formulation of the problem:

$$minimize \sum_{k=1}^{K_{max}} \sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} * x_{ij}^k, \qquad v_i, v_j \in r_k, i \neq j \qquad (9)$$

$$subject\ to:$$

$$\sum_{k=1}^{K_{max}} y_0^k \leq K_{max} \qquad (10)$$

$$\sum_{i=0}^{n} \sum_{j=0}^{n} c_{ij} * x_{ij}^k \leq T_{max}, \qquad \forall\, r_k \in R \qquad (11)$$

$$\sum_{i=1}^{n} q_i * y_i^k \leq Q_{max}, \qquad \forall\, r_k \in R \qquad (12)$$

$$y_i^k \in \{0,1\} \qquad \forall\, v_i \in V, r_k \in R \qquad (13)$$

$$x_{ij}^k \in \{0,1\} \qquad \forall\, e_{ij} \in E, r_k \in R \qquad (14)$$

The objective function (9) minimizes the total time spent by the fleet of vehicles. Constraint (10) limits the number of routes to the number of vehicles required. Constraint (11) limits each tour's duration and constraint (12) ensures that the total amount of waste collected on each tour is feasible with regards to vehicle capacity Finally, (13) and (14) are the definitions of the decision variables.

4.3.2 Branch and Bound

Branch-and-bound algorithms use state space search for finding the best solution: the set of all candidate solutions forms a rooted tree with the full set at the root. Each branch of the tree represents a subset of the solution set. Each state tree's branch is first checked against the upper and lower estimated bounds on the current optimal solution and is discarded if it cannot produce a better solution than the best found so far by the algorithm. The algorithm depends on efficient estimation of lower and upper bounds of regions/branches of the search space. If no bounds are available, the algorithm degenerates to an exhaustive search. The formulation presented here is based on (Laporte, 1992). First, we first introduce $K_{max} - 1$ new (artificial) depots. Now we have one depot vertex for each vehicle and $n' = (n - 1 + K_{max})$ vertices in total. This defines a new complete set of vertices $V' = \{v_0, v_1, \dots, v_,\}$, such that $V \subseteq V'$, and a new complete edge set $E' = \{e_{ij}, v_i, v_j \in V'\}$. Let us define an extended cost matrix $C' = [c'_{ij}]$ as:

$$C_{ij} \quad \begin{cases} c_{ij} & (v_i, v_j \in V), \\ c_{i0} & (v_i \in V \backslash \{v_0\}, v_j \in V' \backslash V), \\ c_{0j} & (v_i \in V' \backslash V, v_j \in V \backslash \{v_0\}), \\ 0 & (v_i, v_j \in (V' \backslash V) \cup \{v_0\}) \end{cases} \qquad (15)$$

The following allows to eliminate subtours. Let $k_{min}(V'')$ denote the minimum number of vehicles required to visit any given subset of vertices V'' such that $V'' \subset V' \backslash \{v_0\}$ and $|V''| \geq 2$:

$$k_{min}(V'') = \left\lceil \frac{\sum_{i \in V''} q_i}{Q_{max}} \right\rceil \qquad (16)$$

which comes from the observation that the following must apply:

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \geq k_{min}(V''), \;\; v_i \in V'', v_j \in V' \backslash V \qquad (17)$$

$$|V''| = + \sum_{i \in V''} \sum_{j \in V' \backslash V} x_{ij} \qquad (18)$$

The following subtours are illegal in this definition:

- Subtours over a set V'' of vertices $V \backslash \{v_0\}$
- Subtours that violate the capacity or maximum duration constraints:

Now, let us define the VRP of interest:

$$minimize \sum_{i=1}^{n'} \sum_{j=1}^{n'} c_{ij} * x_{ij}, \quad i \neq j \qquad (19)$$

$$subject\ to:$$

$$\sum_{i=1}^{n'} \sum_{j=1}^{n'} x_{ij}, \quad v_i, v_j \in V' \qquad (20)$$

$$x_{ij} \in \{0,1\} \;\; \left(v_i, v_j \in V'; i \neq j\right) \quad (21)$$

$$\sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij} \leq |V''| - k_{min}(V''), v_i, v_j \in V'' \qquad (22)$$

(19), (20) and (21) define a modified assignment problem where assignments on the main diagonal are not allowed and constraint (22) prevents subtours.

Since we transformed the problem at the start by adding duplicate depot vertices, the resulting solution must first be corrected. For each edge e_{ij} in the solution:

- if $v_i \in V \backslash \{v_0\}\ and\ j \in V' \backslash V$, replace e_{ij} by e_{i0}
- if $v_i \ \in V' \backslash V\ and\ j \in V' \backslash \{v_0\}$, replace e_{ij} by e_{1j}
- if $v_i, v_j \in V' \backslash V$, delete e_{ij}

4.4 Tour construction heuristics

VRP is an NP-hard problem which means that the time required to find the optimal solution grows exponentially with the number of vertices (destinations). The size of the VRPs that can be solved by exact methods within a reasonable amount of time constantly grows with the advance in computing power but the problems with hundreds of destinations, as may be the case with garbage collection, may still take many hours or even days to solve on a regular machine, especially with the increase in the number of constraints (capacity, tour duration, number of vehicles, etc.). For this reason, heuristics are widely used to solve large problems in a reasonable amount of time.

The performance of any heuristic A can be measured as a ratio $\frac{T(S_A)}{T(S_{OPT})}$ where S_A is the solution obtained by the heuristic and S_{OPT} is an optimal solution obtained by one of the exact methods, hence $T(S_A)$ and $T(S_{OPT})$ are their respective costs measured in terms of the objective's value. The ratio $\frac{T(S_A)}{T(S_{OPT})}$ measures the degree of departure of the solution obtained by heuristic A from the optimal solution S_{OPT} (Cordeau, Gendreau, Laporte, Potvin, & Semet, 2002; D. Johnson & McGeoch, 1995). The heuristics presented in this section are based on Prodhon & Prins (2016), Cordeau et al. (2002), Johnson & McGeoch (1995), Vidal et al. (2013) and Rosenkrantz et al. (1977).

4.4.1 A foreword on triangle inequality and directed graphs

For each heuristic, the worst- and best-case performance ratios may depend on whether the given problem satisfies the triangle inequality, i.e. for any triangle, the sum of the lengths of any two sides must be greater than or equal to the length of the remaining side. In our case, a triangle is formed by a triplet of vertices, and their sides are the shortest paths between those vertices, i.e. edges of our graph (Johnson & McGeoch, 1995; Prodhon & Prins, 2016). For example, for any three vertices $v_i, v_j, v_k \in V$, $c_{ij} + c_{jk} \geq c_{ki}$, $c_{ki} + c_{jk} \geq c_{ij}$ and $c_{ij} + c_{ki} \geq c_{jk}$. The problems that satisfy triangle inequality can be called *metric* or *Euclidean*, and those that do not – *non-metric* (Skopal, 2006).

In our problem, the shortest paths between the destinations do not, generally, correspond to the Euclidean distance between them since they are calculated along the road network. Most authors run tests only on metric problems. In fact, finding the worst- or best-case bounds for any TSP/VRP heuristic for problems that do not necessarily satisfy the triangle inequality is NP-complete (Rosenkrantz, Stearns, Lewis, Ravi, & Shukla, 1977). Therefore, we advise the reader to take the information about the performance of the heuristics in this chapter with caution. We always mention whether the triangle inequality has an effect on the algorithm's performance. However, triangle inequality can be *enforced* by adding a large enough constant to each edge's cost in the graph (Rosenkrantz et al., 1977; Skopal, 2006). The present an example in fig.7 below: the original graph on the left does not satisfy the triangle inequality as $c_{ij} + c_{ik} < c_{kj}$ (the graph is undirected but the same can be applied to a directed graph). By adding a constant $const = 2$ to each edge's cost, we enforced the triangle inequality (the graph on the right).

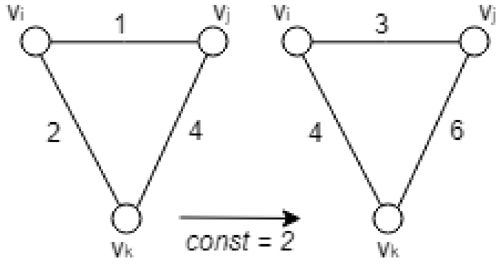


Figure 7. Triangle inequality inforcement. The graph on the left does obey the triangle inequality ($c_{ki} + c_{ij} < c_{jk}$). The graph on the right, produced by incrementing the cost of each edge by 2, does.

The minimum required value of $const$ depends on the values of the edge weights (costs) and is, as such, problem dependent. In this study, we do not cover the method for finding the suitable value for $const$ and the related matters of enforcing the triangle inequality. The cost of the solution found after applying this enforcement will differ from the 'true' cost by $const * \sum_{k=1}^{K_{max}} |r_i|$, i.e. by the number of edges in the solution multiplied by the chosen constant. The optimality of the solution will not be affected however, as all edges have been incremented by an equal amount, and the true cost can be found by subtracting it from the 'false' cost.

Another important thing to consider when talking about heuristics is the whether the graph on which the problem is solved is directed. Some methods work only on undirected graphs, at least in their original formulation, so we do not cover them in our study. One example is the *Double Minimum Spanning Tree* heuristic. But the most prominent of those is the Christofides method. First presented in 1976, it still has the best worst-case performance ration among all existing tour construction heuristics for symmetric problems (Vidal, Crainic, Gendreau, & Prins, 2013). If $c_{ij} = c_{ji}\ \forall\ e_{ij} \in E$, the problem is said to be *symmetric* and the graph to *undirected.* Otherwise, the problem is *asymmetric,* and the graph is *directed.* There exist, to our knowledge, some adaptations of this method to asymmetric problems (Roughgarden, 2016), but leave those beyond the scope of this study.

4.4.2 Nearest Neighbor

This is arguably the simplest heuristic: starting from the depot, the algorithm iteratively adds the closest vertex (to the latest added vertex in the tour). A greedy approach, it is likely to yield routes that are far from being optimal and can sometimes find no feasible solution at all, even if such exists for the given problem. The benefits are, of course, the ease of implementation and a runtime of $O(n^2)$. For the problems that satisfy the triangle inequality, $\frac{T(S_A)}{T(S_{OPT})} \leq 0.5(\lceil \log_2 N \rceil) + 0.5)$ (Rosenkrantz et al., 1977).

Acting *sequentially*, a heuristic will be constructing one tour at a time, and it will finish the tour by linking it back to the depot if the next candidate node violates the constraints: makes the tour exceed its length or duration limits, the vehicle capacity, etc. *Sequential* construction naturally tends to minimize the number of routes, thus minimizing the number of vehicles used. It may perform better for the problems where using as few vehicles as possible is an objective. At the same time, sequential construction makes the eventual number of routes unpredictable and often results in the last route being much shorter than the rest. If the number of vehicles is fixed than the parallel way is the only option.

Acting in a *parallel* fashion, the heuristic will initiate one tour for each vehicle and add one vertex to each route in every iteration. The principle for adding the vertices remains the same, but the order in which the routes are modified in each iteration may affect the results. Imagine a vertex equidistant from the endpoints of two or more routes. In the Nearest Neighbor heuristic, the choice of the route to add it to will depend purely on the arbitrary order in which they have been initialized. Besides, there is no way to minimize the number of vehicles used. Fig. 8 below shows the sequential and the parallel ways of the Nearest Neighbor heuristic.

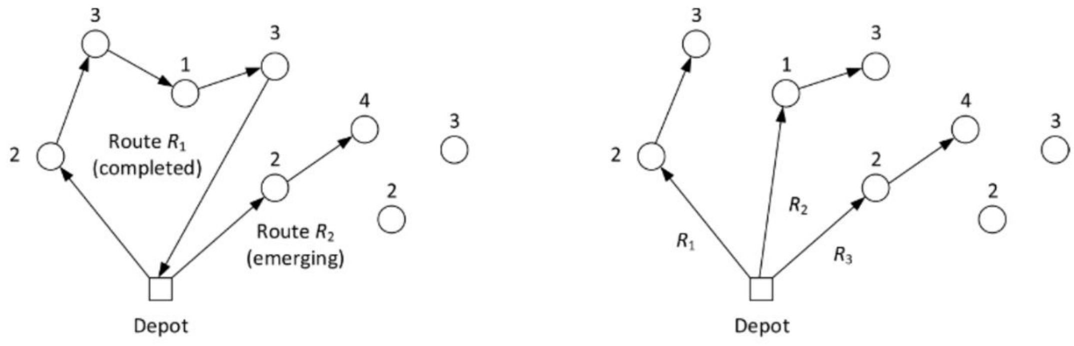


Figure 8. Nearest Neighbor heuristic. Left: sequential. Right: parallel (Prodhon & Prins, 2016). Prodhon & Prins use R_i instead of r_i to denote a single route. Numbers are the costs of adding the node.

4.4.3 Nearest Insertion, Farthest Insertion and Cheapest Insertion

Nearest Insertion works the following way. For a given emerging tour r_i find v_x ($v_x \notin r_i, v_x \in V$) closest to it (in terms of perpendicular distance). For that v_x, find $e_{jk} \in r_i$ such that $increase = c_{jx} + c_{xk} - c_{jk}$ is minimal; introduce such e_{jx} and e_{xk} in r_i and remove e_{jk}. If $r_i = \emptyset$, simply introduce two edges e_{0x} and e_{x0}.

Farthest insertion only differs in that the farthest node from the current tour is selected. This might feel counter-intuitive, but the general idea is to establish an outline of a tour first, and then 'fill in'. Rosenkrantz et al. (1977) argue that it prevents accidental deletion of the shorter edges by later insertions.

Cheapest Insertion method first computes $increase = c_{jx} + c_{xk} - c_{jk}$ for each $v_x \notin r_i$, and chooses the insertion candidate with the minimum $increase$ value. That is, Nearest insertion chooses the closest candidate vertex and finds the best way to introduces it in a tour while Cheapest Insertion precomputes the costs of insertion and chooses the cheapest candidate.

Both Nearest and Farthest Insertion run in $O(n^2)$; Cheapest Insertion – in $O(n^2 \log n)$. All three methods have a fixed worst-case performance ratio of $\frac{T(S_A)}{T(S_{OPT})} \leq 2$ (Johnson & McGeoch, 1995; Prodhon & Prins, 2016).

The way these three heuristics can be adopted to VRP is the same as in Nearest Neighbor: either by sequentially augmenting one tour until the constraints are violated or by initializing K_{max} tours and augmenting one tour with one vertex at a time.

4.4.4 Clarke-Wright Savings Method

This method generates an initial solution in which one vehicle is assigned to each destination: it travels from the depot to the destination and back, the next one travels to its respective destination and back to the depot, and so on. This, obviously, creates a very expensive solution that most likely violates the maximum number of vehicles constraint; hence the constraint is relaxed:

$$R = \{\, [e_{0i}, e_{i0}], \dots, [e_{0n}, e_{n0}] \,\}$$

At the second step, the method determines how much can be saved if any two tours are merged: if instead of sending a new vehicle to the next destination, we could let the current vehicle travel to the next destination without returning to the depot. To do so, we calculate the saving σ_{ij} for each pair of destinations $v_i, v_j \in V$:

$$\sigma_{ij} = c_{i0} + c_{0j} - c_{ij} \qquad (23)$$

The algorithm must avoid cycles and isolated edges by ensuring that for each vertex v_i, $deg^-(v_i) = deg^+(v_i) = 1$ (for directed graphs) or $deg(v_i) = 2$ (for undirected graphs). An example is shown in fig. 9 we eliminated the return paths to the depot and connected the destinations directly to obtain two routes.

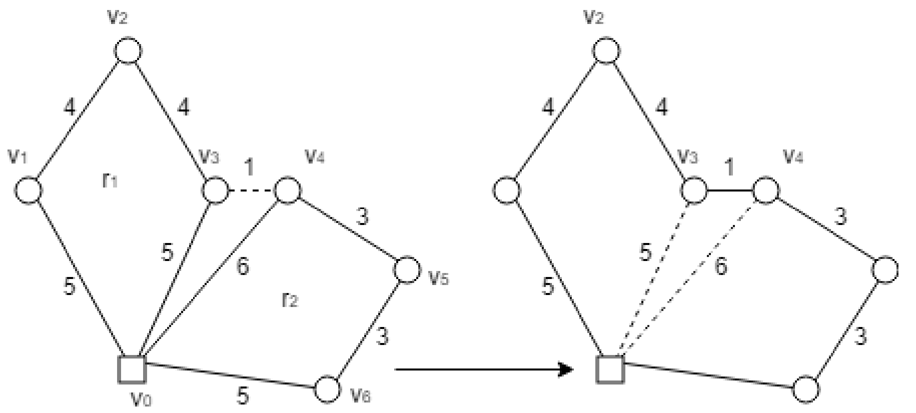


Figure 9. Clarke-Wright savings method. Here, we assume that σ_{ij} is on top of our savings list. Hence, we connect those vertices and remove the edges that connect them with the depot.

The implementation can be *sequential:*

1. Build an initial solution S_0 and compute saving for all pairs of vertices and sort them in the descending order
2. Consider, in turn, each route $r_x = [e_{0i}, e_{j0}]$ in the initial solution S_0
3. Determine the first σ_{ki} or σ_{jl} from the top of the list that can be used to feasibly merge r_x with another route r_y that either starts with e_{0l} or ends with e_{k0}
4. If there are no such mergers, repeat the previous steps for the next route r_{x+1}; otherwise, implement the merger and repeat the previous steps for the resulting route

Alternatively, a *parallel* version can be implemented:

1. Build an initial solution S_0 and compute saving for all pairs of vertices and sort them in the descending order
2. Starting from the top, check if there exists a pair of routes r_x, r_y such that one of them starts with e_{0j} and the other one ends with e_{i0}
3. If such a pair is found, merge the two routes by adding e_{ij} and removing e_{i0} and e_{0j}
4. Stop if there are no feasible mergers

A merge is infeasible cannot be performed if the resulting route violates the vehicle capacity constraint, the maximum tour duration, and also if it reduces the number of tours in the solution to a value less than K_{min}. If such is not specified, it can potentially reduce the solution to a single TSP tour.
This method also operates in a 'greedy' fashion, because it prioritizes the mergers with larger savings. To address this issue, the heuristic can be adjusted to skip the highest priority merger with a certain probability α and perform the next merger on the list (this concept is known as *randomization)*. The original Clarke-Wright algorithm runs in $O(N^2 \, log_2 N)$ and guarantees $\frac{T(S_A)}{T(S_{OPT})} \leq \left(log_2 N\right) + 1$ for the problems that satisfy the triangle inequality (Johnson & McGeoch, 1995; Prodhon & Prins, 2016). It generally outperforms the Nearest Neighbor and Nearest Insertion heuristics and minimizes the number of tours, thus being well-suited for the problems where the number of vehicles must be minimized.

4.4.5 GENI (Generalized Insertion Procedure)

This method was originally designed by Gendreau et al. (1992) for the TSP, but in a successive paper (Gendreau, Hertz, & Laporte, 1994) the authors describe its adaptation to the VRP. Let us first present the procedure for the TSP, and then show how it can be adapted to produce multiple routes.

The algorithm starts by initializing a tour of three randomly selected and randomly ordered vertices (besides the depot): $\bar{r} = [e_{0i}, e_{ij}, e_{jk}, e_{k0}]$. In each iteration, the algorithm randomly picks one candidate $v_x \in V, v_x \notin \bar{r}$ and considers its insertion between any two vertices $v_i, v_j \in \bar{r}$. Importantly, v_i, v_j need not be consecutive. Let us define $v_k \in \bar{r}$ as a vertex on the path from v_j to v_i and $v_l \in \bar{r}$ – on the path from v_i to v_j $(v_i, v_j, v_k, v_l \in \bar{r})$. Now, for each such combination of v_x, v_i, v_j and v_l, the algorithm considers two types of insertion, code-named '1' and '2' (fig. 10 & 11):

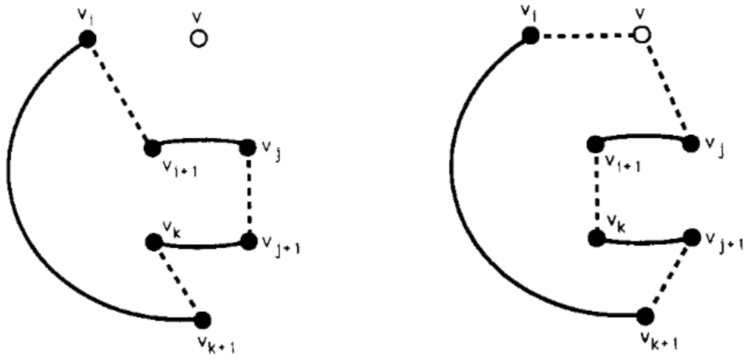


Figure 10. Insertion type I. The left part shows the original graph, the right one – the result; edges that are altered in the process are shown by dashed lines; v is the insertion candidate; (Gendreau, Hertz, & Laporte, 1992)

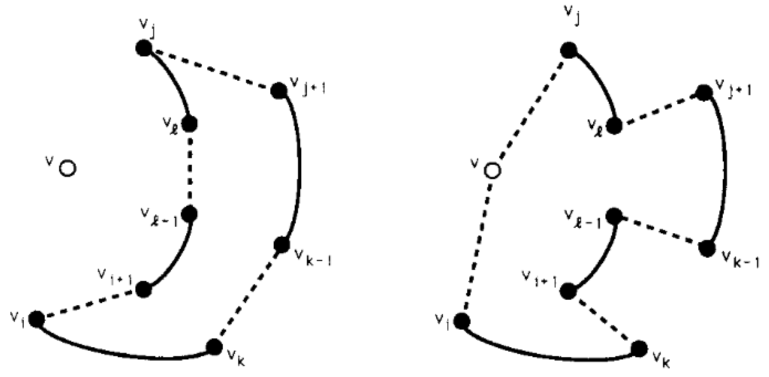


Figure 11. Insertion type II. The left part shows the original graph, the right one – the result; edges that are altered in the process are shown by dashed lines; v is the insertion candidate; (Gendreau et al., 1992)

In both types of insertion, v_i and v_j become immediate neighbors of v_x on either side; some edges are removed, and new ones are introduced. In *type I*, paths $v_{i+1} \dots v_j$ and $v_{j+1} \dots v_k$ are reversed; edges $e_{i,i+1}$, $e_{j,j+1}$, $v_{k,k+1}$ are deleted. In *type II*, paths $v_{i+1} \dots v_{l-1}$ and $v_l \dots v_j$ are reversed; edges $e_{i,i+1}$, $e_{j,j+1}$, $v_{l,l+1}$ are deleted. This means that there are n^4 potential insertions for each candidate vertex (Gendreau et al., 1992). To reduce the time complexity, the method uses the notion of a *p-neighborhood* to limit the number of candidate insertions. Let us define a neighborhood $N_p(v_x)$ as a set of p closest vertices to v_x (in terms of the cost function's value) that are in the same tour. If v_x has fewer than p neighbors on the tour, then they all fall into the $N_p(v_x)$. The value of the p is chosen by the implementor. Let us now select $v_i, v_j \in N_p(v_x)$, $v_k \in N_p(v_{i+1})$, $v_l \in N_p(v_{j+1})$ and consider inserting v_x between any of the v_i, v_{i+1}, as long as $v_i \in N_p(v_x)$. The value of the objective for each of these potential moves is evaluated and the best one is accepted. The p-neighborhoods for each of the vertices are updated in each iteration.

For each candidate vertex, there are $O(p^4)$ possible v_i, v_j, v_k, v_l combinations. Once the best insertion is found, it takes $O(n)$ to update the tour. And there are $n-3$ such iterations for each V, hence the total time complexity of GENI for the TSP is $O(np^4 + n^2)$.

Now, to accommodate multiple routes, the authors propose a *Split* algorithm that constructs a *giant tour*, i.e. a tour with all vertices in V, $\bar{r} = [e_{0i}, \dots, e_{n0}]$ using GENI first and then splitting this tour into at most K_{max} tours. Let us define a procedure $split(tour, vehicleCapacity, durationLimit)$ that works the following way:

1. Initialize $k = 1$ and $R = \emptyset$
2. If $k > K_{max}$, stop; otherwise, Initialize a new tour r_k;
3. If $k = K_{max}$, set $r_k = tour$, add r_k to R and stop; otherwise, go to step 4;
4. Remove the first edge e_{ij} in $tour$ and append it to r_k *if and only if* its insertion does not violate either the vehicle capacity or the tour duration constraint; if it does, proceed to step 4; otherwise, repeat step 3;
5. Reconnect r_k to the depot by appending e_{j0} to it; if for the resulting r_k, $T(r_k) \leq T_{max}$, go to step 6; otherwise, delete e_{j0}, remove e_{ij} from r_k and append it back to $tour$, append e_{i0} to r_k thus reconnecting it to the depot, go to step 6;
6. Reconnect $tour$ to the depot: given that e_{xy} is currently the first in $tour$, insert e_{0x} before it; go to step 7;
7. Set $k = k + 1$, add r_k to R, insert and go to step 2;

The last tour $r_{K_{max}}$includes all remaining destinations and may not be feasible. Alternatively, the last k tours may be empty. To this end, the authors propose a tour improvement procedure US that is described in section 4.5.2.

4.4.6 Comparison

Rosenkrantz et al. (1977) compare the three insertion heuristics described in section 4.4.3, along with the Nearest Neighbor method, on random Euclidean instances, and conclude that Farthest Insertion performs the best on average for small instances (up to 100 nodes), followed by Cheapest Insertion, Nearest Insertion and Nearest Neighbor respectively, but for larger instances the results are less consistent, with only Nearest Insertion performing worse on average than the other three. They also observe that the optimality of the solution obtained empirically does not, generally, correspond with the theoretical performance bounds of the respective heuristics.

Johnson & McGeoch (1995) compare Nearest Neighbor and the Savings method on a set of different problems and report the following results (the results are average over multiple runs): the Savings method provides better solutions on average, with $\frac{T(S_A)}{T(S_{OPT})} \approx 9.6\%$ for small-sized problems (10^2); but the for larger problems, the excess over the optimal solution starts to deteriorate, reaching 12% for problems with 10^5 vertices. For Nearest Neighbor, these figures are 26% and 23%, respectively, hence its performance improves with the problem size. The authors do not explain this behavior. Separately, the authors test GENI on instances containing from 100 to 500 vertices. The conclude that GENI outperforms all other compare heuristics for those problem sizes. The excess over the optimal solution is 9.1% on average for

$p = 3$ and drops to 5.6% for $p = 20$. We only provide the ratios because these results date back to 1995, and the absolute values must have changed dramatically. The Savings method, despite being slower than Nearest Neighbor, gives much better results. Cordeau et al. (2002) argue that this merit, in combination with the algorithm's easy implementation, made the Saving method perhaps the most commonly used tour construction heuristic implemented in software applications. This claim, however, dates back to 2002, and the advance in computing power may have changed the balance of powers since then.

4.5 Tour improvement heuristics

In this section, we discuss local search methods. Local search algorithms start with an *initial solution* S generated by one of the tour construction heuristics and inspect the solutions in its *neighborhood* $\mathrm{N(S)}$. A neighborhood is as a set of similar solutions, where the notion of similarity depends on the employed heuristic. Generally, the neighbor solutions are those that are located one move away from the current *(incumbent)* solution. A move can be a swap of vertices within a tour or between the tours, for example. An improvement heuristic evaluates the solutions is $\mathrm{N(S)}$ iteratively and either accepts or rejects them. Local search procedures can be either *first-improvement* or *best-improvement*. In the first case, the first downhill (improving) neighbor solution S' is accepted, in the second case, all neighbor solutions are inspected and the best one is accepted. The efficiency of these two types depend on the given problem and either one may perform better than the other (Johnson & McGeoch, 1995; Prodhon & Prins, 2016). Last but not least, a downhill move that violates the constraints, such tour duration limit or the vehicle capacity, cannot be accepted. We will later discuss some metaheuristics that allow local search to accept uphill solution under certain conditions, but in the description of these tour improvement heuristics we assume that a move that violates the constraints cannot be accepted.

4.5.1 X-Opt

Algorithms of this family work by iteratively changing X edges in the initially generated solution. At each iteration, X edges are removed and must be replaced with X new edges; $\mathrm{X}!$ ways to reconnect the affected vertices are available (Johnson & McGeoch, 1995). As in other local search methods, these ways are compared one by one, either all of them (best-improving search) or until a downhill solution is found (first-improving search). X-Opt itself originally refers to this local search procedure only, and it must be combined with some other method used to generate the initial solution, e.g. Nearest Neighbor, although some authors do not mention the tour construction method they used.

Since it takes $O(n^x)$ to test all possible moves for a given solution, time complexity quickly grows with X. Increasing the X beyond $X = 3$ usually improves the results by fractions of a percent (if improves at all) while resulting in significantly longer execution times. Therefore, the most widely used variants are 2-Opt and 3-Opt, with the latter often showing better results (Prodhon & Prins, 2016). The former takes an initially generated tour and replaces two of its edges, thus producing a new tour (fig.12).

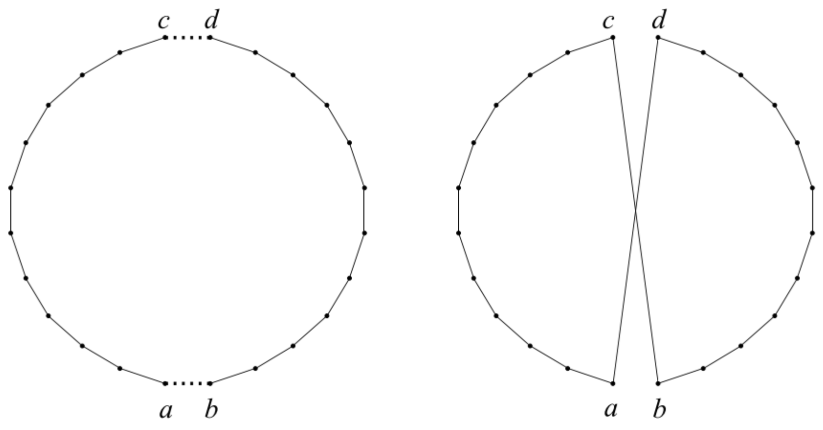


Figure 12. An example of a 2-Opt move.

In the picture above, edges e_{cd} and e_{ba} were removed and e_{cb} and e_{da} were introduced instead. The algorithm performs one or more such moves in each iteration. An iteration ends when a new, downhill solution is accepted; afterwards, the procedure is repeated for the new solution. This procedure stops when it there are no improving moves in the neighborhood of the incumbent solution. Needless to say, this may mean that the algorithm found the optimal solution, or simply got stuck in a local minimum. 3-Opt acts in a similar fashion but changes three edges at a time (fig.13).

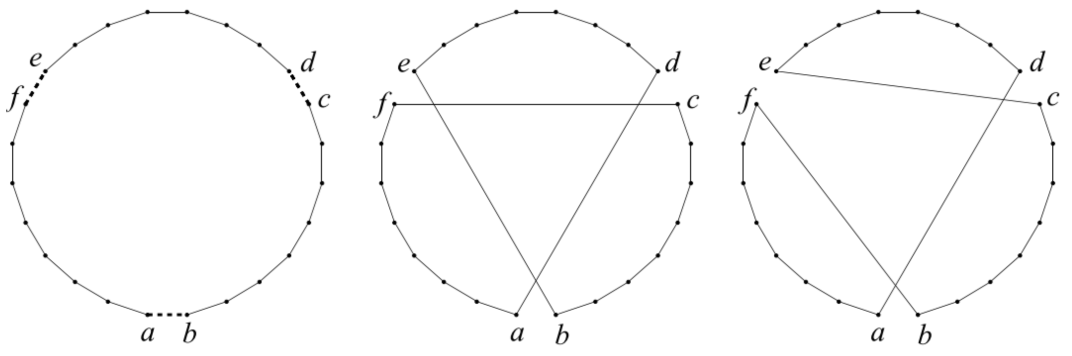


Figure 13. An example of a 3-Opt move.

The solution obtained by X-Opt lies within $0.25\sqrt[2X]{N} \leq \frac{T(S_A)}{T(S_{OPT})} \leq 4\sqrt{N}$ under triangle inequality and if the initial solution was generated randomly (Johnson & McGeoch, 1995; Prodhon & Prins, 2016).

The examples above illustrate simple moves on a single tour. Alternatively, a move can affect more than one tour. Fig. 14 shows an example of a 2-Opt move for a VRP with two vehicles.

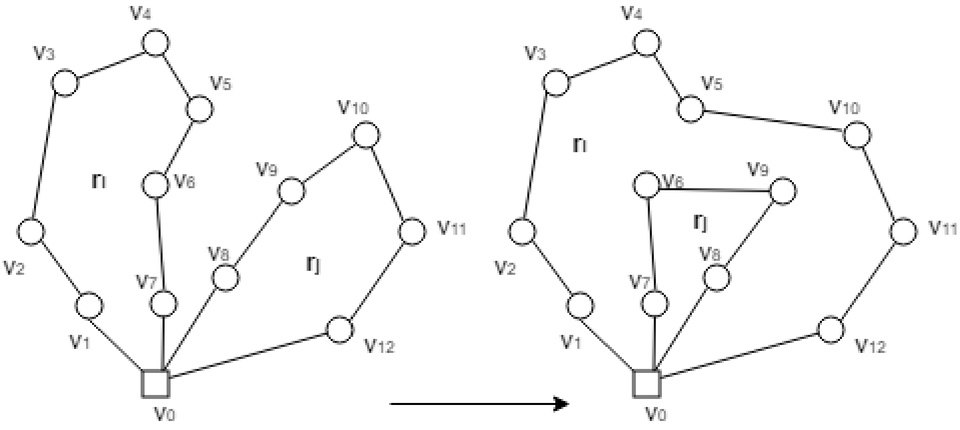


Figure 14. 2-Opt move for a VRP with two vehicles.

Given two routes r_i and r_j, we replaced edges $e_{5,6}$ in r_i and $e_{9,10}$ in r_j with new edges $e_{5,10}$ and $e_{6,9}$ respectively. Here, the number of vehicles required must be equal to the value X picked for X-Opt: a 3-Opt algorithm can perform a similar move with three tours at a time, etc. Nevertheless, as we mentioned above, X-Opt algorithms with $X > 3$ are rarely used, so this approach is not scalable, although the algorithm can also be implemented to affect two or three tours at a time within a larger number of tours in the problem, so that only part of all tours are affected in each iteration. In fact, several versions of X-Opt can be combined in one algorithm that features several different types of moves. In this case, each move type has its own neighborhood in each iteration, and the best move across all types must be found (Prodhon & Prins, 2016).

Johnson & McGeoch (1995) run tests on 3-Opt and 2-Opt comparing them with the results obtained by Clarke-Wright and Nearest Neighbor construction heuristics with no tour improvement applied. They use Nearest Neighbor, Randomized Nearest Neighbor (Nearest Neighbor in which the probability of adding the nearest vertex is $2/3$ and the second nearest vertex – $1/3$) and Clarke-Wright to construct the initial solution. Their results show that by using 2- and 3-Opt one can achieve significantly better results: solution obtained by 3-Opt are on average only 3% worse than the optimal solution, 2-Opt – 4.5% worse, and these figures vary by fractions of a percent for the instances from 10^2 and 10^6 vertices. As to the choice of the construction heuristic used to generate the starting solution, they conclude that Clarke-Wright, in the implementation they use, performs surprisingly poorly, while their implementation of Nearest Neighbor shows much better results, and this holds for both 2- and 3-Opt. These results were obtained on the problems with 10^3 vertices.

4.5.2 US (Stringing and Unstringing)

US was developed by Gendreau et al. (1992) and presented in combination with GENI. The terms 'stringing' and 'unstringing' refer to adding a vertex to a tour and removing a vertex from a tour, respectively. The original version works on a single (TSP) tour. In Gendreau et al. (1994), the authors use US for solving a VRP by applying it to the giant tour produces by GENI (see section 4.4.5) before splitting it into separate tours. Theoretically, it can be applied to each single tour after the splitting, but only if a tour has at least three vertices apart from the depot. Since this cannot always be guaranteed, in the description below we assume it is applied to the giant tour produced by GENI or any similar procedure before the splitting. However, if in the given solution all tours have three or more non-depot vertices, it can be applied to each of these tours instead.

Since US and GENI shared the same notions we refer the reader to section 4.4.5. Briefly:

- $N_p(v_x)$ is called a *p-neighborhood of* v_x and is defined as a set of p closest vertices to v_x (in terms of the cost function's value) that are on the same tour
- v_i, v_j, v_k, v_l are vertices on the given tour r such that for the given orientation of r, v_k lies on the path from v_j to v_i and v_l – on the path from v_i to v_j $(v_i, v_j, v_k, v_l \in V)$

There are two types of unstringing possible. For *type I*, let us consider $v_j \in N_p(v_{i+1})$ and, for a given tour orientation, $v_k \in N_p(v_{i-1})$ as a vertex on the path $(v_{i+1} \dots v_{j-1})$. Then the tour is altered as showed in fig.15:

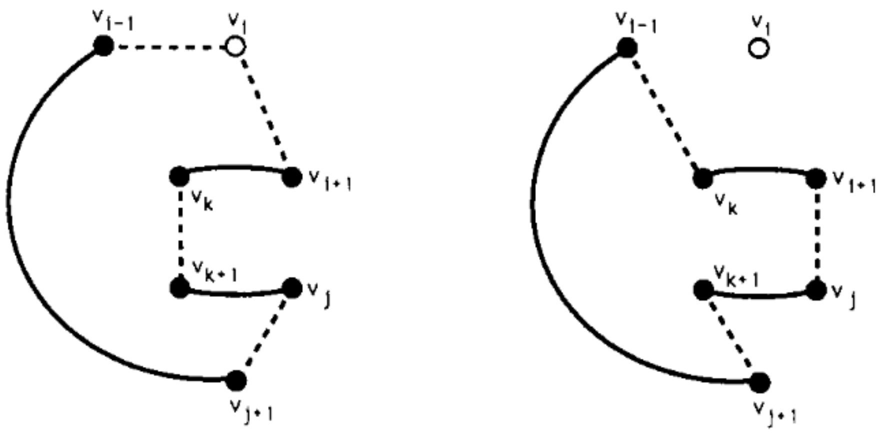


Figure 15. US Type I operation. The left part shows the original graph, the right one – the result; edges that are altered in the process are shown by dashed lines; v_i is unstrung; (Gendreau et al., 1992)

For *type II* operation, let us consider $v_j \in N_p(v_{i+1})$ and, for a given tour orientation, $v_k \in N_p(v_{i-1})$ as a vertex on the path $(v_{j+1} \dots v_{i-2})$ and $v_l \in N_p(v_{k-1})$ as a vertex on the path $(v_j \dots v_{k-1})$. Then the tour is altered as shows in fig. 16:

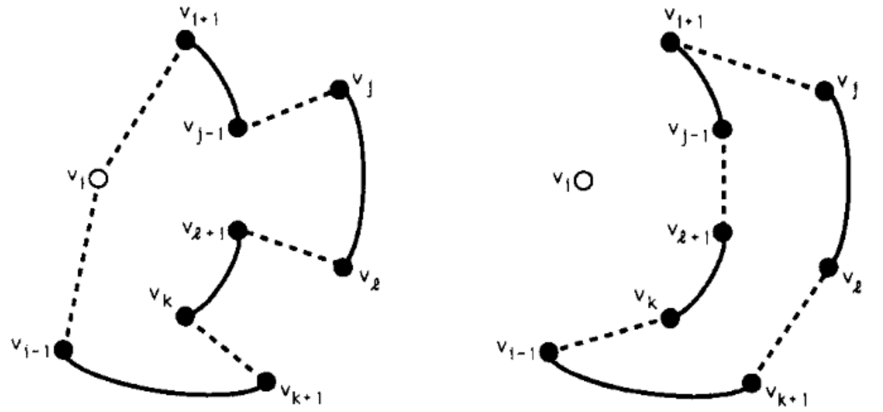


Figure 16. US Type II operation. The left part shows the original graph, the right one – the result; edges that are altered in the process are shown by dashed lines; v_i is unstrung; (Gendreau et al., 1992)

The two stringing procedures are identical to the two types of vertex insertion in GENI (see section 4.4.5), and the unstringing are essentially the reverse versions of those. Now, the entire US algorithm proceeds as follows: starting with an initial tour, generated by either GENI or any other tour construction heuristic, it considers each of the two types of unstringing for each of the two possible tour orientations and accepts the most improving of these moves. The algorithm stops once all the vertices have been tested. The authors run a series of test on problems with Euclidean distances and from 100 to 500 vertices and conclude that GENI combined with US ('GENIUS') consistently outperforms several combinations of tour construction and tour improvement heuristics, such as Clarke-Wright's saving method combined with 2- or 3-Opt tour improvement heuristic. No results are reported, however, for the problem with non-Euclidean distances (costs).

4.6 Metaheuristics

Metaheuristics typically control the behavior of the underlying heuristic(s). Most tour improvement heuristics employ *Local Search*. The *search space* is defined by the set of possible *solutions*. The *adjacent (neighboring) solutions* constitute a *neighborhood* and have one-move difference between each other (the definition of a move depends on the employed heuristic), e.g. one 2-Opt move. An adjacent solution is produced by applying one iteration of the tour improvement heuristic to the current solution. The collection of solutions produced by applying all possible moves to the current (*incumbent)* solution constitutes the that solution's *neighborhood.*

A metaheuristic can guide Local Search to avoid getting stuck in local optima. There are several ways to do it: restart the heuristic from different initial solutions, memorize the solutions that have already been tried and should be avoided, allow the heuristic to accept uphill solutions with a certain probability, etc. What is important to understand, many metaheuristics use one or more heuristics and orchestrate them throughout the procedure, thus metaheuristics are not strictly alternatives to heuristics but rather have a hierarchical relationship with those.

The CVRP, at least in our formulation, is a *minimization* problem, meaning that the objective function seeks to find the solution with the minimal possible objective's value. Therefore, by saying 'local optima' we imply 'local minima'. However, in describing the general principle of the given metaheuristic, we would like to emphasize that it can as well be used for *maximization* problem, so we use 'local optimum' as a more general term. If one solution has lower cost than another, we call the former *'downhill'* and the latter one – '*uphill'*, when compared with each other. Informally, 'downhill' means 'better' and 'uphill' means 'worse', in terms of the objective's value, in a minimization problem. We call a set of solutions a '*plateau'* if they have equal costs.

Another important thing to consider is that most metaheuristics are problem independent. They may be designed and tested for a certain problem or a class of problems, but can, with some adjustment, be used for other problems as well, although some methods are known to perform better for certain problems. Specifically, in the case of Vehicle Routing Problems, popular heuristics like Genetic algorithms and Ant Colony Optimization have proved to be less efficient (Prodhon & Prins, 2016), and are superseded by other metaheuristics, three of which we discuss in the following sections: Simulated Annealing, Tabu Search and Guided Local Search.

4.6.1 Simulated Annealing

This metaheuristic is named after the process of annealing used in metallurgy, a technique involving heating and controlled cooling of a metal to change its properties. Similarly, in Simulated Annealing (let us call it SA for brevity), there is a dynamic *'temperature'* parameter that decreases with time. The higher the temperature, the greater the probability of accepting uphill solutions along the solution space. Accepting worse solutions allows the procedure to escape local optima and eventually find the optimum solution (Kirkpatrick, Gelatt, & Vecchi, 1983).

At each iteration, the search examines one randomly picked solution in the neighborhood. It compares it with the incumbent solution and decides whether to accept or reject the new solution with a probability based on the current temperature value. When the temperature of the system decreases to zero, the procedure turns into a greedy search with a randomized step: only downhill solutions are accepted, and the algorithm loses the ability to jump out of local minima. However, by this time we expect it to have found either the global minimum or a solution sufficiently close to it in terms of its cost. The performance of the algorithm depends on the temperature function of time that we choose. More specifically, in each iteration the probability of accepting an uphill solution S' is defined by the probability function $P(S, S', \tau)$ where τ is the current temperature and S is the incumbent solution. The lower the temperature, the lower the probability, and the other way around:

$$\lim_{\tau \to 0} P(S, S', \tau) = 0; \quad \lim_{\tau \to +\infty} P(S, S', \tau) = 1 \qquad (24)$$

The acceptance probability for downhill moves can simply be set to 1, but more sophisticated implementations make it an inverse function of the difference between the compared solution: the larger the difference, the lower the probability (Vidal et al., 2013).

For any given finite problem, the probability that the SA algorithm terminates with a global optimum approaches 100% as the annealing schedule is extended, but it will take at least as much time as any exact method (Prodhon & Prins, 2016). An efficient schedule is the one that produces results sufficiently close to the global optimum but finishes the search in a reasonably short period of time. Implementation of SA thus requires defining three functions: the *acceptance probability function*, the temperature function of time (*annealing schedule*) and the *initial temperature* τ_0. Last but not least, the local search procedure must be chosen; in the case of the VRP, - one of the tour improvement heuristics (see section 4.5). For the acceptance probability function, it is common (Prodhon & Prins, 2016) to use

$$P(S, S', \tau) = \begin{cases} \exp\left(-\frac{(T(S') - T(S))}{\tau}\right), & \forall\, T(S') > T(S) \\ 1, & \forall\, T(S') \le T(S) \end{cases} \qquad (25)$$

Here, a downhill move is always accepted, and the probability of accepting an uphill move is essentially the Boltzmann's probability factor from the field of statistical mechanics (Kirkpatrick et al., 1983). The optimality of the resulting solution does not, generally, depend on the quality of the initial solution, because of the high scatter of the moves under high temperature values at the start of the procedure (Prodhon & Prins, 2016).

For the annealing schedule, a common approach is to reduce the temperature by a constant factor, e.g. 0.95, after each iteration t_x. Alternatively, a maximum number of iterations t_{max} can be provided, after which the temperature is set to zero, and the search stops:

$$\tau = \begin{cases} \frac{\tau_0}{t}, & t_x \le t_{max} \\ 0, & t_x > t_{max} \end{cases} \qquad (26)$$

Alfa, Heragu & Chen (1991) use SA with the 3-Opt heuristic to solve a CVRP. They start by constructing a single *giant tour* that includes all destinations, split it according to the $split$ procedure that we defined in section 4.4.6, and then optimize the resulting routes via 3-Opt. Their version of 3-Opt is different to the one we described in section 4.5.1: they pick three edges from any of the tours in R, at random. It can be three edges of the same tour, or three edges in three different tours, or two edges in one tour and another one in another tour. The authors use the following configuration for SA:

- For the acceptance probability function, they use formula (23) discussed above
- Initial temperature is set to some $\tau_0 \in \mathbb{Z}$
- For the cooling rate, they use a constant factor α_{cool}, such that $0 < \alpha_{cool} < 1$
- This cooling rate is applied every $t_{cool} \in \mathbb{Z}$ iterations
- The number of iterations is limited to $t_{max} \in \mathbb{Z}$, after which the procedure stops

$\tau_0, \alpha_{cool}, t_{cool}\ t_{max}$ are parameters. The algorithm proceeds as follows:

Step 1:
Select the values for all four parameters above. Initialize S^* for storing the best solution found so far, $S_{current}$ for the incumbent solution, temperature value $\tau = \tau_0$, $t_{total} = 1$ to keep the iteration count and $t_\tau = 1$ to store the number of iterations done with the current value of τ. Importantly, in their algorithm, t is only incremented when τ is changed, that is, every t_{cool} iterations, thus essentially keeping the number of different values of τ used, rather than the total iteration count. The latter can then be calculated as $t_{cool} * t_{total}$.

Step 2:
Construct a single *(TSP)* tour $r_{init} = [e_{0i}, \dots, e_{n0}]$ with *all* vertices in V (relax the capacity and duration constraints but ensure that it is a valid TSP tour otherwise: no subtours; v_0 is the start and end vertex of the tour). The authors use the Nearest Neighbor heuristic, but any other tour construction method can be used instead.

Step 3:
Run $split(r_{init}, Q_{max}, T_{max})$ that returns a solution S_o with a route set R, $|R| \leq K_{max}$; S_o may not be feasible. Set $S^* = S_o$ and $S_{current} = S_o$.

Step 4:
Generate a 3-Opt neighborhood for $S_{current}$ and evaluate their costs. Select the best of these solutions $S_{candidate}$.

Step 5:
If $T(S_{candidate}) < T(S^*)$, set $S^* = S_{current} = S_{candidate}$ and go to step 7; otherwise, go to step 6.

Step 6:
Generate a random number x in $[0.0, 1.0]$. If $P(T(S_{current}), T(S_{candidate}), \tau) > x$, go to Step 6; otherwise, set $S_{current} = S_{candidate}$ and go step 5.

Step 7:
If $t_\tau \geq t_{cool}$, proceed to step 8; otherwise, set $t_\tau = t_\tau + 1$ and go to step 1.

Step 8:
If $t_{total} \geq t_{max}$, stop; otherwise, set $t_{total} = t_{total} + 1, t_\tau = t_\tau + 1, \tau = \tau * \alpha_{cool}$ and go to step 1.

The authors run several tests, each time with the same:

- $t_{cool} = 80 * |V|$
- $\alpha_{cool} = 0.95$

- $\tau_0 = 200$

but with different t_{max} values: they use 15, 20 and 40. Comparing their results with those produced by other researchers for the same VRP instances, they conclude that their method returns results at least as good as several other methods including 3-Opt alone, Lagrangian Relaxation and Integer Linear Programming, but takes more time on average. They also state that augmenting their method with a way for optimizing the initial solution could improve the results and reduce the CPU time used (Alfa, Heragu, & Chen, 1991).

4.6.2 Tabu Search

Tabu Search (for brevity, let us call it 'TS') attempts to prevent getting stuck in local optima or repeating the same moves by memorizing their locations within the search space. It stores these in a *tabu list* in memory (usually as a *queue)*. The entries are stored for a number of iterations called *tabu tenure*. Separately, it also stores the best solution currently encountered to return it as the outcome of the search at the end (Cordeau et al., 2002).

For example, let us imagine that in a CVRP, the destination v_i has been moved from route r_j to route r_k. To prevent cycling, a new entry (v_i, i, j) should be added to the tabu list and stored there for the duration of the tabu tenure. Entries in a tabu list can also store other information, such as how many times each destination has been swapped between the routes; when a limit on the number of swaps for the given is reached, the metaheuristic may attempt to alter the search procedure: allow more complex moves, restart the search from another initial solution, etc. Alternatively, the other way around, the destinations that have not yet been swapped a single time may be forced to swap (Prodhon & Prins, 2016).

Different variants of TS can implement different *forbidding* and *freeing strategies.* A *forbidding strategy* defines what solutions and when to add to the tabu list. A *freeing strategy* tells which solutions, and when, to remove from the list. These two strategies can themselves be managed and altered by a top-level *short-term* strategy. These strategies depend on the tour improvement heuristic used. To make examples simple and concise, let us assume that our heuristic is to *swap* two vertices between the two tours that make up the same solution, e.g. given a solution S that consists of two routes

$$r_1 = [e_{01}, e_{14}, e_{43}, e_{30}] \text{ and } r_2 = [e_{05}, e_{57}, e_{76}, e_{60}],$$

we swap vertices $v_1$and v_6, obtaining a neighbor solution S' consisting of routes

$$r_1' = [e_{06}, e_{64}, e_{43}, e_{30}] \text{ and } r_2' = [e_{05}, e_{57}, e_{71}, e_{10}]$$

The forbidding strategy is defined by its *tabu criteria*. The most common practice is to ban repeating the same move for a certain number of iterations, so that the search does not go in cycles. For example, the swap (v_6, v_1) that we made earlier can be disallowed for three iterations. Another common example is forbidding the moves that resulted in an uphill solution. For example, if after our (v_6, v_1) swap we performed a (v_5, v_3) swap, and the resulting solution has higher overall travel duration, the latter swap can be considered unpromising and put in the tabu list.

Imagine that the best solution in the current neighborhood is in the tabu list, moreover that is the only improving solution in the current neighborhood, what should be done then? This is where the freeing strategy and its *aspiration criteria* come into play. Under certain conditions, a move can be removed from the tabu list before the end of its tenure. The exact condition depends on the implementation. One example would be: if a given move yields a solution better than the current best solution, that move is released from the tabu list and the move is accepted, otherwise the best of the non-tabu moves is selected even if it is an uphill move.

Unlike exact methods, metaheuristics like TS never converge unless a stopping criterion is provided. Here are some examples of the stopping criteria often used in TS (Glover, 1990; Prodhon & Prins, 2016):

- A maximum number of iterations
- Number of iterations since a downhill solution was last found
- Absence of feasible moves in the incumbent solution's neighborhood
- Objective's value for the incumbent solution (e.g. less than a certain threshold)

The use of tabu lists is an intuitive and effective solution but it also means that tabu search can consume a lot of memory. The recency-based tabu list (disallowing recently performed moves) can be implemented as a *queue* of a fixed length n, where n is the tabu tenure. This implements the short-term memory. The intermediate-term memory is represented by the rules that make the search favor downhill moves. Finally, long-term memory can be implemented by the diversification rules that restart the search from a previously unexplored region of the solution space if it got stuck or reached a dead-end (Glover, 1990; Prodhon & Prins, 2016).

Gendreau, Hertz & Laporte (1994) present a TS algorithm for solving the CVRP which they call Taburoute. They use the GENIUS heuristic, i.e. a combination of the GENI tour construction heuristic (see section 4.4.5) combined with the US tour improvement procedure (see section 4.5.2). The TS orchestrates those and guides them through the solution space. The method attempts to improve the solution moving a given vertex from one route to another. All these combinations for each given vertex form a solution's neighborhood. The algorithm keeps a list W of all vertices that are allowed to be moved to another route. Initially, all $v_i \in V$ are on this list and all moves are allowed.

The authors use *penalties,* a notion that is not innate to TS, but is common in other metaheuristics. For example, penalizing undesirable solutions is a core practice in Guided Local Search that we will discuss in the next section. Therefore, we can state that Gendreau et al. (1994) hybridize their TS algorithm by using an *augmented objective function.* Let F_1 denote the (original) objective function associated with *any feasible* solution:

$$F_1(S) = \sum_{k=1}^{K_{max}} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \qquad (27)$$

Now, let F_2 denote the augmented objective function associated with *any* solution:

$$F_2(S) = F_1(S) + \alpha \sum_{k=1}^{K_{max}} \left[\left(\sum_{v_i \in r_i} q_i \right) - Q_{max} \right]^+ + \beta \sum_{k=1}^{K_{max}} \left[\left(\sum_{e_{ij} \in r_i} c_{ij} \right) - T_{max} \right]^+ \qquad (28)$$

where $[x]^+$ is $max(x, 0)$, $\alpha > 0$ and $\beta > 0$. If solution S is feasible, $F_1(S) = F_2(S)$, otherwise two penalty costs are added: one for excessive duration and one for excessive capacity. Now, the SEARCH algorithm proceeds as follows:

Step 0:
Initiate the following variables:

- list $W = [v_i, \dots v_n]$
- S for the incumbent solution

- F_1* and F_2* to store the best currently encountered $F_1(S)$ and $F_2(S)$
- S* to store the best currently known feasible solution
- $\tilde{S}$ to store the best currently known solution
- t to store the iteration count

Define the search strategy (forbidding and freeing strategies):

- u – number of vertices from W to consider in each iteration
- p_1 – neighborhood size used in GENIUS
- p_2 - number of p-neighbors a candidate v_i must have in route r_i to be inserted there
- θ_{min}, θ_{max} – upper and lower bounds on the tabu tenure
- h – the frequency at which changing β and α is considered
- t_{max} – the number of consecutive iterations with no downhill solution found after which the search will converge
- g – a scaling factor used to penalize an uphill solution (see step 2 below); *the authors generally recommend using $g \in [0.005, 0.2]$. Lower values do not produce enough diversification in the search, higher values generally decrease the optimality of the results*
- Δ_{max} – the largest $|F_2(S_{i+1}) - F_2(S_i)|$ for any two successive solutions S_i, S_{i+1} examined
- $timesMoved(v_i)$ – the number of times a given vertex has been moved

Step 1:
Generate an initial solution S_0 using GENI and set $S = S_0$. Randomly select u vertices from W.

Step 2:
For each of u randomly selected vertices from W: consider all possible moves from its current route r_i to another route r_j, where r_j must either contain at least one of the p_1 nearest neighbors of v_i or be empty. Insert v_i into r_j according to the GENI procedure and evaluate the resulting solution S'. If the move is tabu, it is disregarded unless either of the two following conditions apply:

- S' is feasible and $F_1(S') < F_1$*
- S' is infeasible and $F_2(S') < F_2$*

If the move has been disregarded, proceed with the next vertex; otherwise, set $T(S')$ equal to:

- $F_2(S')$ if $F_2(S') < F_2(S)$
- $F_2(S') + \Delta_{max}\sqrt{|R| * g * timesMoved(v_i)/t}$ otherwise

Step 3:
Find the best move among the ones performed in Step 2 (except the disregarded moves). The solution S' produced by this move is accepted unless all three of these conditions are satisfied:

1) Incumbent solution S is feasible
2) $F_2(S') > F_2(S)$
3) US was not used in the previous iteration

If they are satisfied, then the current candidate solution, despite being the best move in the current neighborhood, is worse than the incumbent solution. In such case, the new solution is produced by applying the US tour improvement heuristic to the incumbent solution S instead.

Step 4:
If US was not applied in the previous step and the candidate move was accepted, that move is added to the tabu list for θ iterations, where θ is a randomly selected integer in the

interval $[\theta_{min}, \theta_{max}]$. *The authors postulate that a variable tabu tenure improves the results. They use* $\theta_{min} = 5, \theta_{max} = 10$.

Step 5:
If t is a multiple of h, update α as follows:

- If in *all* previous iterations where t was a multiple of h the solution did not violate the capacity constraint, set $\alpha = \alpha/2$
- If in *all* previous iterations where t was a multiple of h the solution did violate the capacity constraint, set $\alpha = 2\alpha$

Apply the same to β with respect to the duration constraint.

Step 6:
If F_1 and F_2 have not been updated for t_{max} iterations, stop, otherwise go to step 1.

This was the description of the SEARCH procedure. Now, the main overall algorithm TABUROUTE works in the following way: first, n initial solutions are generated using GENI (n is a parameter). Then, SEARCH is applied to each of them for a certain number of iterations t_x with some arguments $p_1$and p_2, and the most promising solution is chosen as S_0 for SEARCH. The number of initial solutions and iterations are input parameters defined by the user or implementor. Finally, SEARCH is applied with that starting solution twice, each time with different p_1 and p_2 values (hence, there are three pairs of p_1 and p_2 used throughout the TABUSEARCH altogether).

Step 0:
Set $\alpha = \beta = 1$ and $F_1{}^* = \infty$.

Step 1:
For t_x iterations do:

- Construct an initial solution S_0 using GENIUS and update $F_1{}^*$, $F_2{}^*$, S^* and $\tilde{S}^*$ accordingly
- Call SEARCH with the first pair of arguments p_1 and p_2
- If the resulting $F_1{}^* < \infty$, set S = S*; otherwise set S = $\tilde{S}^*$

Step 2:
Call SEARCH with other p_1 and p_2; set $S = S^*$; otherwise set $S = \tilde{S}^*$

Step 3:
Call SEARCH with the third pair of p_1 and p_2. If $F_1{}^* < \infty$, return S*, otherwise return NULL (no solution found). Stop the algorithm.

Gendreau et al. (1994) test their method on fourteen different problem instances that contain between 50 and 200 cities and compare their results with a number of different methods, including two other versions of Tabu Search, a version of Simulated Annealing and Clarke-Wright savings method on its own. Their results indicate that all simple heuristics like Clarke-Wright, are dominated by metaheuristics that use in their comparison, specifically, Simulated Annealing and Tabu Search. They report that among the latter, their Taburoute algorithm results in solutions that are closer to optimality, but subsequently discuss that those results should be taken with caution for a number of reasons (for a detailed explanation, see the paper).

4.6.3 Guided Local Search

This section is based on Voudouris & Tsang (2001) and Prodhon & Prins (2016). As the name suggests, Guided Local Search (GLS) is a metaheuristic that sits on top of Local Search and guides it to escape local optima. It does so by applying a *penalty* to a solution if it exhibits certain *features*. The choice of features used for penalization depends on the move(s) used. Edges (shortest paths between destinations) are commonly used for GLS (Arnold & Sörensen, 2019; Kilby & Prosser, 2002; Prodhon & Prins, 2016; Voudouris & Tsang, 2001). In fact, the authors of GLS, Voudouris & Tsang (2001) argue that edges are 'ideal GLS features', so we make the same choice in this section. By penalizing a solution, GLS changes the solution space: a penalized local optimum may cease to be one, essentially 'rising' above the neighbor solutions (in a minimization problem). This approach is an alternative to, for example, random restarts or acceptance probability. Penalizing is implemented by assigning a given solution S a new cost $T'(S)$ defined by an *augmented objective function* $F_{pen}(S)$:

$$F_{pen}(S) = T(S) + \lambda \sum_{i=1}^{n} \sum_{j=1}^{n} (p_{ij} \ * \ \varepsilon_{ij}(S)), \quad i \neq j \qquad (29)$$

where:

- p_{ij} is the current penalty given to S for having edge e_{ij} in it
- $\varepsilon_{ij}(S)$ is a binary variable equal to 1 if e_{ij} is present in S and 0 otherwise
- λ is a parameter

Higher values of λ lead to *diversification* of the search: plateaus and basins are searched more coarsely; lower values *intensify* the search: the search employs a finer step.

Now, the use of penalties is not unique to GLS, but the way GLS chooses which features to penalize is what makes it truly efficient (Voudouris & Tsang, 2001). It uses a notion of *utility* of a feature; the higher the feature's (let us take an edge as a feature) utility $u_{ij}(S)$, the more prominent its role in the solution. It is defined as follows:

$$u_{ij}(S) = \varepsilon_{ij}(S) \ * \ \frac{c_{ij}}{1 + p_{ij}} \qquad (30)$$

The higher the edge cost, the greater the utility of penalizing it. At the same time, the higher the edge's penalty, the lower the utility. Most often, the feature with the highest utility is penalized. If there are more than one such features, all of them may be penalized, but the implementation is flexible. When it comes to the amount of penalty, the most common approach (Arnold & Sörensen, 2019; Kilby & Prosser, 2002; Voudouris & Tsang, 2001) seems to be to increment p_{ij} by 1 every time e_{ij} is penalized; the penalty then essentially represents the number of times the feature has been penalized. This value is then scaled by λ to reach the desired level of effect on the objective function. The value of λ is problem-dependent and experimentation is encouraged although Voudouris & Tsang (2001) argue that empirical results suggest that λ for the given problem can be computed the following way:

0. Let us define a function $num\_edges(R_s) = \sum_{k=1}^{K_{max}} |r_k|$, where R_s is the set of routes in solution S; then $num\_edges(R_s)$ returns the number of edges in solution S
1. Run the algorithm with an arbitrary value λ, e.g. $\lambda = 1$
2. Introduce a parameter α_{gls}
3. At the first encountered local minimum S, set $\lambda = \frac{\alpha_{gls} * T'(S)}{num\_edges(R_s)}$

The authors believe that α_{gls} is relatively instance-independent and should be in $[{}^{1}/_{8}, {}^{1}/_{2}]$ for routing problems like TSP and VRP.

Just as with other metaheuristics, a termination condition must be specified, most commonly it is a certain number of iterations t_{max}. This part is also problem-dependent, and the best value should be found empirically. Below is a general step-by-step description of GLS that uses edges as features:

Step 0:
Generate an initial solution S_0 randomly or using a tour construction heuristic.

Step 1:
Initiate:

- p_{ij} for each e_{ij} and set $p_{ij} = 0$ for all e_{ij}
- S* to store the best solution met so far

Step 2:
Repeat until the stopping condition is met:
Perform Local Search using the objective function $F_{pen}(S)$ until a local minimum S^*_{local} is reached.
Once it has been reached:

- set S* = S^*_{local}
- compute u_{ij} for each e_{ij} in S^*_{local}
- set $p_{ij} = p_{ij} + 1$ for the feature with the highest utility

Step 3:
The stopping condition has been met. Return T(S*).

Finally, a note on the underlying heuristics. The authors, Voudouris & Tsang (2001) state that GLS is not, generally, very sensitive to the optimality of the initial solution given that sufficient time is allocated for the search. That means that simple tour construction heuristics like Nearest Neighbor can be used. In fact, the starting solution can be generated randomly. For the tour improvement heuristics, the authors' view is similar: they believe that more sophisticated algorithms provide less space for GLS itself to act and can sometimes even reduce the optimality of the result. They recommend 2-Opt as simple method that performs well for the routing problems.

4.7 Conclusion

There are several exact methods for solving Vehicle Routing Problems but since the VRP is NP-hard, heuristics are still widely used to solve large instances in an acceptable span of time. Some real-world applications of the VRP may be sensitive enough to the optimality of the found solution to favor long execution times of exact algorithms over the speed of (meta-)heuristics. We believe that municipal waste collection is not one of those. The traditional fixed-schedule and fixed-routes approach is entirely based on approximations, and the consequences of route and schedule suboptimality are, generally, tolerable. This is to say that haulers are likely to prefer methods that arrive at near-optimal solution in an acceptable period of time.

Prodhon & Prins (2016) argue that local search procedures are crucial for solving a VRP, and the metaheuristics that employ local search solve routing problems more efficiently than those that do not use local search, for example, Genetic Algorithms and Ant Colony Optimization. Among those that do, Tabu Search generally outperforms Simulated Annealing and since the early 1990s, when the former became widely used, it has been considered the best-performing metaheuristic (or, rather, a metaheuristic family). However, in the last decade, they argue, it was largely outperformed by other metaheuristics, such as Guided Local Search.

Cordeau et al. (2002), state that Tabu Search generally performs better than Simulated Annealing or any other metaheuristic, but their paper was published before Guide Local Search was invented. Arguments like this, dating almost twenty years back, should, of course, be taken with caution, because each family of metaheuristics gradually evolves over time as new variations appear, and the balance between them may shift. Same applied to Voudouris & Tsang (2001), the inventors of Guided Local Search, who report having tested their method (using Lin-Kernigan's variable-opt as a tour improvement heuristic) on the problems from the TSPLIB and found that it had outperformed certain implementations of Simulated Annealing and Tabu Search.

Finally, Vidal et al. (2013) do a thorough analysis using 20 different metaheuristics and their combinations, and testing them on a number of VRP instances from the VRPLIB (Vidal et al., 2013). The Guided Local Search methods they include in their tests (they have two variations that are combinations of Guided Local Search with other metaheuristics) perform better on average than Simulated Annealing, pure Tabu Search and various implementations of Genetic Algorithms, although these Guided Local Search methods still give worse results than a few complex hybrid metaheuristics. The runtimes for those, however, are notably longer, and they are harder to implement. In terms of implementation, Cordeau et al. (2002) point out that the most efficient heuristics are not always the most widely used ones. In order to be implemented, a heuristic must be described in sufficient detail. In fact, heuristics that are easy to implement (most likely because of a clear and, possible, simple design) are more likely to be widely used than their complex peers. Cordeau et al. (2002) attribute the wide use of the Clarke-Wright savings method to this fact. Prodhon & Prins (2016) draw the same conclusion with regard to metaheuristics, stating that despite hybridization (combining several metaheuristics) appears to be a trend, the test runs show that the performance improvement they promise is most of the time not as great as the increase in runtimes and implementation complexity that the hybridization ensues. From the above discussion, we conclude that Guide Local Search appears to be best-performing method among regular (non-hybridized) metaheuristics, followed by Tabu Search.

5 Implementation

In this chapter, we present a minimalistic software application for receiving, pre-processing and storing sensor data, as well as a QGIS 3 plugin for scheduling, routing and visualizing the results.

5.1 The Things Network application

The sensor that we present in section 3.1 is connected to a The Things Network (TTN) gateway via LoRa (wirelessly). To do so, we use the TTN account of GIS Specialisten (since the sensor and the gateway belong to them) but to replicate the procedure, one could create a free account on the TTN's website (The Things Industries, 2019). An account holder has one or more *applications*. An application groups one or more sensors into a single project that has its unique *Application ID* and *Access Key* (used as a password). A sensor sends its *payload* to the gateway via LoRa. The gateway is a stationary device that is connected to the Ethernet. One gateway can receive data from multiple sensors located in the range of up to 1 km in an urban environment. A gateway then sends this data to the TTN network server over the Ethernet cable although it can also use cellular connection. The architecture overview is presented in fig.17:

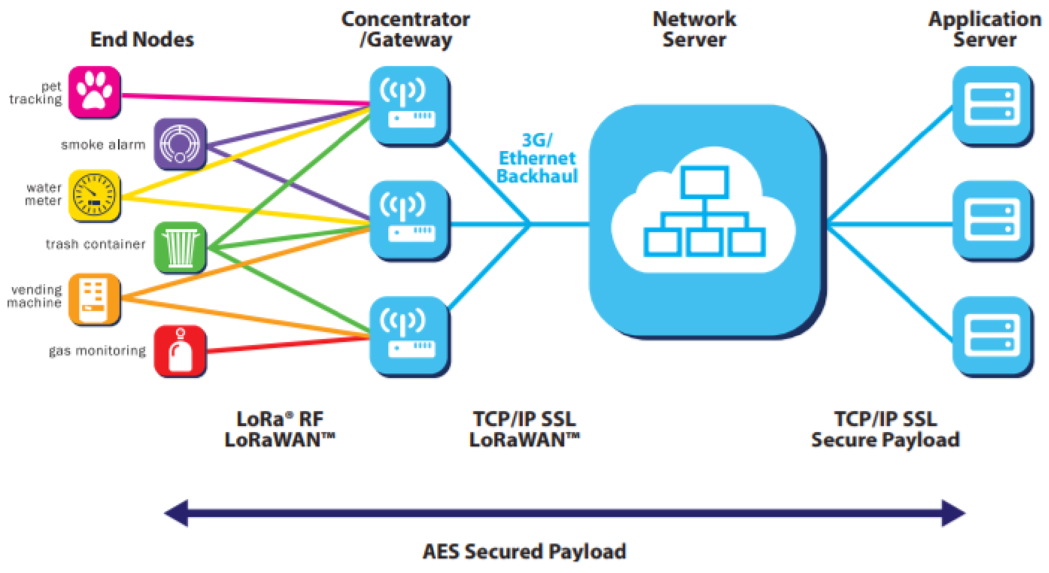
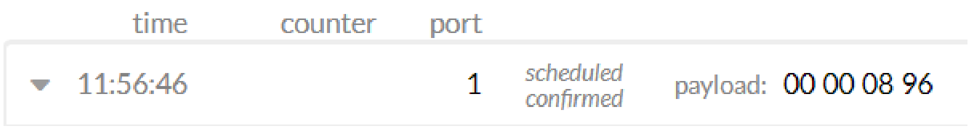


Figure 17. The general TTN architecture.

The data sent by the sensors can be viewed on the TTN console in the account:



time	counter	port		
11:56:46		1	*scheduled confirmed*	payload: 00 00 08 96

Figure 18. An example of a data record sent by a sensor, as displayed on the TTN console.

The CNDingTek waste container sensor used in this study sends a 4-byte uplink that is represented in the TTN as 4 pairs of digits, or 4 hexadecimal values. The sensor reports on 5 different values in the following order:

- Whether it is full (1/0)
- Whether it caught fire (1/0)
- Whether it is positioned upright (1/0)
- Whether the battery is empty (1/0)

- The distance between the sensor and current garbage level (mm, max = 2000)

By default, five bytes would be required for such a message. However, the first four measurement will never actually need a whole byte and can in fact always be represented by only one hexadecimal digit (decimal 0 through 15). Hence it is possible to logically put two values into one byte. For example, if the first byte is '11' in hexadecimal, it represents a decimal number '16', but if we read it as a string of two characters instead, it can be interpreted as two decimal numbers: '1' and '1', which means 'yes, it is full' and 'yes, it is on fire'. This structure is summarized in table 4:

Table 4. Structure of the sensor's payload ('X' stands for 'any value').

Meaning	*Message pattern*
full (true)	1XXXXXXX
on fire (true)	X1XXXXXX
not upright (true)	XX1XXXXX
empty battery (true)	XXX1XXXX
remaining distance	XXXX1234

The last value, the remaining distance, can vary from 0 to 2000 meters (the maximum distance the sensor can scan), which will mean the number of millimeters left to the top of the container. A single byte can only represent the decimal 0 through 255, so again, the optimal logic would be to send four different hexadecimal values and read them as a string representing a decimal number. In table 4, the last four number are technically two hexadecimal values: '12' and '34', but read together, they represent '1234' (decimal) which tells that there is 1.234 m left free in the container.

Using the application's ID and Access Key, any client (in respect to the TTN network server) application can *subscribe* to the *uplinks* from the sensors that belong to that application. A subscription here means that the payloads (uplinks) from the sensors will be received automatically and in near-real time: a delay of up to a few seconds can be expected. The frameworks on which such a client can be built are listed on the TTN website (The Things Industries, 2019) and include Python, Node.js, Node Red, Go and Java. We chose *Node.js* for its asynchronous nature that allows to accelerate the processing. For storing the data, we use *PostgreSQL* with the PostGIS extension. Fig.19 presents the data model:

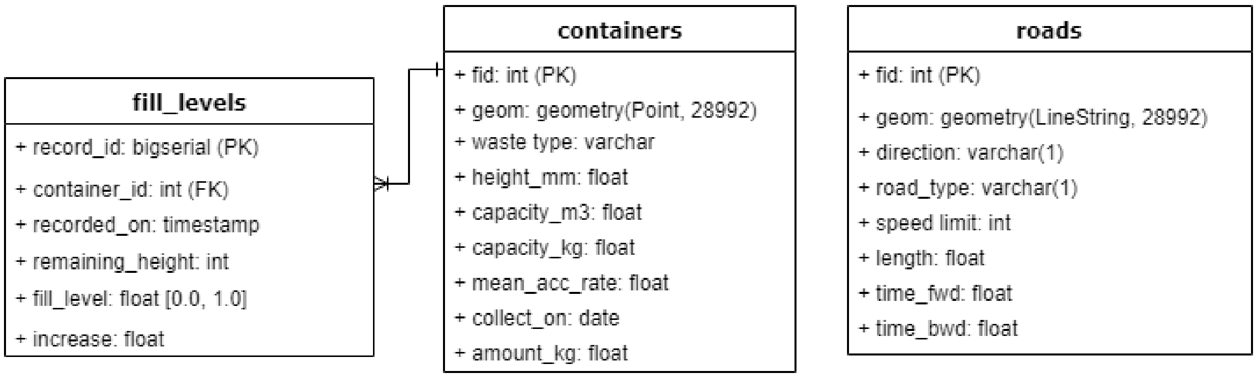


Figure 19. PostgreSQL Data model.

An uplink from the sensor is decoded in the TTN client. It sends two output values: the id of the container / sensor (must correspond to one of the values in the 'fid' column of relation 'containers') and the 'remaining height', i.e. the distance in mm reported by the sensor. A new record is then inserted into the 'fill_levels' table:
- 'record_id' is an autoincremented field
- 'container_id' is received from the TTN

- 'recorded_on'is received from the TTN
- 'remaining height' is received from the TTN
- 'fill_level' is computed automatically using a trigger function
- 'increase' is computed automatically using a trigger function

5.2 QGIS plugin

QGIS allows extending its functionality with custom plugins built either on C++ or Python. We use the latter. Please, notice that the plugin is developed for QGIS 3 and may be incompatible with QGIS 2. The link to the repository and some instructions can be found in Appendix C. The plugin combines the scheduling and routing functions. Fig.20 presents a view of the plugin's toolbar. We describe the buttons from left to right. There are five buttons:

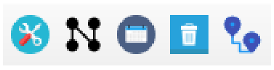


Figure 20. The plugin's toolbar.

<u>Setup:</u>
Setup is required to activate the other functions. It presents the user with a dialog with two types of input parameters: those related to the database connection (credentials and the names of the tables and fields to use) and the VRP constraints: the minimum and maximum number of vehicles, the tour duration limit and the vehicle capacity. Finally, the user is also asked to specify the depot coordinates. *The coordinates must be in WGS 84! Internally, they will be converted to the coordinates of the CRS of the containers table. The plugin does not check whether the container and the roads tables are in the same CRS, this is up to the user to ensure.* We used the coordinates of Stadsreiniging Almere (52.35, 5.22) where, as we know from the interview, all the vehicles are stationed.

After the user rounds off, the plugin checks the PostgreSQL inputs by running a test connection. If the connection fails, the user will be presented with a warning message 'Could not connect to the database!' If any other inputs are invalid, the plugin shows another warning message saying 'One or more input values are invalid!' If everything is correct, however, the other buttons become enabled.

<u>Convert road network into graph:</u>
This function does what is described in section 4.6. It takes the road dataset (table specified in the previous step) and applies the GRASS v.net.allpairs to it. Subsequently, it applies 'Aggregate' from the QGIS native tools to aggregate the elements of the same shortest path into a single line feature as described in section 3.5. The outcome is a GeoJSON file stored in the plugin's 'temp_data' directory that contains the shortest paths between each pair of vertices, where vertices are either containers or the depot (i.e. their locations).

<u>Schedule collection dates:</u>
The collection dates are assigned to containers according to the procedure described in section 5.2. Upon its success, the user is presented with 'Scheduling done' message.

<u>Show containers:</u>
Loads a set of containers scheduled one layer per date.

<u>Find routes:</u>
Solves the CVRP for each of the collection dates and loads each solution as a separate layer. The algorithm is based on Google OR Tools (Google Developers, 2019b). We use an implementation of Guided Local Search provided in that toolkit to solve the VRP in

combination with the Cheapest Insertion method to construct the initial solution. Next section discusses the method in more detail.

5.3 Solution method used in the implementation

Google OR Tools offer simple Local Search (Steepest Descent), i.e. a greedy tour improvement heuristic, and three metaheuristics that we described in the previous chapter: Simulated Annealing, Tabu Search and Guided Local Search. The developers recommend Guided Local Search for VRP but mention that there is no single 'best' (meta-)heuristic, and the performance is problem-dependent (Google Developers, 2019b). In this argumentation they refer to the 'No Free Lunch' theorem by Wolpert and Macready (Wolpert & Macready, 1997).

For the reasons described in the previous chapter, our choice fell on Guided Local Search. In the Google OR Tools implementation, shortest paths between destinations are used as features, penalties for each edge are initialized to zero and are incremented by one every time the search runs into a local minimum. Only the highest-utility (single) edge is penalized each time. λ is a parameter with default value $\lambda = 0.1$. The Local Search methods implemented in OR Tools uses the 'best-of' approach. We also specified one-minute service time for each container so simulate the time spent emptying the containers.

There are different options for construction of the initial solution: Nearest Neighbor (default), the Savings method, Nearest Insertion and Cheapest Insertion (both parallel and sequential versions) and also other heuristics that were not covered in the study: Christofides', Cheapest Edge and Sweep methods. The user can also construct a custom heuristic. We chose the Cheapest Insertion method (called 'Best Insertion' in Google OR) because it has a fixed worst-case performance ratio 2 (unlike, e.g. Nearest Neighbor and Clarke-Wright), is suitable for directed graphs (unlike the Cristofides' method) and the literature suggests that it performs better on average than Nearest Insertion (see section 4.4.3).

The stopping condition can be either a runtime limit or a limit on the number of moves performed. Additionally, a limit can be set on the time spent on evaluating a neighbor solution. In our case, the runtime limit is the most intuitive choice.

During the development, we noticed that the minimum number of routes sufficient for a feasible solution was always the optimal one, meaning that employing extra vehicles would always give us solutions of at best the same cost or higher. Therefore, our algorithm finds a solution for each number of vehicles starting from the lower bound and stops one a feasible solution has been found. We cannot provide any strict proof of this concept, but we believe that haulers would, in most cases, like to minimize the number of vehicles used. We also presume that there may be (substantial) marginal costs of using an extra vehicle (and hence, also, a few more people) and since we do not incorporate them in our cost function, the actual cost of a solution, as the number of vehicles increases, may rise faster than the cost function's value reflects.

5.4 Synthetic data used for testing

Since we did not have an opportunity to deploy sensors in actual containers, we had to use mock data to test our applications. The structure of the software did not change, but in the plugin's *data_manager.py* file, we added a function called *generate_fill_levels*. It can be called with an array of integers in the interval $[1,12]$ that represent the months for which the synthetic fill levels should be generated. The function is in the 'main' of the module, so it will automatically execute upon a call to the file. For each sensor (container), it inserts one record for each calendar date in the specified time interval; daily increase is picked randomly in the interval $[0.05, 0.2]$. We chose these bounds arbitrarily as we did not manage to receive any information from Almere. Therefore, we do not aim at evaluating the results, but rather use mock data to facilitate the application testing.

5.5 Testing

The tests were run on HP Pavilion laptop with an Intel Core i7-7700HQ CPU with two cores 2.80GHz each, and 16 GB of RAM. We used metaheuristics, and for those to converge, one should provide a stopping condition; we used time limit. Hence, the runtimes are predefined by the user, and there is no 'speed' comparison possible. However, there is an important preparatory step of finding a fastest path between each pair of containers or a container and a depot. This is implemented as an exact algorithm, and it does vary significantly with the number of containers. Table 5 shows the average time it took to produce fastest paths for all pairs of vertices using our GRASS GIS + QGIS combination that was described in 3.5.

Table 5. The runtimes for v.net.allpairs (GRASS) + aggregate (QGIS).

20 containers	100 containers	500 containers
< 1 minute	≈ 8 minutes	≈ 300 minutes (5 hours)

The runtime grows very fast with the number of vertices which was expected. We attempted to run the procedure for 1000 vertices, but it crashed after almost 20 hours of runtime. We believe that this was mostly caused by the suboptimality of our implementation, namely using GeoJSON files for input-output (the options we had to choose because of v.net.allpairs does not seems to offer an interface to write to Geopackage or PostgreSQL, for example, at least not when used via QGIS API). We discuss this matter at the end of Chapter 6.

We used maximum route duration $T_{max} = 420$, i.e. 7 hours, assuming a standard 8-hour working day with a lunch break of 30 minutes and two coffee breaks of 15 min. The same constraint value is used, for example, by Nuortio et al. (2006). This assumption only works, of course, if the workers do not have to return to the depot for these breaks (have a lunch box with them or have their lunch at a location along their route. Next, we use vehicle capacity $Q_{max} = 10000$, i.e. 10 tons. This value was obtained from the specifications (RDW, 2019) of the vehicles (DAF CF 290 FAN trucks with a waste compression mechanism) that is, according to some sources (Geesinknorba Group, 2018; Omroep Flevoland, 2018), are used by Stadsreiniging Almere. We do not know the number of vehicles they have, nor whether there is a minimal number required to be used. We assume that the goal is to minimize the number of vehicles used as described in section 5.3.

For testing, we randomly selected 100 containers from BGT. Next, we generated simulated fill levels for them as described in the previous section and did the scheduling (fig.21):

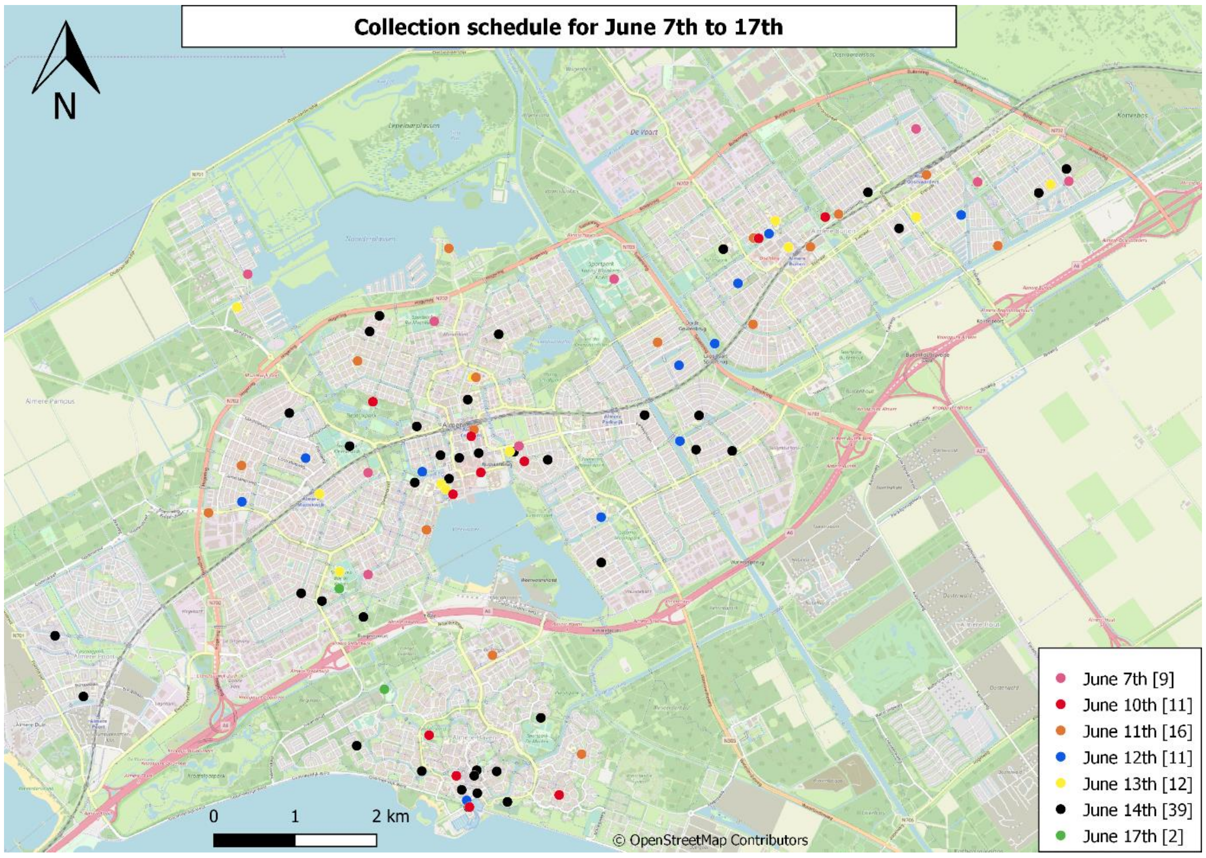


Figure 21. Collection schedule generated for the 100 randomly selected containers. The legend specifies the date and the number of containers on that date, in square brackets.

Fig. 22 presents the resulting routes:

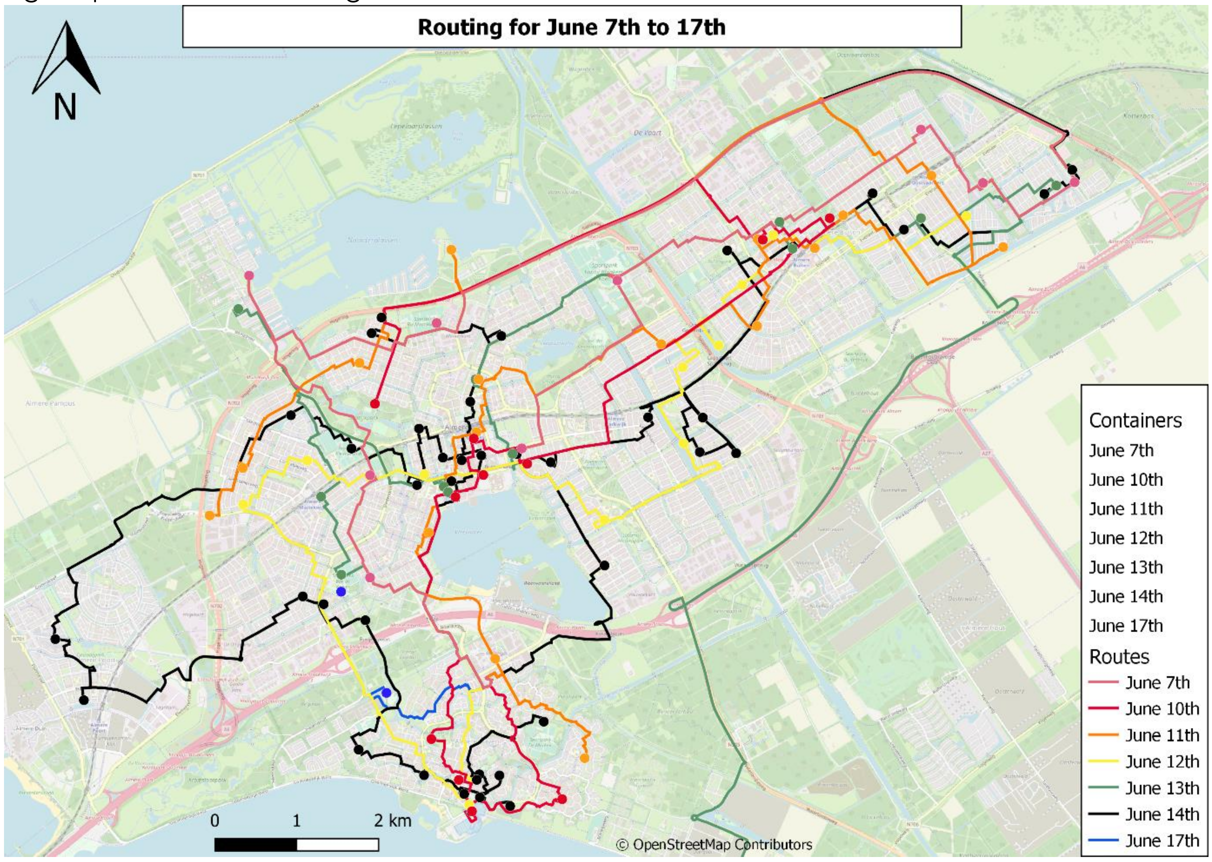


Figure 22. Routes generated for the 100 randomly selected containers from fig.20 using Guided Local Search and Cheapest Insertion for the initial solution. Note that there is one route for each date.

Despite our choice in metaheuristics fell on Guided Local Search for the reasons discussed in the previous chapter, we decided to compare it with the other two metaheuristics implemented in Google OR tools: Tabu Search and Simulated Annealing. All these metaheuristics require an initial solution, and the results may differ depending on the heuristic chosen to generate it. We justified our choice for the Cheapest Insertion method in section 5.3. To check how the initial solution impacts the results, we run each of the three metaheuristics with each of the three heuristics. We ran these cross tests for two different selection sizes, 100 containers and 500 containers. Finally, we use a runtime limit as a stopping condition for the metaheuristics and test two values: one minute and ten minutes. Let us first present the schedules for 100 containers (fig.23) and 500 containers (fig.24):

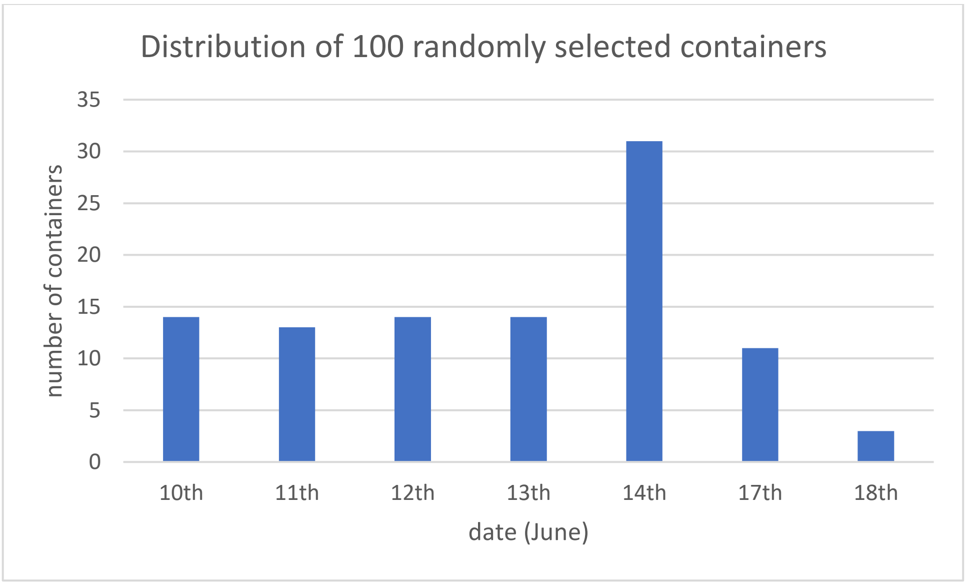


Figure 23. Number of containers scheduled (synthetic data).

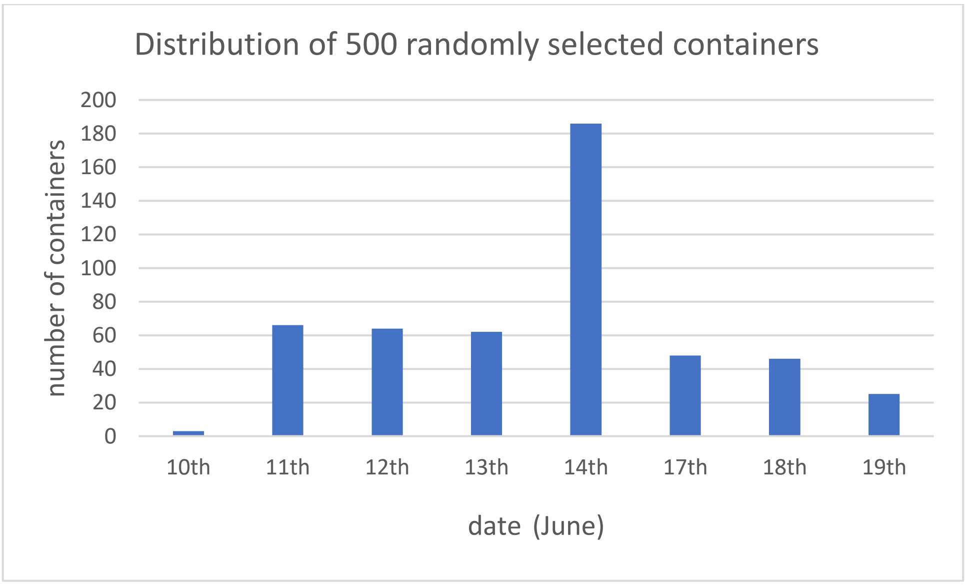


Figure 24. Number of containers scheduled (synthetic data).

For 100 containers, each date's solution required only one vehicle. For 500 containers, however, the number of vehicles ranged from one to three. And both statements apply to the results obtained by all metaheuristics and tour construction method combination (fig.25):

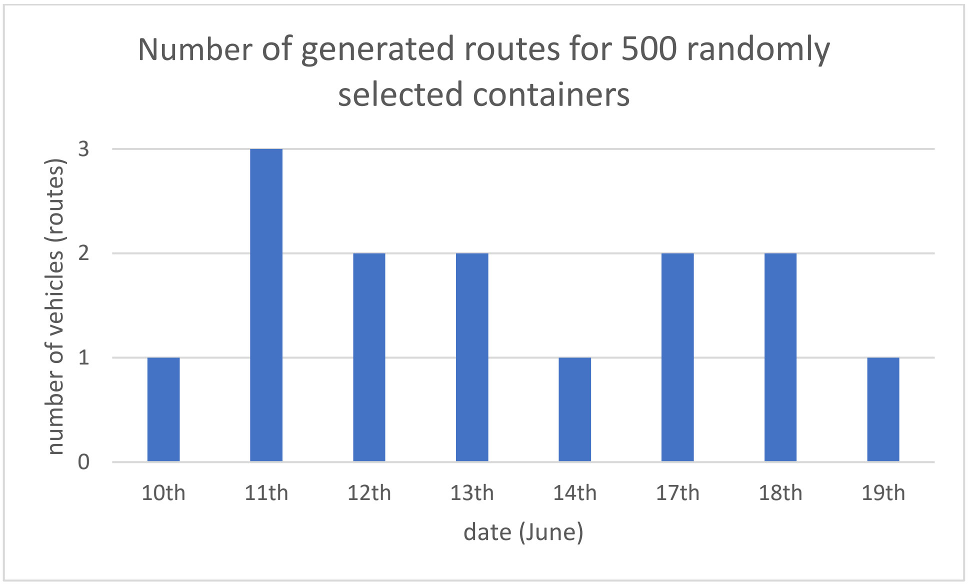


Figure 25. Number of generated routes (synthetic data). This number coincided for all metaheuristics.

We can see that the distribution in fig.25 does not logically follow the one in fig. 24. For example, 14th of June is by far the leading day in terms of the number of containers scheduled, but they all fit in one route, just like on the 10th of June for which there were just a few containers scheduled. At the same time, for the 11th of June, three vehicles were required. We inspected these solutions and found that *all* extra vehicles were added after the capacity constraint was reached. No routes came close to reaching their duration constraints; the longest of them, the solution for the 11th of June, was only 245 minutes as found by an exact method. It is fair to repeat here, however, that we did not use traffic data and did not have access to the real speed limits either, so the absolute tour duration values may not make much sense here.

First, we found the optimal solution for each of the selection sizes. To do so, we did not specify any metaheuristic in Google OR tools, thus using the default parameters. The documentation does not indicate which exact method is used, but this would only matter for runtime comparisons. Table 6 presents the optimal solution costs (total for each date):

Table 6. Performance of an exact method for 100 random containers.

	100 containers	500 containers
Total cost	511	1232
Runtime	≈ 3 minutes	≈ 390 minutes (3h 30 min)

Let us now present the comparison between the solution obtained using different combinations of metaheuristics and tour construction heuristics of our choice. We ran each combination ten times. Since the objective is to minimize the overall cost of transportation for the time horizon, i.e. all scheduled dates, below we only present the summarized costs for all the dates in the give solution. These summarized costs were further averaged over the ten runs. Tables 7 and 8 present the results obtained for 100 containers (runtimes one minute and ten minutes, respectively). Table 9 and 10 show the results for 500 containers for the same runtimes.

Table 7. Total travel time (min) for 100 random containers, search time limit – 1 minute

	Nearest Neighbor	Cheapest Insertion	Clarke-Wright
Guided Local Search	556	556	557
Tabu Search	558	566	558
Simulated Annealing	560	561	559

Table 8. Total travel time (min) for 100 random containers, search time limit – 10 minutes

	Nearest Neighbor	Cheapest Insertion	Clarke-Wright
Guided Local Search	556	556	556
Tabu Search	556	561	556
Simulated Annealing	558	561	559

Table 9. Total travel time (min) for 500 random containers, search time limit – 1 minute

	Nearest Neighbor	Cheapest Insertion	Clarke-Wright
Guided Local Search	1354	1353	1353
Tabu Search	1360	1369	1364
Simulated Annealing	1381	1382	1378

Table 10. Total travel time (min) for 500 random containers, search time limit – 10 minutes

	Nearest Neighbor	Cheapest Insertion	Clarke-Wright
Guided Local Search	1353	1353	1353
Tabu Search	1355	1359	1358
Simulated Annealing	1381	1382	1378

It appears that Tabu Search is the most sensitive to the optimality of the initial solution. It turned out to have performed the worst with the Cheapest Insertion method, despite our hopes for its efficiency. It is hard for us to say what the reason may be. Guided Local Search, on the opposite, is almost insensitive to the choice of the tour construction heuristic. This may be due the similarity of the tours they produce, or because the improvement procedure generally has a much bigger impact on the result, as it was described in section 4.6.3. Unfortunately, our implementation does not allow to inspect those initial solutions. Simulated Annealing was somewhere in between, with slight variations in the results. It seems to work best with the Savings method (among the ones tested).

As to the time limits set for the metaheuristics, for Guided Local Search and Simulated Annealing, in both cases the results were nearly identical. It appears that one-minute runtime is enough for these metaheuristics, and the results do not improve much afterwards; for the given problem sizes, of course. Tabu Search showed the most noticeable improvement from one-minute to ten-minute runtime.

Let us now take the best result for each metaheuristic (among all its combinations) and compare their performance with the results obtained using an exact method. We use the formula $\frac{T(S_A)-T(S_{OPT})}{T(S_{OPT})}$, i.e. the excess of the metaheuristic's best solution's cost over the optimal solution's cost. Table 11 summarizes the results:

Table 11. The excess of the metaheuristic's best solution's cost over the optimal solution's cost.

	Guided Local Search	Tabu Search	Simulated Annealing
100 containers	8.81%	8.81%	9.20%
500 containers	9.82%	9.98%	11.86%

Overall, Guided Local Search and Tabu Search produced much better results than Simulated Annealing, especially for 500 containers. Between the former two, there is only a minor difference for 500 containers, so it fair to say that they performed equally good. However, Guided Local Search was able to arrive at those results within one-minute runtime, while for Tabu Search it took more time. Additionally, the insensitivity to the underlying tour construction heuristic may be an advantage of Guided Local Search because it makes it easier to implement.

6 Discussion

In this chapter, we reflect on the results of the study. We start with some general discussion (6.1), continue with the limitation imposed by the project's scope (6.2), discuss some direction for future work (6.3) and round off with a more specific talk about the presented implementation (6.4).

6.1 General notes

In chapter 4, we presented a number of exact methods and heuristics for solving the VRP. We had to narrow our selection down to only include two exact methods, three metaheuristics and several simple heuristics. There exist more algorithms in each category, but we had to mind our time constraints and only show a few examples. We encourage the reader to explore beyond those and challenge our findings. In our examples, we tried to be concise and find the right balance between generality and specificity.

Metaheuristics tend to be complex and relatively abstract: there exist different versions of each, and those versions tend to have multiple parameters, so that the reader may find it hard to translate the general recipe into a specific implementation. With this in mind, we provide one example implementation for each metaheuristic with pseudocode and parameter choice examples. At the same time, we try to avoid being too specific, leaving it to the reader, for example, whether to implement the given algorithm in an iterative or in a recursive fashion. We also present the test results reported by various author, but we advise the reader to not take those numbers as a well-proven result, because, first, some of them are several decades old and, second, they depend on the exact implementation of the given algorithm and may even be affected by chance or human error.

We emphasize the difference between the symmetric and asymmetric problems (see section 4.4.1) as it may render some of the existing methods inapplicable to the given situation. A large body of research on the TSP and the VRP focuses on symmetric problems, while the real-world applications, especially in logistics, mostly require that the problem be modelled on an asymmetric graph. This requirement naturally stems from the way the human-made transportation networks operate; one-way streets are an example. Garbage truck routing is one of those applications, and we decided to omit some prominent algorithms, e.g. the Christofides method, that only work on symmetric networks.

This project was devised as a case study, but due to the circumstances that were out of our control, we had to correct the original plan. Instead of a case study, we presented a minimalistic two-piece software prototype that one could use a template for further development. We used free-and-open-source software because of our limited resources and because it is easy to customize and combine the different pieces, such as Python and QGIS. We believe that there may exist commercial software packages (the inner workings of which may not be known or are intentionally hidden) that are able to solve the problems of the type and size presented in this work to optimality, using exact methods, but in our implementation, we resort to metaheuristics, namely Guided Local Search.

The diversity of existing methods and their implementations suggests that there is no one best method for solving the VRP. Most authors present some comparisons and benchmarks but many of them only do so for symmetric problems and only include some of the existing methods while omitting the other. The empirical results strongly depend on the size of the problem, the spatial configuration of destinations (randomly distributed or clustered) the imposed constraints, whether the costs obey triangle inequality, etc.

Municipal solid waste management is a multipart process. In this study, we focused on the waste collection and transportation, and on the scheduling of these. We left recycling and the methods of collection (curbside collection vs communal containers, etc.) aside, only giving those a few comments. However, those aspects may influence the way the waste is collected and transported. There exist different types of waste collection vehicles, from small dust carts that service the containers located in pedestrian areas to multicompartment garbage trucks that can collect more than one type of waste at a type. On the other hand, there are also different types of containers: small street bins, communal recycling bins with underground storage space, traditional communal containers, personal containers like Klikos, and potentially more different variations. A working solution may require a more complex model than the one we present in this study. Let us summarize some aspects that were left beyond the scope of this study but are, nevertheless, important for routing and/or scheduling of waste collection:

6.2 Study limitations

Here, we briefly discuss some simplification we imposed on our problem model form the very start because we had to bound the project's scope to fit into the limited time span allotted.

6.2.1 Single depot

In this project, we used Stadsreiniging Almere as the depot, which, as far as we know, is the actual starting point for all waste collection vehicles. We also used it as the end point for the routes, although we believe that after emptying the last container on the tour, a vehicle first goes to a recycling plant located elsewhere. We do not know exactly if this assumption is true, however, and if it is, where this plant may be and whether there is one such end point or, perhaps, different types of waste are delivered to different facilities.

6.2.2 Fleet homogeneity

We potentially simplified the problem by assuming that all vehicles are exactly the same: their capacity, the suitability for the different types of waste containers, etc. We believe that this is, generally, not true. We focused on the communal recycling containers that are normally located along the streets. Those require, to the best of our knowledge, garbage trucks with a special lifting mechanism. At the same time, there are smaller 'street bins' that are often located in pedestrian zones, e.g. parks, where the large garbage trucks cannot go. Those containers are commonly emptied by smaller dust carts. Therefore, it is arguably more common for the fleet to be heterogeneous. At the same, we believe in this case the overall problem can be split into several problems, one for each type of vehicle and the corresponding type of containers. For the examples of studies that address this issue, we refer the reader to Archetti et al. (2014).

6.2.3 Waste separation

Even if the fleet is homogeneous, the waste is not, and its separate collection gradually becomes a norm. Therefore, the clustering of containers by route should also be based on the type of waste. It may be one type of waste for one vehicle, but to our knowledge, many modern garbage trucks feature two or more isolated compartments, so one vehicle may collect more than one type. However, first, there are usually four to five types of waste, and we believe that garbage trucks are limited to three compartments at most, and, second, each vehicle, in this case, has several capacities. This adds a lot of complexity to the problem and its modeling. For an example of solving a multicompartment VRP, see Reed, Yiannakou, & Evering (2014)

6.2.4 Collection methods

As far as we know, curbside (by-household) collection plays a prominent role in the Netherlands and is common, if not prevalent, in some other countries as well (although there are countries where it is uncommon, e.g. Russia). The ideas presented in this study mostly addressed the communal containers. Even though curbside collection can be done separately from collecting communal containers, we presume that there may be cases in which the haulers would like to be able to do both on the same day and by the same vehicles.

6.2.5 Floating schedule

In this study, we point out the flexibility of the schedule as an advantage because it may either reduce the operational costs or prevent overflow, or both (which would be a lucky situation). We suspect that in reality there are also some legal and organizational constraints to the scheduling process. Municipal solid waste collection is most often operated on tax money, and should it turn out that a more frequent collection is required according to the sensor measurements, the haulers might have to raise the taxes. Another problem that we discussed at the start of this paper, is that from the managerial perspective, as well as from the perspective of the collectors, assumingly, the more predictable and even the schedule is, the better. To address this problem, we discarded the idea of the 'on-the-fly' waste collection and emphasized the need for a scheduling horizon, e.g. 2-4 weeks. However, this still leaves room for longer-term dynamics, under which the workload may differ between the months, etc. Therefore, we believe that the implementation of such 'smart garbage collection' may face a certain amount of objection.

6.2.6 Time windows

Last but not least, there may be cases in which some containers can only be emptied within certain time spans. We believe that it is more common in commercial waste collection, restaurants being one prominent example. Here, by 'commercial' refers to the waste disposed by organizations, both private and public, and the legal status of the hauler. However, we cannot exclude such possibility in the municipal solid waste collection as well. Vehicle Routing Problems with Time Windows are a popular research subject, and for some examples we again refer the reader to Archetti et al. (2014).

6.3 Future directions

There were also some things we wanted to include in our study but did not manage to. These improvements may become the subject of a future project.

6.3.1 Schedule optimization

Our scheduling algorithm prevents overflows by choosing the date before the expected overflow, and it avoids the weekends, but afterwards it does not attempt to redistribute the number of containers more evenly across the scheduling horizon. From the managerial and operational perspective, an even distribution among the days is highly desirable, in the first place because the work hours for drivers and collectors must be acceptable, predictable and rational. A truly intelligent solution should be able to reshuffle the schedule to make it balanced.

We believe that there are two ways to do so: spatial and non-spatial. In the non-spatial way, the original schedule may be reshuffled based on statistical metrics such as variance of the number of containers scheduled for each day, e.g. variance must not exceed a certain value. We expect this approach to be much faster, but then which containers do we move from one date to another? This question will probably require a spatial approach. For example, we can decide which containers to move based on how they fit into the solution generated for a previous date. To this end, however, we have to first solve the VRP for each date, then correct the schedule, and then solve the VRP again, and it may require several such iterations. This is where (meta-)heuristics should come in handy. We did not, however, manage to devise such an algorithm within the time we had for this project. But we consider it crucial for a decent implementation and would like to address this need in our further research.

6.3.2 Accumulation rate prediction

We use mean accumulation rates, i.e. a simple average of the previous fill level records (their differences, to be more precise). We believe, however, that there may be temporal patterns, and their detection can improve the prediction. We expect weekly and seasonal cyclicity but without the data from Almere, we were unable to say anything with certainty. The implementation does not, however, necessarily require empirical results. We were able to make our plugin detect the periodicity use autocorrelation: for each time lag (number of days), it calculates the correlation coefficient, e.g. the actual time series of fill levels records and its copy shifted seven days forward reveals a correlation coefficient equal to a certain value. However, this is not enough, as these results must then be used to arrive at a complex function that would predict how much waste will be accumulated at each container within each given time span. Within the time we had for this project, we did not manage to solve this problem. For some ideas, we refer the user to Kannangara et al. (2017).

6.3.3 Cost function

In section 3.3, we discussed what a realistic cost function could be but came to a conclusion that we lacked sufficient knowledge about how vehicles work: how fuel consumption could be modeled as a function of travel time and/or distance, and what role vehicle maintenance would play in that function. Therefore, we resorted to travel time as the cost parameter. This might be an oversimplification, but this decision was also supported by the majority of the papers that we had studied. In fact, we had not come across any studies on Vehicle Routing Problems that would develop a complex cost function. Most of the authors seem to prefer travel duration (Archetti et al., 2009, 2007; Archetti & Speranza, 2014; Babaee Tirkolaee, Abbasian, Soltani, & Ghaffarian, 2019; Cordeau et al., 2002; Faccio et al., 2011). Some choose travel distance (Ramos et al., 2018a) and some leave the question open, saying that the implementation may use either travel distance or duration or mentioning both and not specifying the actual cost function (Ferrer & Alba, 2018; Silva, 2016).

6.4 Reflection on the implementation

At the end of this chapter, let us share our reflection on the implementation that we presented. During the work, we came across quite a few free-and-open-source tools for network analysis: Python libraries (NetworkX), C++ libraries (Boost Graph), PgRouting (an extension for PostgreSQL), GRASS GIS, QGIS native functionality. In addition, Google, TomTom and HERE (and possibly more companies that work in the same field) provide web APIs with powerful network analysis tools that can be used for free up to some limits (Google Developers, 2019a; HERE Maps, 2019; TomTom, 2019). There are limits on the number of calls to the API, but those are usually in the order of thousands of calls a month, which is more than enough for the application in this study. Much more restrictive are the limits on the number of destinations for routing, i.e. containers in our case. Google and TomTom will allow up to 25 destinations, for HERE Map, this figure is 50. Moreover, these limits remain the same for paid accounts as well. Therefore, we left these options aside in this study.

Among those FOSS options, mentioned before, we first had to first restrict our choice to the ones we have the necessary skills to work with, e.g. we do not have experience with C++ and we had to set it aside. Among the rest, Google OR tools impressed us the most: while NetworkX and PgRouting have many staple tools for network analysis like shortest path algorithms, we could not find more complex tools for solving the TSP, let alone the VRP (at least not in the online documentation). Google OR tools are FOSS and have a set of tools for solving Vehicle Routing Problems, including three metaheuristics and ten route construction heuristics. However, these tools take a complete graph (a cost matrix) as input, i.e. shortest paths between each pair of containers or a container and the depot, but there are not built-in tools for producing one from the actual road network and a set of containers (see section 3.6 for more explanation). The documentation suggests using Google Maps Distance Matrix API that, despite its name, can also produce a travel duration matrix. However, this API is web based and it is also limited to 25 destinations at a time. Therefore, we had to search for another way to do this part of the task.

QGIS was an obvious candidate; it features a function that computes the shortest/fastest path between two locations in a road network, and another one that extends it to find the shortest/fastest path between a point and a set of points. These functions are called 'Shortest path (point to point)' and 'Shortest path (point to layer)' respectively. So, to perform our task of finding all shortest paths for a number of points, we would have to batch those. And although this is not a challenging endeavor, we soon discovered a GRASS GIS tool called 'v.net.allpairs', i.e. it is called 'all pairs' and it makes part of the GRASS network analysis module. It does exactly what we needed with minimum effort, but we still compared the batched version of the QGIS tools with v.net.allpairs. The runtime results were much in favor of the latter. One obvious reason is that the batching we did was essentially telling Python to execute the QGIS function multiple times and then stitch the results into a single layer. The runtime was in fact 7-8 times longer, and the proceeded with the GRASS GIS tool. It does, however, have a major inconvenience: it returns each segment of each shortest path as a separate record. This is why we used a native QGIS tools 'Aggregate' to aggregate those segments into one record for each pair of destinations. The result is the desired all pairs of shortest paths that can be used further in OR tools, but this aggregation slows down the process. It is still much faster than the batched version of the native QGIS tools, but we believe that there may be a better 'all pairs' algorithm in PgRouting, namely the implementations of Floyd-Warshall's and Johnson's algorithms (pgRouting Contributors, 2019). Within the time bounds of this study we did not manage to test and compare those but we encourage the reader to do so because we believe that the runtime of our v.net.allpairs-aggregate pair are unnecessarily long.

This was also the reason we had to limit our testing samples to 500 containers for a planning horizon. Even Almere, which is a relatively small municipality with approximately 200,000 inhabitants, has about 1500 underground waste containers, and if we add the other types that potentially exists, this number may rise to several thousands. This means that our application should be tested for larger samples. We believe that the suboptimality of GRASS v.net.allpairs is the main and potentially the only bottleneck in our implementation. We encourage the reader to replace it with another solution.

7 Conclusion

There is a substantial amount of research done on the topic of optimizing waste collection. The authors come from a vast array of countries which indicates that the solutions in this field are being sought all over the world as the municipal waste collection remains one of the crucial and inevitable activities in any populated area. Despite being a mundane task, it can, in fact, incur high expenditures. But it also appears that these costs can be reduced by enhancing the scheduling and routing. In the search for a solution, many authors venture into the field of operations research and apply or develop relatively complex mathematical models.

Working in the field of GIS, we are most interested in the routing aspect of municipal solid waste collection, since it is a spatial problem. However, we discussed that routing is tightly linked to scheduling in reality, and a realistic solution cannot ignore the temporal dimension of the problem. We argued that this increases the problem's complexity so that it does not fit into the bounds of a classical Vehicle Routing Problem and therefore addressed both collection scheduling and garbage truck routing. In the scheduling part, we showed how the measurements received from the sensors can stored and analyzed to predict the accumulation rate for each waste container. Based on this rate, each of them can be scheduled for a certain date. For the routing part, we conclude that finding an optimal set of routes for a given date (hence through the containers scheduled for that date) can be described as solving a capacitated vehicle routing problem.

Vehicle routing problems have been studied extensively over the past 60 years, which resulted in an entire family of those, as well as in multiple alternative methods for solving them, among which there seems to be no universal solution. The fitness of the different approaches depends on two major variables: the size of the problem, most commonly defined by the number of waste containers, and the complexity of the problem expressed by the number of imposed constraints. We concluded that a scalable approach most likely requires the use heuristics, and we picked one of them, - Guided Local Search, - as a promising method.

Finally, we have presented a minimalistic software kit that receives the data from an ultrasonic sensor via the Things Network, a LoRa-based wireless data communication technology network, and saves that data into PostgreSQL. Further, the other part of the kit, - the QGIS plugin, - analyzes that data to calculate the expected overflow date for the given container and schedule its collection on one of the days prior to the overflow. Finally, the plugin uses Google OR tools, open-source and free software that implements a number of methods used for solving routing problems, to construct the routes for the garbage trucks for the territory of Almere. For testing however, we had to resort to creating synthetic fill level data as we did not have an opportunity to install sensor in actual waste containers.

We regarded Almere as a use case at the outset of the project and hoped that the emerging collaboration opportunities would give us a chance to base our study on the real municipal waste collection process in Almere. However, our plans had to be changed later. We focused on the algorithms and on a software prototype rather than on real implementation for a use case. Nevertheless, we hope that this study presented ideas and prototypes that can be used for further development.

References

Alfa, A. S., Heragu, S. S., & Chen, M. (1991). A 3-OPT based simulated annealing algorithm for vehicle routing problems. *Computers and Industrial Engineering*, *21*(1–4), 635–639. https://doi.org/10.1016/0360-8352(91)90165-3

ANWB. (2019). Snelheid in het verkeer: de regels. Retrieved May 27, 2019, from https://www.anwb.nl/verkeer/veiligheid/snelheid

Archetti, C., Feillet, D., Hertz, A., & Speranza, M. G. (2009). The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society*, *60*(6), 831–842. https://doi.org/10.1057/palgrave.jors.2602603

Archetti, C., Hertz, A., & Speranza, M. G. (2007). Metaheuristics for the team orienteering problem. *Journal of Heuristics*, *13*(1), 49–76. https://doi.org/10.1007/s10732-006-9004-0

Archetti, C., & Speranza, M. G. (2014). Chapter 12: Arc Routing Problems with Profits. *Arc Routing*, (August), 281–299. https://doi.org/10.1137/1.9781611973679.ch12

Archetti, C., Speranza, M. G., & Vigo, D. (2014). Vehicle Routing Problems with Profits.

Arnold, F., & Sörensen, K. (2019). Knowledge-guided local search for the vehicle routing problem. *Computers and Operations Research*, *105*, 32–46. https://doi.org/10.1016/j.cor.2019.01.002

Babaee Tirkolaee, E., Abbasian, P., Soltani, M., & Ghaffarian, S. A. (2019). Developing an applied algorithm for multi-trip vehicle routing problem with time windows in urban waste collection: A case study. *Waste Management and Research*, *37*(1_suppl), 4–13. https://doi.org/10.1177/0734242X18807001

Bundala, D., Bergenheim, W., & Metz, M. (2009). GRASS GIS manual: v.net.allpairs. Retrieved May 27, 2019, from https://grass.osgeo.org/grass77/manuals/v.net.allpairs.html

Centenaro, M., Vangelista, L., Zanella, A., & Zorzi, M. (2016). Long-range communications in unlicensed bands: The rising stars in the IoT and smart city scenarios. *IEEE Wireless Communications*, *23*(5), 60–67. https://doi.org/10.1109/MWC.2016.7721743

CNDigitek. (2019). Smart Waste Bin Detector DF702 LoRaWAN. Retrieved February 14, 2019, from http://www.dingtek.com/product/5-en.html

Cordeau, J. F., Gendreau, M., Laporte, G., Potvin, J. Y., & Semet, F. (2002). A guide to vehicle routing heuristics. *Journal of the Operational Research Society*, *53*(5), 512–522. https://doi.org/10.1057/palgrave.jors.2601319

EPA Victoria. (2018). Waste Materials Density Data. Retrieved May 20, 2019, from https://www.epa.vic.gov.au/business-and-industry/lower-your-impact/~/media/Files/bus/EREP/docs/wastematerials-densities-data.pdf

Faccio, M., Persona, A., & Zanin, G. (2011). Waste collection multi objective model with real time traceability data. *Waste Management*, *31*(12), 2391–2405.

Ferrer, J., & Alba, E. (2018). BIN-CT: Urban Waste Collection based in Predicting the Container Fill Level. Retrieved from http://arxiv.org/abs/1807.01603

Geesinknorba Group. (2018). GCP voor gemeente Almere. Retrieved May 31, 2019, from https://www.geesinknorba.com/gcp-voor-gemeente-almere/

Gemeente Almere. (2019). Almere Afval app en Afvalkalender. Retrieved February 22, 2019, from https://www.almere.nl/wonen/afval/almere-afval-app-en-afvalkalender/

Gendreau, M., Hertz, A., & Laporte, G. (1992). New Insertion and Postoptimization Procedures for the Traveling Salesman Problem. *Operations Research*, *40*(6), 1086–1094. https://doi.org/10.1287/opre.40.6.1086

Gendreau, M., Hertz, A., & Laporte, G. (1994). A tabu seach heuristic for the Vehicle Routing Problem: TABUROUTE. https://doi.org/10.1057/jors.2009.51

Glover, F. (1990). Tabu Search: A Tutorial. *Interfaces*, *20*(4), 74–94. https://doi.org/10.1287/inte.20.4.74

Golden, B., & Wasil, E. (2008). *The Vehicle Routing Problem: Latest Advances and New Challenges*. (B. Golden, S. Raghavan, & E. Wasil, Eds.), *ORCS* (Vol. 91). New York: Springer. https://doi.org/10 .1007/ 978-0-387-77778-8

Google Developers. (2019a). Developer Guide | Directions API. Retrieved June 2, 2019, from https://developers.google.com/maps/documentation/directions/intro

Google Developers. (2019b). Vehicle Routing Problem. Retrieved June 2, 2019, from https://developers.google.com/optimization/routing/vrp

Greco, G., Allegrini, M., Del Lungo, C., Gori Savellini, P., & Gabellini, L. (2015). Drivers of solid waste collection costs. Empirical evidence from Italy. *Journal of Cleaner Production*, *106*, 364–371. https://doi.org/10.1016/j.jclepro.2014.07.011

Hannan, M. A., Arebey, M., Begum, R. A., & Basri, H. (2011). Radio Frequency Identification (RFID) and communication technologies for solid waste bin and truck monitoring system. *Waste Management*, *31*(12), 2406–2413.

HERE Maps. (2019). HERE Developer. Retrieved June 1, 2019, from https://developer.here.com/

IMGeo. (2019). Objectenhandboek BGT. Retrieved May 28, 2019, from http://imgeo.geostandaarden.nl/

Johnson, D., & McGeoch, L. (1995). The traveling salesman problem: A case study in local optimization. *Local Search in Combinatorial Optimization*, 1–103. Retrieved from http://www.csc.kth.se/utbildning/kth/kurser/DD2440/avalg14/TSP-JohMcg97.pdf

Johnson, N. E., Ianiuk, O., Cazap, D., Liu, L., Starobin, D., Dobler, G., & Ghandehari, M. (2017). Patterns of waste generation: A gradient boosting model for short-term waste prediction in New York City. *Waste Management*. https://doi.org/10.1016/j.wasman.2017.01.037

Kadaster. (2019). BGT. Retrieved May 28, 2019, from https://zakelijk.kadaster.nl/bgt

Kannangara, M., Dua, R., Ahmadi, L., & Bensebaa, F. (2017). Modeling and prediction of regional municipal solid waste generation and diversion in Canada using machine learning approaches. *Waste Management*.

Kilby, P., & Prosser, P. (2002). Guided Local Search for the Vehicle Routing Problem. https://doi.org/10.1007/978-1-4615-5775-3

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Annals of Physics*, *54*(2), 671–680. https://doi.org/10.1126/science.220.4598.671

Laporte, G. (1992). The Vehicle Routing Problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, *59*(2), 231–247. https://doi.org/10.1016/0377-2217(92)90138-Y

Likotiko, E. D., Nyambo, D., & Mwangoka, J. (2017). Multi-Agent Based IoT Smart Waste Monitoring and Collection Architecture. *International Journal of Computer Science, Engineering and Information Technology*, *7*(5), 01–14. https://doi.org/10.5121/ijcseit.2017.7501

Lopes, D., Ramos, T. R. P., & Barbosa-Póvoa, A. P. (2015). Optimization of a recyclable waste collection system-the valorsul case study. In *Operations Research and Big Data* (pp. 97–105). Springer.

Mamun, M. A. Al, Hannan, M. A., Hussain, A., & Basri, H. (2015). Integrated sensing systems and algorithms for solid waste bin state management automation. *IEEE Sensors Journal*, *15*(1), 561–567. https://doi.org/10.1109/JSEN.2014.2351452

Nuortio, T., Kytöjoki, J., Niska, H., & Bräysy, O. (2006). Improved route planning and scheduling of waste collection and transport. *Expert Systems with Applications*, *30*(2), 223–232. https://doi.org/10.1016/j.eswa.2005.07.009

Omroep Flevoland. (2018). Er wordt echt gescheiden gestort. Retrieved May 31, 2019, from https://www.omroepflevoland.nl/nieuws/158506/er-wordt-echt-gescheiden-gestort

pgRouting Contributors. (2019). pgRouting Manual (2.6). Retrieved June 2, 2019, from http://docs.pgrouting.org/latest/en/

Prodhon, C., & Prins, C. (2016). *Metaheuristics for vehicle routing problems*. *Metaheuristics*. https://doi.org/10.1007/978-3-319-45403-0_15

Ramos, T. R. P., de Morais, C. S., & Barbosa-Póvoa, A. P. (2018a). The smart waste collection routing problem: Alternative operational management approaches. *Expert Systems with Applications*, *103*(351), 146–158. https://doi.org/10.1016/j.eswa.2018.03.001

Ramos, T. R. P., de Morais, C. S., & Barbosa-Póvoa, A. P. (2018b). Waste Collection Planning Based on Real-Time Information, 325–337. https://doi.org/10.1007/978-3-319-71583-4_22

Ramos, T. R. P., Gomes, M. I., & Barbosa-Póvoa, A. P. (2014). Planning a sustainable reverse logistics system: Balancing costs with environmental and social concerns. *Omega (United Kingdom)*, *48*, 60–74. https://doi.org/10.1016/j.omega.2013.11.006

RDW. (2019). Kentekencheck. Retrieved June 1, 2019, from https://ovi.rdw.nl/

Reed, M., Yiannakou, A., & Evering, R. (2014). An ant colony algorithm for the multi-compartment vehicle routing problem. *Applied Soft Computing Journal*, *15*, 169–176. https://doi.org/10.1016/j.asoc.2013.10.017

Rijkswaterstaat. (2019). Nationaal Wegenbestand. Retrieved May 27, 2019, from https://www.rijkswaterstaat.nl/zakelijk/zakendoen-met-rijkswaterstaat/werkwijzen/werkwijze-in-gww/data-eisen-rijkswaterstaatcontracten/nationaal-wegenbestand.aspx

Rosenkrantz, D. J., Stearns, R. E., Lewis, P. M., Ravi, S. S., & Shukla, S. K. (1977). An Analysis Of Several Heuristics For The Traveling Salesman Problem. *SIAM J. Computing*, *6*(3), 563–581. Retrieved from https://pdfs.semanticscholar.org/2081/25449f697c46d02a98eceb18b8c4622384c5.pdf

Roughgarden, T. (2016). CS261: A Second Course in Algorithms Lecture #16: The Traveling Salesman Problem *, 1–15. https://doi.org/10.1111/joim.12162

Rovetta, A., Giusti, A., Minghua, Z., Qichang, H., Vicentini, F., & Xiumin, F. (2009). Early detection and evaluation of waste through sensorized containers for a collection monitoring application. *Waste Management*, *29*(12), 2939–2949. https://doi.org/10.1016/j.wasman.2009.08.016

Silva, R. (2016). The multicompartment vehicle routing problem in the collection of recyclable municipal solid waste.

Skopal, T. (2006). On fast non-metric similarity search by metric access methods. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3896 LNCS*, 718–736. https://doi.org/10.1007/11687238_43

Stimular. (2018). Bepaling Hoeveelheid Afval. Retrieved May 22, 2019, from http://www.duurzamebedrijfsvoeringoverheden.nl/themas/afval/hoeveelheden.html

The Things Industries. (2019). The Things Network. Retrieved May 30, 2019, from https://www.thethingsnetwork.org/

TomTom. (2019). Routing API. Retrieved June 1, 2019, from https://developer.tomtom.com/routing-api

Vidal, T., Crainic, T. G., Gendreau, M., & Prins, C. (2013). Heuristics for multi-attribute vehicle routing problems: A survey and synthesis. *European Journal of Operational Research*, *231*(1), 1–21. https://doi.org/10.1016/j.ejor.2013.02.053

Voudouris, C., & Tsang, E. P. K. (2001). Guided Local Search.

Wolpert, D. H., & Macready, W. G. (1997). No Free Lunch Theorems for Optimization 1 Introduction. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. https://doi.org/10.1145/1389095.1389254

Appendix A. Preliminary interview with Stadsreiniging Almere

The interview was done on the 29th of January 2019. The respondents were the three managers of the municipal waste collection service (Stadsreiniging): Dani Puzic, Gerzon Chandler and Gerrie Kalkhoven. The responses represent their collective opinion.

1. Is waste collection in Almere done by the municipality itself or by a contractor company? And does this apply to both households and organizations?

Reply: Both are done by the municipality

2. What types of containers are used?

Reply: Communal underground containers are used for multi-apartment building. There are approximately 1500 such containers and 90 of those are equipped with sensors. For smaller houses, curbside collection is used (Kliko's and the like); sensors are not used.

3. How are container locations stored? Is this open data?

Reply: No, this data is not open, and it belongs to the municipality of Almere

4. Are the routes the waste collection vehicles travel defined by the actual fill-level of the containers at the time or are they fixed based on some kind of estimations?

Reply: Currently, the data from the 90 sensors that we have is not used for truck routing, but we would like to start using it as soon as possible with the new contractor.

5. We know that some municipalities use the software called Afvalris to optimize waste collection. Do you use Afvalris or any similar software?

Reply: Yes, we use Afvalris.

6. Which of these elements would you like to have done at the municipality, and not by the contractor?
   a) Visualization
   b) Visualization and Analysis
   c) Data storage, Visualization and Analysis
   d) Other (please, specify) ✓
   Reply: we are absolutely okay if the contractor does everything internally and only provides access to the final result. Our biggest wish is that it would all be integrated in one program (visualization, routing, data storage)

7. Which information would you like the waste container sensors to collect (fill level, fire alert, geolocation, etc.)?

Reply: Fill level and geolocation, the latter is needed for the Kilko's because they tend to be moved around and get lost. We would also like to use sensors to count the Kliko's because those are the property of municipality but sometimes it is hard for us to keep track of how many there are. We would like to have the same smart locks installed as we ones used for the underground containers. Then, they can be integrated with the fill level sensors so they could send both the waste level and the opening times count.

8. What are the current challenges/points of improvement?

Reply: The biggest issue is that these ultrasonic sensors often send erroneous fill levels if the waste inside the container is not spread evenly. We are now considering a sensor produced by Micodata,; it sends a batch of signals from different positions on the container's lid. Another major issue is short-lived batteries of the currently used sensors. The current model lasts eight months on average, instead of the promised two years, and its replacement costs around 125 euros per sensor, plus the work cost.

Appendix B: Interview with the municipality Houten

This interview was done on 2nd of February. Unfortunately, the contact person, Maarten van Schaik, did not have time to have an in-person interview, so instead we sent him a questionnaire. Here are the questions:

1) The municipal website provides some rules for household waste collection but only mentions curbside collection. Are there also large communal waste containers used for apartment buildings or organizations?

Maarten: Yes, we have those too, but we only serve households, not organizations.

2) How are container locations stored? Is it open data?

Maarten: No, they are stored in an old-fashioned way, in an Excel list.

3) Do you currently use any sensors to monitor the fill level of the municipal waste containers dynamically?

Maarten: No, we don't.

4) Are the routes that the waste collection vehicles travel defined by the actual fullness of the containers or are they fixed based on some kind of estimations?

Maarten: Fixed, based on historical records and empirical experience.

5) I know that some municipalities use the software called Afvalris to monitor the waste collection system. Do you use Afvalris or any similar software?

Maarten: No, not yet.

6) Some IT companies provide solutions for waste collection optimization. Which of these workflow elements would you like to have done at the municipality, and not by the contractor?
    a) Visualization
    b) Visualization and Analysis
    [✓ c) Data storage, Visualization and Analysis]
    d) Other (please, specify)

7) Which information would you like sensors to collect (fill level, fire alert, geolocation, etc.)?

Maarten: Fill level and geolocation.

8) Do you think installing sensors in waste containers and using real-time data about their fill levels could improve the waste collection system, and would it be worth the costs of implementation?

Maarten: Yes, of course, and let's not forget how it could help the households.

Appendix C: Links to the Code

QGIS plugin:
https://gitlab.com/nedaevg/almere-routing

TTN client:
https://gitlab.com/nedaevg/ttn-client

Database backup (PostgreSQL + PostGIS):
https://gitlab.com/nedaevg/almere_db