Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Juho Saarela

Optimising and automating work planning by approximating the vehicle routing problem

Master's Thesis
Espoo, November 24, 2016

Supervisor: Professor Lauri Malmi
Advisor: Licentiate of Philosophy Florian Berger

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF MASTER'S THESIS

Author:	Juho Saarela		
Title:	Optimising and automating work planning by approximating the vehicle routing problem		
Date:	November 24, 2016	**Pages:**	52
Major:	Software Technology	**Code:**	T-106
Supervisor:	Professor Lauri Malmi		
Advisor:	Licentiate of Philosophy Florian Berger		

The vehicle routing problem is a classic combinatorial problem with numerous real world applications. The basic problem is as follows: considering a depot and a list of service targets that all need to be visited, one needs to find the paths starting and ending at the depot with which the travelling time is as low as possible. In other words, VRP is the same as travelling salesman problem, except that there can be numerous routes and there are capacity limits to a single route. Because the problem is NP hard, finding the optimal solution is often out of the question. However, through heuristics, very good solutions can be found and used.

From package delivery to cargo transports and taxi services, VRPs are a part of our everyday life. In this thesis we present that an algorithmic approach can be applied to a construction company's job planning process to automate manual labour and produce more optimal results. This is accomplished by abstracting the company's business logic, such as job target locations and estimated working times into parameters for the algorithm. The algorithm then creates routes for technicians covering all upcoming job targets so that the driving time is minimised. Due to the simplicity of the routing, the main benefit in this case comes from the automatisation itself, as the quality of the plans is close to the results of manual labour.

Though the jobs are still manually assigned to technicians and the algorithm only optimises the order in which the jobs are done, it is shown that further development would make it possible to fully automate the process. The conclusion is that many companies could potentially benefit from this kind of automatisation.

Keywords:	vehicle routing problem, heuristic, optimisation
Language:	English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Juho Saarela		
Työn nimi: Työnsuunnittelun optimointi ja automatisointi käyttäen heuristisia menetelmiä ajoneuvojen reititysongelmaan			
Päiväys:	24. marraskuuta 2016	**Sivumäärä:**	52
Pääaine:	Ohjelmistotekniikka	**Koodi:**	T-106
Valvoja:	Professori Lauri Malmi		
Ohjaaja:	Filosofian lisensiaatti Florian Berger		

Ajoneuvon reititysongelma (Vehicle Routing Problem, VRP) on klassinen kombinatorinen ongelma, jota voidaan soveltaa lukuisiin tosielämän ongelmiin. Ajoneuvon reititysongelma voidaan kuvata seuraavasti: Etsi reitit, joita käyttämällä käydään kaikissa työkohteissa siten, että matka-aika minimoidaan ja jokainen reitti alkaa ja päättyy tukikohtaan. Se muistuttaa vahvasti kaupparatsun ongelmaa (Travelling Salesman Problem, TSP), paitsi että reittejä voi olla useita ja yksittäinen reitti ei voi ylittää sille asetettuja kapasiteettirajoituksia. Ajoneuvon reititysongelma on NP-täydellinen, joten optimaalisen ratkaisun tuottaminen on usein laskennallisesti mahdotonta. Kehittyneet heuristiikat kykenevät kuitenkin tuottamaan erittäin hyviä ja käyttökelpoisia ratkaisuja.

Pakettien ja rahdin toimituksesta aina taksiliikenteeseen, ajoneuvon reititysongelma liittyy vahvasti jokapäiväiseen elämäämme. Tässä opinnäytetyössä esitetään algoritminen lähestymistapa rakennusyrityksen työnsuunnittelun automatisointiin. Sen avulla pystytään vähentämään ihmistyön tarvetta ja tuottamaan parempia tuloksia. Abstrahoimalla yrityksen toimintalogiikkaa, kuten työkohteiden sijainteja ja työmääräarvioita voidaan näitä käyttää parametreina algoritmille, joka luo reitit työkohteiden kautta siten, että ajoaika minimoidaan. Reitityksen yksinkertaisuuden vuoksi tässä tapauksessa suurin hyöty saavutettiin itse automatisaatiosta, sillä reittisuunnitelmien laatu on likimain käsityöllä tehtyjen suunnitelmien tasoa.

Vaikka työt edelleen allokoidaan käsityöllä asentajille ja algoritmi ainoastaan optimoi asennustöiden järjestyksen, tässä opinnäytetyössä todetaan, että jatkokehityksen myötä koko prosessi olisi automatisoitavissa. Johtopäätöksenä todetaan, että monet yritykset voisivat hyötyä tämänkaltaisesta automatisaatiosta.

Asiasanat:	vehicle routing problem, heuristiikka, optimointi
Kieli:	Englanti

Acknowledgements

I wish to thank my psyche for finding the effort to do this!

Likewise, I'd like to posthumously thank Yrjö Kallinen and Carl Sagan for sharing their insight about reality with me. Though there are plenty of others who also deserve many thanks for their work, these two have happened to influence me the best.

Helsinki, November 24, 2016

Juho Saarela

Abbreviations and Acronyms

ABC	Artificial Bee Colony, a VRP solving heuristic
API	Application Programming Interface
ACO	Ant Colony Optimisation, a VRP solving heuristic
CVRP	Capacitated Vehicle Routing Problem
GA	Genetic Algorithm
GVRP	Green Vehicle Routing Problem
GRASP	Greedy Randomized Adaptive Search Procedure
HVRP	Heterogenous Fleet Vehicle Routing Problem
IDE	Integrated Development Environment
LRP	Location Routing Problem
MDVRP	Multi-Depot Vehicle Routing Problem
PVRP	Periodic Vehicle Routing Problem
R&R	Ruin and Recreate, a VRP solving heuristic
VRPPD	Vehicle Routing Problem with Pickup and Delivery
VRPTW	Vehicle Routing Problem with Time Windows
VRP	Vehicle Routing Problem

Contents

Chapter 1

Introduction

The vehicle routing problem (VRP) is a topic that has been researched with great intensity for a long time. The basic idea behind vehicle routing problem is that given a depot and a set of target locations, the depot must dispatch a number of vehicles so that each location is visited once and that the vehicles finally return to the depot. The total travelling cost should be minimised. Vehicle routing problem is a classic problem first introduced by Dantzig and Ramser in 1959. [12] It is closely related to the even more well-known travelling salesman problem, first presented by Hassler Whitney in 1934 [15].

Because VRP has numerous applications in many fields, there is much to benefit from the ability to create good solutions to it. Since VRP is an NP-complete problem as it is a superset of the travelling salesman problem, it is typically necessary to use heuristics to get a solution, as finding a perfect solution would be too intensive computationally [23]. The goal of this thesis is to survey the existing heuristics to the problem and use them in practice. I will analyse the quality of the solutions and computational cost of the algorithms.

1.1 Problem statement

In many businesses from postal services to food delivery and cargo transport a lot of time is spent travelling between the customers or depots. There is a good motivation to reduce the time spent travelling, because by itself it does not produce any value to the company, but rather generates only costs through wages, car usage and reserved equipment. Likewise, maximally employing vehicle capacity is important to prevent unnecessary round trips.

While planning the routes by manual work is feasible for small businesses, as a company scales up and the number of customers and technicians increases, automation becomes more and more important. At some point the overhead of implementing an automated system for the planning overcomes the cost of the ever increasing planning work.

Once the factors affecting the work are known, it is possible to abstract the company's operation to numbers which in turn can be used as input parameters for algorithms. Because the parameters affecting the route planning vary from business to business, not all algorithms will be feasible for all applications. However, there are numerous subcategories of the vehicle routing problem to address this issue.

The goal of this thesis is to find a way for a construction company operating in Finland to automate their route planning so that routes visiting all upcoming work sites can be generated near instantly. The plans should be better than what a human can do. The main problems are adapting the business data into information suitable for algorithmic use and actually generating the route plans from customer data. The end result should be a program that fetches customer data from a server, processes the data, and outputs the route suggestions for the user. Because the problem is NP-complete, the goal is not to find the optimal solution. However, modern heuristics are able to provide good enough solutions.

1.2 Structure of the Thesis

Chapter 2 discusses the business logic of the client company and describes what the needs for VRP in the context are. The various types of VRPs and the ways to solve them are presented in chapter 3. Chapter 4 explains how the business logic is translated into data usable in the algorithm, and chapter 5 describes how I actually solved the problem. Chapter 6 discusses how successful the project was according to my own views and those of the client company. Finally, in chapter 7 I describe how I see this project relates to the rest of the world and draw some conclusions about the project.

Chapter 2

Environment

This chapter describes how the client company operates and how the operation parameters translate into a VRP.

2.1 Business aspect

The client company for which this project is made is in the house maintenance business. They specialise in specific house features and do little else. They have multiple offices in Finland and they do deliveries and installations nationwide. Since the customer can be located anywhere in the country, the distances between the offices and customers vary greatly. Closest ones can be in the same neighbourhood, while the most distant ones are hundreds of kilometres away.

The jobs are put to the backlog and assigned to a technician after a contract for work has been made. This means that the routes are created separately per technician and not company-wide. The daily schedules for the technicians are made a few weeks in advance because the customers need to know beforehand when the installation job will be done. The technicians work in fixed pairs and each one acts as a franchising type entrepreneur. The "depots" of the technicians are their homes for the purpose of VRP, as they typically start and end their working days at home. This essentially means that instead of solving one big VRP including all the job targets and all techicians, there are instead numerous small VRPs, grouped by the technician assigned for the jobs.

The job types can be lumped into two categories. Installations of new items and service jobs for if it turns out that additional work needs to be done or an installation error needs fixing. In this thesis the focus is solely on the installation jobs.

When planning routes for the technicians, one has to take into account the requirements for each job and the limits of the workers and vehicles. There are also date and time constraints involved. The jobs have to be scheduled within 4-week windows determined by customers' preferences. Likewise, there may be specific hours of day during which the job must be done, if the customer needs to be present to let the technicians inside, for example. The installation materials are delivered separately to the customer beforehand from the factory and only after the delivery can the job be started. Optimally, the job should start as soon as possible after the delivery. The fact that the installation item deliveries are done separately means that the storage capacity of the installers' cars does not play a role in the route generation.

The total hours of the working day typically should not exceed 8 hours. Setting a hard limit on 8 hours per day would make the average lower than 8 hours as it is impossible to exactly use the full amount of time for a working day. To balance this out, the maximum working day duration is set to 8.5 hours in the solving phase.

Different technicians have different skills, and some may perform faster than others. This is difficult to take into account as measuring performance is an extremely difficult and error prone task. For the purpose of this thesis, all installers are considered to be equally capable.

The type of the house also plays a role. A detached house has different set of challenges than an apartment building. The size of the house also affects on the difficulty as well as the equipment and time requirements of the job.

Once the job is done, it is possible that the customer has additional requirements or has noticed some faults in the work, in which case a new trip has to be made to ensure customer satisfaction.

Currently planning routes for the technicians is done by hand by an employee dedicated to the task. Due to the mechanical, repetitive and complex nature of the task, it is a suitable target for a computer algorithm to solve.

The goal the client company has set is a decrease in driving distances as well as saving the 4 hours of work per week it currently takes to do the planning manually.

Because the vehicle routing problem is typically associated to solving cargo deliveries or pickups, applying it here requires some customisation to take the various constraints and requirements into account.

2.2 Input data conversion

The input data for the program comes from the client company's sales data. The data contains the following information:

- The address of the customer.
- The address of the technician ("depot").
- The type of the house.
- Types and number of tasks involved in the job.
- The items to be installed to the house or the items previously installed in case of a service visit.
- Equipment required for the job.
- The time window for the job.
- Estimates of the technicians' performances. How fast they are at the job.

Since the installation items are delivered separately, capacity is of no concern for the VRP. To prevent the algorithm from becoming overly complex, the time and resource requirements of a single job should be simplified as much as possible. This means that various weights need to be applied to various tasks and installation items, with bigger and heavier items taking more time and difficult building type or installation location also creating additional time costs for the job. The end result should be the following parameters per job target:

- Total time requirement for the technicians, affected by the house type and the amount of work to be done at the location.
- Distance and travelling time to other job targets and depot.

With this simplification, applying the customer data for a VRP algorithm becomes feasible without loss of any important data. Chapter 4 goes into detail as to how this conversion is done.

Chapter 3

Vehicle routing problem introduction

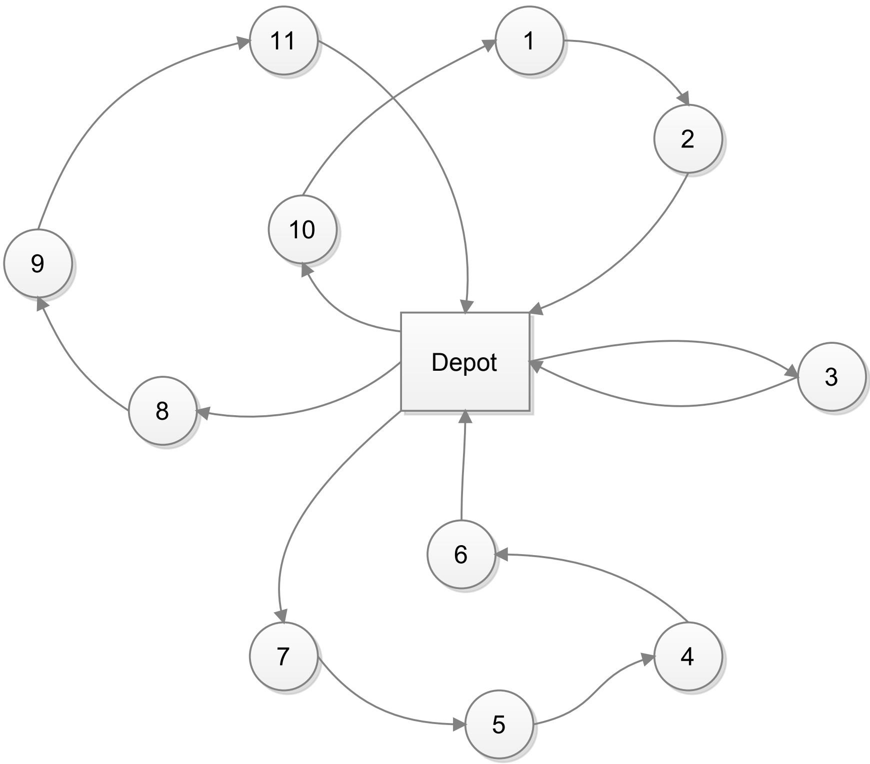

Figure 3.1: An example of a VRP solution

3.1 Classical vehicle routing problem

The basic VRP can be defined with a complete graph $G = (V, A)$ where $V = \{v_0, v_1, v_2 \dots\}$ represents the vertices, i.e. the depot and targets for the visitations and $A = \{(v_i, v_j) : i, j \in V, i \neq j\}$ represents the arcs between the vertices. The vertex v_0 typically represents the depot. For every arc, there is a non-negative travel cost $C = (c_{ij})$. In this context, it is used to represent the travelling time between vertices. If the travel cost is symmetric between all pairs of vertices, an undirected graph $G = (V, E)$ can be used instead, where $E = \{(i, j) : i, j \in V, i < j\}$. In all cases, the end result is an all-to-all network. [23]

The main idea in the network is that every node (i.e. vertex) has a connecting edge to every other node. Even if in real life there would be overlapping in the routes as one road probably leads to more than just one node, for the sake of the problem the edges are considered unique, as can be seen in Figure 3.2. In other words, even though the road map of a country does not consist of an all-to-all type network topology, the mathematical model used for this problem uses one. Even though the following diagrams do not show the lines between nodes unless they are utilised by a route, it can always be assumed that from every node there is a direct path to all of the rest of the nodes.

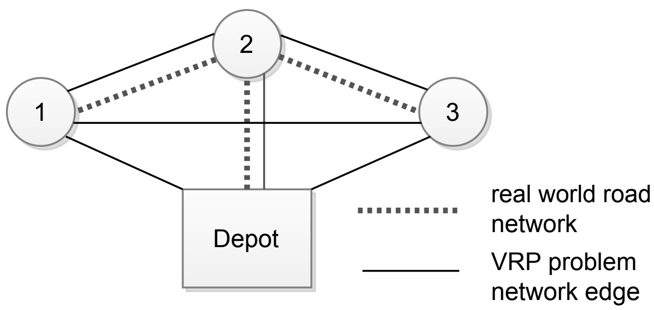

Figure 3.2: Real life network and the mathematical network. For example, in a VRP, if a vehicle wanted to go from depot to node 1 and back to the depot, the node 2 would be skipped, even if the real world route does go via node 2. The distance from depot to node 1 is calculated as the sum of the distances from Depot to node 2 and from node 2 to node 1, but past that, node 2 can be disregarded if it is not part of the route.

The first vertex in V represents the depot, where all the vehicles start from and where they must end their trips. m identical vehicles have a capacity denoted by Q which symbolises the resources available for each trip. In the classic VRP, m is logically equivalent to the number of trips needed to visit every customer. With time windows this no longer holds true. Every customer vertex $i \in V \setminus \{v_0\}$ has a demand $q_i \leq Q$ associated with it. This symbolises the various resources, including time, required at the target. [23]

Vehicle routing problem is NP-complete and it is a superset of the travelling salesman problem. In travelling salesman problem, the number of vehicles is $m = 1$, there is no upper limit on the costs of a route and the capacity of the vehicle is greater than the combined requirement of all the nodes in the network ($Q = \infty$). [23] The classic VRP is sometimes called the capacitated vehicle routing problem (CVRP). [20]

Let x_{ij} denote the number of trips made over the edge $[i, j]$ in a solution. The total cost that should be minimised is then:

$$\sum_{[i,j] \in E} c_{ij} x_{ij}. \tag{3.1}$$

The following conditions must hold true:

$$\sum_{j \in V \setminus \{v_0\}} x_{0j} = 2m, \tag{3.2}$$

meaning that given m routes, the edges between the depot and other vertices are used exactly $2m$ times, as seen in Figure 3.3. [23]

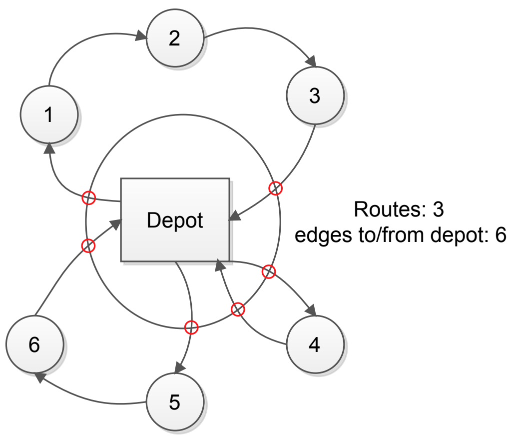

Figure 3.3: Routes to and from the depot

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \quad (k \in V \setminus \{v_0\}), \tag{3.3}$$

meaning that every vertex that is not the depot has two connecting edges that are used in the solution. In other words, every non-depot vertex is visited exactly once. [23]

If $b(S)$ represents the minimum number of vehicles required to satisfy the needs of the customers of S, the constraint

$$\sum_{\substack{i \in S, j \notin S \\ \text{or } i \notin S, j \in S}} x_{ij} \geq 2b(S) \quad (S \subset V \setminus \{v_0\}). \tag{3.4}$$

means the number of edges travelled between the vertices in the network contained in S and the vertices of the rest of the network has to be at least 2 times the minimum number of vehicles required to satisfy the needs of the customers of S. [23]

$$x_{i,j} = 0 \text{ or } 1 \quad (i, j \in V \setminus \{v_0\}) \tag{3.5}$$

and

$$x_{0,j} = 0, 1 \text{ or } 2 \quad (j \in V \setminus \{v_0\}) \tag{3.6}$$

mean that every edge whose other end is the depot is travelled at most two times, while the rest of the edges are travelled once at most. The reason why an edge can be travelled twice if it connects to the depot is that a route can consist only of visiting a single node and returning back to the depot. Using edges not connected to the depot twice would be pointless in all cases, as it would mean that the route returns to a node that has already been visited. [23]

3.2 Variations of the vehicle routing problem

The real life needs of route planning are rarely satisfied with the basic vehicle routing problem. There are typically additional constraints and an increased complexity that require expanding the problem statement. The vehicles used are probably not identical in terms of capacity and cost. The number of depots might also be greater than 1 and each target has to be allocated to one of them. [28] In addition, there might be specific time windows during which the targets have to be visited [18].

All of these additional features can be combined into a multitude of variations of the VRP. As all the variations add to the complexity of the VRP, they too are NP-complete. The variations listed here represent the most common types of VRP, but in addition, there are more specific types still available, such as one where only certain types of vehicles may visit certain targets. [25]

The variation employed in this thesis is a VRP with time windows, because the business logic of the client company dictates that customers need to be served within a certain time period. Vehicle capacity is not an issue since the installation items are delivered beforehand to the target sites. If the algorithm was to also allocate the jobs to the technicians, the problem to be solved would be a multi depot VRP with time windows. However, the client company preferred to keep that feature out of scope of the project.

3.2.1 Vehicle routing problem with time windows

Time windows and other scheduling constraints are common in real world and thus including them in VRP is crucial in order to get results that are applicable in common business cases. These can include postal, food or cargo delivery and service visits where it is common that the customer wants the delivery to occur during a specific time. [11]

To define vehicle routing problem with time windows (VRPTW), let s_i denote the time that a vehicle has to spend at a target i and b_i denote the time at which the delivery of the service or goods begins. The earliest time the target accepts the start of the delivery is e_i and the latest is l_i. If a vehicle arrives too early at j, it will have to wait so that the earliest time it can deliver the goods or services is $b_j = \max\{e_j, b_i + s_i + t_{ij}\}$. Here t_{ij} represents the time required to travel between i and j. [30]

3.2.2 Multi-depot Vehicle routing problem

Multi-depot Vehicle routing problem (MDVRP) is a variation of VRP where there are additional depots which can be utilised as the start and end points of routes as can be seen in Figure 3.4. A vehicle must start and end the route at the same depot. While at first it might seem that MDVRP can be just split into multiple single depot VRPs by assigning each target to the nearest depot, this leads to suboptimal solutions. [28]

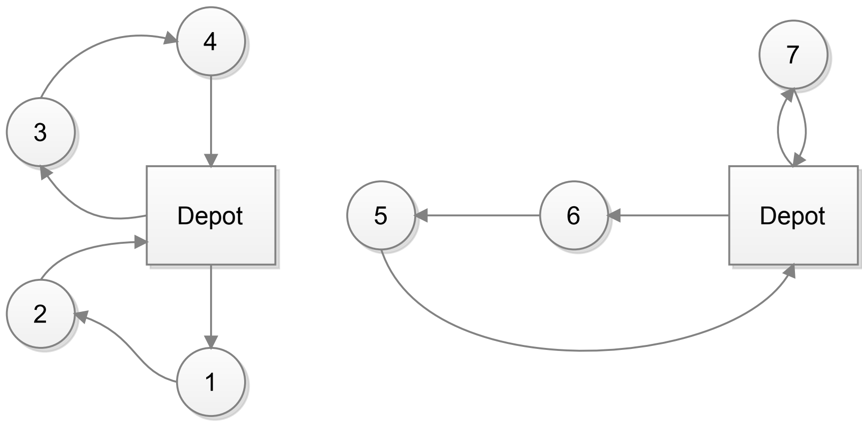

Figure 3.4: A VRP with two depots

3.2.3 Heterogenous fleet vehicle routing problem

In real world scenarios, it is uncommon that all the vehicles in a fleet are identical. A typical variation of the VRP called heterogeneous fleet vehicle routing problem (HVRP) one where the vehicles are not assumed to be identical. There is a fixed cost associated with each vehicle type and a variable cost per distance unit. Vehicle types also have unique capacities which play an important role in route generation. [17]

3.2.4 Vehicle routing problem with pickup and delivery

Vehicle routing problem with pickup and delivery (VRPPD) defines a VRP where a set of transportation requests must be satisfied. The requests all have a pickup point and a delivery point where the goods or passenger has to be brought. In case of passengers, a common application in real life can be found in taxi service. A courier service would be an example of a scenario where goods need to be transported from one point to another. Due to the nature of the applications for VRPPD, it is usually combined with VRPTW. The standard VRP is a VRPPD where the pickup point is always the depot and the delivery points are spread out elsewhere or vice versa. [13]

3.2.5 Periodic vehicle routing problem

While the classic VRP assumes a single time period, a single day for example, during which a deliveries have to be made, periodic vehicle routing problem (PVRP) defines a case with repeating time windows. Customers may need to be visited once or multiple times, and they may require the visit to occur on a specific weekday. [10] Common applications for the PVRP include waste collection, vending machine replenishment and cleaning service. PVRP is commonly combined with VRPTW as it is likely that in addition to wanting the visit to take place on a specific day, the customers also have requirements regarding the time of the visit. [33]

3.2.6 Location routing problem

In location routing problem (LRP), the location of the depots has to be determined before creating the routes. The factors that have to be taken into account are the one time and running costs of the depots and the travelling costs that are based on the locations of the depots. The main objective is to position the depots as close to the customers as possible. In practice, the maximum size of the depot is also limited. A large depot in a city centre might incur too large costs in order to be feasible. Once the locations of the depots has been decided, the problem turns into a multi-depot vehicle routing problem, though naturally the positions can be altered at any point during the solving. [32]

3.2.7 Green vehicle routing problem

Due to the apparent need to cut down CO^2 emissions and the fact that the vast majority of vehicles, especially heavy transport, run on fossil fuels, there is an increasing motivation to pay more attention to the environmental aspects of transportation. Green vehicle routing problem (GVRP, also known as emissions vehicle routing problem) aims to minimise emissions and helps routing for vehicles that may require special fuelling stations. [14]

3.3 VRP solutions

In this section I present some common algorithms to produce solutions to various VRPs. Because there are so many different types of VRP and each having its own set of solving methods, only the most common ones are listed here. In chapter 5 I will more closely analyse which additional VRP features are required in the business case of this thesis.

There are three main categories of VRP solving algorithms. The first category consists of algorithms that produce exact solutions. Due to the complexity of VRP, exact algorithms can only handle networks with up to about 100 nodes. This has resulted in the bulk of the research focusing on the two main types of heuristics: Classical heuristics and metaheuristics. [23]

Classical heuristics produce good results in reasonable computing time and cover the bulk of most modern heuristics in use. They are also easily suited for real world constraints and requirements. In metaheuristics, the most promising solution space is further explored. They employ advanced neighbourhood search patterns, memory structures and commonly use recombinations of solutions to find better ones. The downside is increased complexity and computational requirements, but they typically produce higher quality solutions. [24]

For notation, we will use (0, i, j, k, 0) to denote a route that starts at depot 0 and goes to targets i, j and k before returning to the depot. In case of multiple depots, they will be denoted with other indexes, e.g. (1, l, m, 1).

3.3.1 Savings algorithm

The savings algorithm is one of the most basic algorithms for VRP. It assumes that the number of vehicles is a decision variable. Figure 3.5 shows an example of how the savings algorithm progresses.

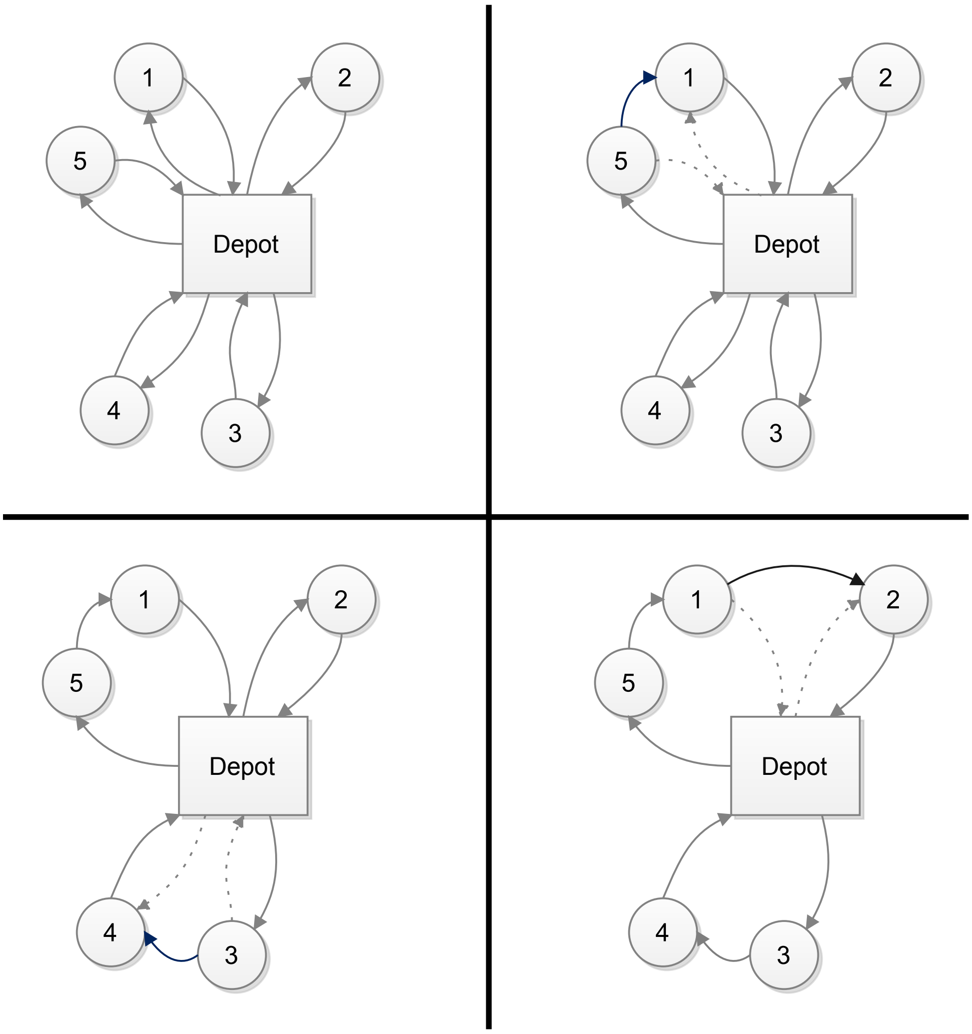

Figure 3.5: An example of savings algorithm progression where car capacity is 3 and demand per node is 1

The algorithm initially creates one vehicle route per node so that the route just goes to the target and comes back to the depot. This will result in n vehicle routes $(0, i, 0)$ for $i = 1, \ldots, n$. [27]

Step 1: Calculate *savings*: $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ for $i, j = 1, \ldots, n$ and $i \neq j$. Sort the savings in descending order. [27] At the top of the savings list are then pairs of targets who are farthest away from the depot and closest to each other. Essentially this gives us the clue that they should probably be merged into the same route. There are two common ways to use the savings list.

Step 2 (best savings first): Starting from the top of the savings list, see if the routes can be merged so that the saving s_{ij} causes the removal of $(0, j)$ and $(i, 0)$ and the addition of (i, j). [27] This means that no routes will be cut in half during the algorithm, but that routes are merged from end to end, excluding the depot.

Step 2 (route first): For each route $(0, i, \ldots, j, 0)$, find the first saving s_{ki} or s_{jl} that allows merging of another route that either ends with $(k, 0)$ or starts with $(0, l)$. Merge these two routes and continue the operation until no more feasible merges exist. Then move on to the next route and start all over again. [24]

3.3.2 Sweep algorithm

Popularised by Gillet and Miller in 1974, the sweep algorithm is usable on data sets where the coordinates of the targets are known. It is based on calculating the polar coordinates between the depot and the targets so that one arbitrarily chosen target represents the zero angle. The targets are then sorted according to the polar angle. The sorted targets are added in order to the new route until the cost or capacity limit of a single route has been reached and no more new targets can be inserted. Then a new route is created and the algorithm is repeated until all targets have been assigned to a route. [19]

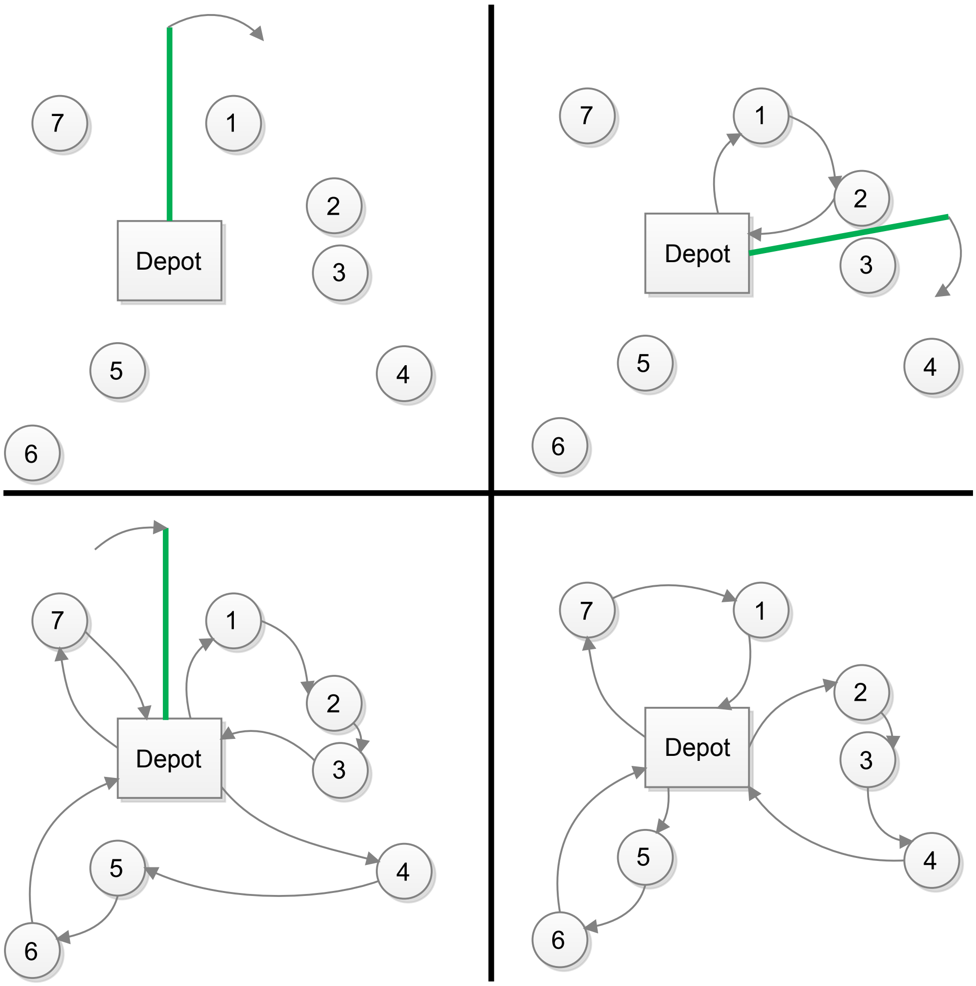

Figure 3.6: An example of sweep algorithm progression where car capacity is 3 and demand per node is 1. In the final iteration some routes swap nodes to produce a more optimal solution.

The found routes are then optimised with any travelling salesman solver. Since the routes are supposedly relatively small at this point, solving the individual TSPs should be a lightweight operation. Once each route has been initially created and optimised, the algorithm then tries to swap neighbour routes' targets according to their distance from the depot and the angle to the next route. If the swap improves the solution, the target is moved to the other route. After no more improvements can be found through swapping, the algorithm finishes. [19] Figure 3.6 shows an example of how the sweep algorithm works in practice.

The downside to this heuristic is that it doesn't take the road network into account during the sweep. It assumes that the geometrical distance is the most important piece of criteria. To counter this, it is typical to run the algorithm so that all targets get picked to be as the beginning point of the sweep. [27]

3.3.3 Petal algorithm

The petal algorithm resembles the sweep algorithm, as all the targets are first sorted according to their polar angle from the depot and an arbitrarily chosen point. A set S of possible routes called petals is then generated and optimised using TSP as above. The problem is then formulated as follows:

Minimise

$$\sum_{k \in S} d_k x_k \tag{3.7}$$

where S is the set of all routes in the set, $x_k = 1$ if and only if route k is part of the solution and d_k represents the cost of the whole route k. Apply the following conditions:

$$\sum_{k \in S} a_{ik} x_k = 1 \quad i = 1, \ldots, n \tag{3.8}$$

and

$$x_k = 0 \text{ or } 1 \quad k \in S \tag{3.9}$$

where a_{ik} is 1 only if the vertex i is in the route k. [24] The conditions state that every vertex belongs to exactly one route. The Sweep and petal algorithms are classic examples of "cluster-fist route-second" types of algorithms, where first the targets are grouped and then for those groups, the optimal routes are discovered.

3.3.4 Greedy Randomized Adaptive Search Procedure (GRASP)

Greedy Randomized Adaptive Search Procedure is a typical heuristic used in combinatorial problems. First an initial solution is created using a greedy function which places nodes into routes according to the effect on solution quality. The best insertions are done first and the ones which increase the total travelling cost the most are inserted last. If a node cannot be inserted into any route without breaking constraints or if the insertion would cost more than making a new route, the node is inserted into a new route. [22]

After the initial solution has been created, the algorithm performs local searches in order to try and improve the solution. It tries to randomly move single nodes to other routes while ensuring that the routes' constraints are not broken and that the new solutions are better than the previous one. [22]

3.3.5 Tabu search

The tabu search is based on randomly removing nodes from existing routes and adding them to other routes. Tabu search is used to optimise a solution already reached using a more basic solving method. What separates tabu search from more basic heuristics is that infeasible constraint-breaking solutions are temporarily allowed to prevent the algorithm from being too rigid. A change required to find a better solution might be so great that it is impossible to reach it by moving single nodes from one route to another and keeping the intermediate solutions valid. Illegal moves are accepted if they result in better moves becoming available later. [16]

Another special feature of the tabu search, and also where it gets its name from, is that the moves made during the solution finding process are put to a "tabu list". The algorithm avoids making these moves to prevent searching solution space which has already been searched. [16]

3.3.6 Ruin and Recreate

Developed by Schrimpf et al., the ruin and recreate algorithm is based on the concept of having a solution of varying quality as the base, after which all trips to a set of nodes are removed ("ruined"). The set can be picked randomly, from a radial area, or by some other rule. The intermediate result is the same solution as in the beginning, with some nodes not being serviced. The routes are then re-created so that every ruined node gets inserted into a route where it creates the least amount of additional time requirement so that no constraints (capacity, time, etc.) are broken. If it is impossible to insert a ruined node to a route without breaking constraints, a new route will be created. Once all the ruined nodes have been readded to routes, the algorithm can be repeated. An example run of the algorithm can be seen in Figure 3.7. [29]

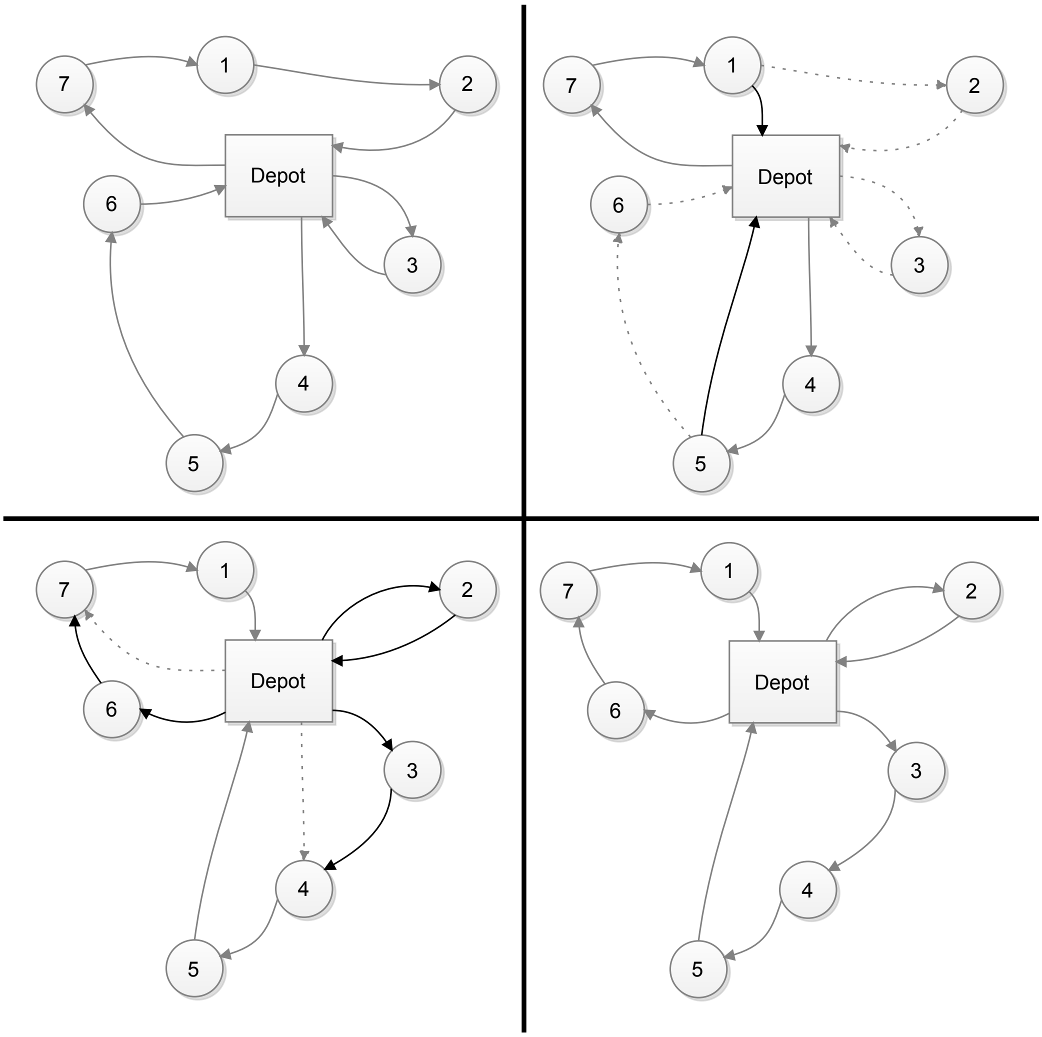

Figure 3.7: An example of ruin and recreate progression where car capacity is 3 and demand per node is 1. The ruined nodes are picked at random.

Shrimpf et al. state that there are probably better and more elaborate algorithms for the recreation procedure, but that even with their simple scheme, very good results can be achieved. Likewise, there are numerous strategies to choose from for the ruin part of the algorithm. Also, choosing which solution to continue from after iterations affects results. One way is to always use the best results gotten so far, but it may also pay off to try to continue from a slightly worse solution. A big benefit of the algorithm is that it allows easy applying of custom constraints. [29]

This algorithm is used in the library I picked to use for the VRP program. In addition it employs optimisation features presented by Pisinger and Ropke[26].

Due to the intuitive and simple nature of the ruin and recreate heuristic, it is easy to adjust the various phases of the algorithm as separate modules without affecting the rest of the algorithm. The ruining can be done randomly or by specific rules. For instance, nodes in routes with high travelling costs can be set to have a higher chance to be ruined. Routes which have not improved in past iterations can be set immune to the ruin phase. In case of time windows, nodes within a certain time frame can get priority for ruining. Capacity demand can also be used as a basis for the ruin strategy and it can be useful in cases where there is a delivery and pickup type of business logic involved. The advantage comes from exchanging packages of similar size to maximally utilise vehicle capacity. Historical knowledge can also aid in choosing the nodes to ruin. If certain edges have been present in all the best solutions so far, it can be beneficial to make the algorithm avoid ruining them. [26]

The recreation phase likewise allows numerous ways to add optimisations. While inserting the ruined nodes into existing routes as greedily as possible already gives good results, more sophisticated recreation methods can yield better results. Placing the easiest nodes first will give fewer options to place the more difficult ones, potentially preventing discovery of more optimal routes. In addition to using route costs to determine which routes the ruined nodes go to, adding some randomly determined costs will steer the algorithm to try out solutions it otherwise would not consider. Each of the recreated routes can finally be optimised using the travelling salesman problem. [26]

Choosing the most promising solution for a new iteration of the algorithm can also give an advantage. After the recreation phase and before the next ruining, always picking the best solution found so far might not result in the finding the best solution in the end. It is possible that a seemingly good solution might need a very improbable ruin in order to improve to a better one, but the better result might be more achievable if the algorithm allows for a slightly worse intermediate solution to be used as a base. Likewise, it can pay off to allow temporarily breaking the constraints to see if it leads to more varying and potentially better solutions. [29]

3.3.7 Advanced heuristics

There are also numerous highly sophisticated heuristics for optimising solutions. These are often inspired by nature, especially swarm behaviour of insects. Many species of insects follow a relatively straightforward set of rules, yet they are capable of optimising their communal activity so that the route distance to a food source for example is minimised. Applying this kind of logic to solving VRPs has resulted in numerous advanced algorithms. The specifics of these algorithms are too complex to be fitted in the scope of this thesis, but I will describe their idea on a general level. More information regarding them can be found from the cited sources. [21]

Ant colony optimisation (ACO) is based on the natural behaviour of ants in nature. A swarm of ants is capable of creating good routes to find resources and nourishment for the hive using pheromones which guide other ants. When encountering a pheromone trail, an ant is more likely to follow it the stronger the trail is. Ants which find a short route to a source of nourishment will mark their paths more often as they move between the nest and the target more often, resulting in a stronger pheromone trail and higher probability for other ants to follow the marked path. As the pheromones constantly evaporate, the suboptimal and less used routes will constantly diminish in strength. Since ants' decision making process does not always choose the path with pheromones, new paths are constantly probed. Creating a swarm of virtual ants can be used to find solutions to the vehicle routing problem. [9]

Artificial bee colony (ABC) solving method uses an abstraction where the solutions represent food sources, and better quality solutions represent higher quality nectar. Scout bees try to find new sources of nectar and onlookers pick from the found sources the ones which can be further used, upon which they become employed bees. Employed bees find new sources of nectar on each iteration of the algorithm and abandon old ones if they have not been providing as much nectar as the new findings or if there has not been an improvement in the food source for a predetermined number of iterations. [31]

Genetic algorithms (GA) use a population of solutions as a base. Offspring are created by merging these solutions together, keeping some characteristics of the different solutions. Natural selection is then applied to the new solutions. The best ones are used for further reproduction while the worst solutions are discarded. In the case of VRP, the goodness of a solution is determined by the total travelling time. Breaking constraints is also a negative aspect that lowers the chance of a solution to be used as a parent for the next generation. The most important aspects of genetic algorithms are the selection of the initial population and the breeding process itself. [8]

3.4 Comparison of algorithms

Comparing different VRP solving algorithms is quite difficult, especially if performance is taken into account. The hardware on which the algorithm is run as well as the specific implementation of an algorithm type serves a big role in how fast the algorithm completes. In iterative algorithms, there is no upper limit as to how long the algorithm can run and it might be impossible to reach the best possible solution reachable, since many times it is impossible to know if an even better solution might be found with additional iterations.

While there are some benchmark problem instances that many papers use in their results presentation, no standard exists, and whether two papers include results against the same instance is a matter of coincidence. Of the basic instances available, the papers select seemingly arbitrarily only some of them for results comparison. In addition, most papers compare the results only with similar solving methods or the currently best known results, making large scale comparison difficult. These differences can be explained in different algorithm types being aimed for different types of VRPs. It might not make sense to compare an algorithm solving a VRPTW with an algorithm capable of solving MDVRP.

Chapter 4

Methods

The basic problem can be divided into two parts. First is abstracting the business data into numbers and the second is applying the found numbers into the chosen vehicle routing problem algorithm. The business data includes locations of the job targets and the depots and the man-hour and equipment requirements of the jobs.

4.1 Abstracting the problem field into numerical data

Raw data from the client company's database is unusable by itself. It needs to be transformed into abstract numbers that can be used as parameters for the routing heuristic. The goal is to minimise the number of variables to prevent the actual algorithm from becoming overly complex while still ensuring that all available information is taken into account when generating the solution.

4.1.1 Generating the distance matrix

One problem in the project is transforming the location data, mainly addresses, into form that can be more readily used programmatically. The goal is to produce a matrix that lists the travelling time units between all the targets, including the depot. Because of the large number of targets, it is reasonable to optimise this step by grouping nearby targets together. Grouping should be done so that the travelling time between them is negligible and thus not much information is lost by the operation. The end result is that as far as the algorithm is concerned, the grouped targets occupy the same address. The threshold was set to 3 kilometres, so targets within that radius of each other are grouped together.

Choosing a suitable routing service is an important decision to be made in the early development. The way routing services work is that the user gives two addresses and the service responds with a recommended route to travel from the first address to the second. Additionally, they typically give the distance and travelling time estimate of the trip. The most important data to get from such a service is the time requirement of travelling between two addresses. This way it will be possible to turn address data into more abstract travelling cost matrix.

Geocoding is the process of turning address information into geographical coordinates which can then be used in a program more easily. Because the addresses of the customers are inputted by hand into the database of the client company, not all of them are valid. Some have typos, while others may have the wrong municipality due misinformation or the frequent municipality merges in the country. Possible ways to mitigate this problem would be allowing the user to re-enter the address in case the routing service does not recognise it. Another option is to group the job's location with one in the already existing jobs. In the context of this thesis, the program skips targets for which coordinates are not previously known. The source data that is used for the program is such that there has already been an attempt to geocode the addresses to coordinates, so it is unlikely that a previously unknown address would be valid on a second attempt. This is not impossible, however, as different geocoding services might produce different results for an address. Still, analysing such behaviour is out of the scope of this thesis.

Generating the many-to-many travelling cost matrix needed in this project requires a great number of queries. The licence fees of the routing services are typically based on the number of queries made within a time period. This means that the number of queries used in the program must be kept reasonable to keep the costs in check. The program should cache queried distances between addresses and group nearby locations together.

Naturally the routing service has to support Finland's addresses and roadmap. Another important requirement is that it can estimate the travelling time between targets, because distance by itself is not very descriptive due to varying speed limits on different roads. Traffic congestion will probably be impossible to take into account because the driving will take place weeks into the future and also because the exact time and date are not known at the time of the query.

One possible way to take congestion into account would be to redo the queries after an initial solution has been generated if the routing service can predict traffic based on previous dates' data and then adjust the solution if needed. However, the potential benefits would be limited at best and would not justify the increased complexity.

4.1.2 Calculating job costs

As shown in section 2.2, the raw data from the client company's database needs to be transformed into simple parameters for the algorithm. The goal is to minimise the loss of information important to the algorithm while also reducing the number of input parameter variables as much as possible.

Each task type has a unique time cost associated with it which is then adjusted with various factors. Different house types, floors and other environmental conditions affect on how much time is required per task. The type and size of item to be installed into the house also plays a role. Also the type and age of the house affect the time requirement. However, these factors are left out of the scope of this project as it would require additional statistical data regarding the time intensiveness of the features.

The number of technicians is fixed to two by the client company's standard procedure, so the number of workers does not play a role.

Once the initial parameters to convert the job information into a single time requirement have been determined, it is necessary to pay attention to how well they correspond with reality and then adjust them to get better accuracy. This is an iterative process, where future field data can be used to improve the time requirement estimate given to different tasks and installation items.

For the purpose of this program, it is estimated by the client company that it will take 1 hour to install one item and about 15 minutes to do the paperwork with the customer regardless of the amount of work at the site. These initial numbers are crude estimates and should be adjusted for more accurate time requirement evaluation as more detailed information becomes available.

The time required to do a single job item can be ideally calculated with the following formula:

$$\sum_{c \in I} (c \times d \times e) + 15min, \tag{4.1}$$

where I contains the time costs (c) of the items to be installed at the target, d is the difficulty factor of the house type and e is the efficiency ratio of the technician.

Chapter 5

Implementation

The goal of this thesis was to create a program which automatically generates route data of a pool of jobs at various locations. This chapter describes the process of determining the best tools for the development of the program, the implementation progress itself and the final structure of the program.

5.1 Technical decisions

Technical decisions made early on can greatly affect the final results. Choosing the wrong technologies, for instance, can cause severe setbacks, additional work and in the worst cases completely impede progress.

5.1.1 Generating the routing data

To create the all-to-all distance matrix necessary for solving the VRP, it is necessary to use a map routing API which computes the distance of the best route between two nodes and estimates the corresponding travelling time. Since there can be a lot of locations for the algorithm, it is necessary to group nearby targets into one to limit the number of queries made to the API. Even if the routes are generated per technician (i.e. it is necessary to only generate the all-to-all cost matrix from all the jobs assigned to the technician at hand), it is still necessary to make thousands of queries to the map routing API.

The grouping was performed by picking a random node and assigning its coordinates to other nodes within a 3 kilometre radius. This is then repeated for the remaining untouched nodes until no more merges are possible. While better results could be obtained by picking the merger nodes according to some rules, this simple algorithm is effective enough. The inaccuracy caused by the maximum of 3 kilometre disparity between the real location and the one used for the algorithm should not usually cause big differences in travel time, though with natural formations such as rivers or lakes, this is a possibility.

The number of queries to the map routing API was lowered by 30 - 50 % by this grouping depending on the test data used. In the number of queries, this provided savings ranging from 1300 to 4000 queries. The more locations involved in the total data, the higher that savings percentage was. This result seems reasonable, as the probability of an additional target being located near an existing one increases as the number of targets increases. The total number of queries made ranged from about 2000 to 4500.

I picked MapQuest[3] to be the routing service provider. They provide 15000 queries per month for free, and that number was high enough for the purpose of this thesis. MapQuest supports getting detailed route information, but for the purpose of this project, only the travelling time value between two locations was used. The routing service provider is easy to replace in the final implementation, as the replacement only requires altering the query made to the service and parsing of the response, so there is no lock-in on a single provider. I tested the quality of MapQuest's routing results by comparing them to the results of other similar services, such as Google maps and Bing maps. The variations between the results were minimal to the point where it was safe to assume that all of them provided good results.

5.1.2 Language requirements

The most important aspect of choosing which language to use depends on the routing algorithm libraries available for the language.

Because the algorithm is CPU-intensive, the more low-level languages take precedence over the potentially slower high-level languages. This is not a deciding factor, however, as many of the high-level languages are still sufficiently fast enough for this purpose. Likewise, even if a program written in pure C, for example, might perform better, the increased development time and reduced maintainability are more significant issues than a slightly reduced performance of a Java program.

Due to the performance-centric nature of the algorithm, it is important that the language chosen can be profiled to see which parts of the algorithm are the most resource-intensive. Though most modern languages fulfil this requirement, it is worthy of mentioning.

As the library I deemed most suitable for this project was Java-based and since there were no notable downsides to using Java, I used it as the language for the program.

5.1.3 Choosing the library to use

The important criteria for choosing the libraries to use are performance, solution quality and suitability to the parameters specific to this project. Ease of use and the quality of documentation are also significant factors, as well as how well the library is being maintained currently.

My main strategy when picking the library to use was to read the API documentation to see if it supported the functionality I need. Since there are so many different variations of the VRP, there are also numerous types of libraries, most of which were specialised for a specific kind of VRP. If a library was aimed for a different purpose it became apparent quickly. Then I examined how well the library has been maintained and how clear the documentation is.

To see if a library was suitable for my requirements, I looked at the API provided by the library and determined if the API supported the functionality I required. If there was no built-in support, I looked how well the library supported writing additional custom constraints to be used in the route generation. I also considered the number and variety of usage examples as one criterion for the decision, as it is a clear indication whether the library supports the various use cases required in the context of this thesis. This method of choosing a library eventually led me to pick a Java-based library Jsprit by Stefan Schröder[2]. In subsection 5.1.5 I list other libraries I researched for this task.

5.1.4 Jsprit routing library

To find solutions to VRP, Jsprit employs the ruin and recreate method described in section 3.3.6. Java is a language I am very comfortable with, so my personal preference can also be considered as a deciding factor.

Jsprit supports many different types of VRP. It allows the programmer to define multiple types of vehicles each with different capacities for various types of items. Each vehicle can have associated fixed and running costs. The fixed cost is applied for simply using the vehicle (this represents, for example, buying the said vehicle) and running costs signify gasoline usage, service costs (averaged over driving distance) and so on. Additionally, the maximum speed and waiting time costs can be specified for vehicles. For each vehicle, the starting and end location of routes can be set. This essentially means that Jsprit is capable of solving complex MDVRPs.

The jobs of routes are called services, and each service is associated with a location and a time cost. Locations can contain time windows. The travelling costs (time and distance) between locations can be set using costs matrix.

Jsprit limits the computation time by providing a setting for iterations, with each iteration consisting of one phase of ruining and recreation.

If the built in constraints found in Jsprit are not enough to satisfy the programmer's need, the library also supports additional constraints using an API provided by the library. Jsprit then uses these constraints during route creation to check whether the newly created route is valid.

This initial choice I made turned out to be a good one and there was little reason to explore other options. The performance, ease of use and quality of results turned out to be as good as I hoped for. Currently the library is mostly maintained and updated by a single person with some random developers providing occasional patches.

Jsprit has built-in xml generator for outputting the solution's route data. It also allows accessing the solution data programmatically, making it possible to more directly use the information for custom outputting or manipulation of the results. Jsprit also produces rough visualisations of the best solution it has found, as can be seen in Figure 5.1.

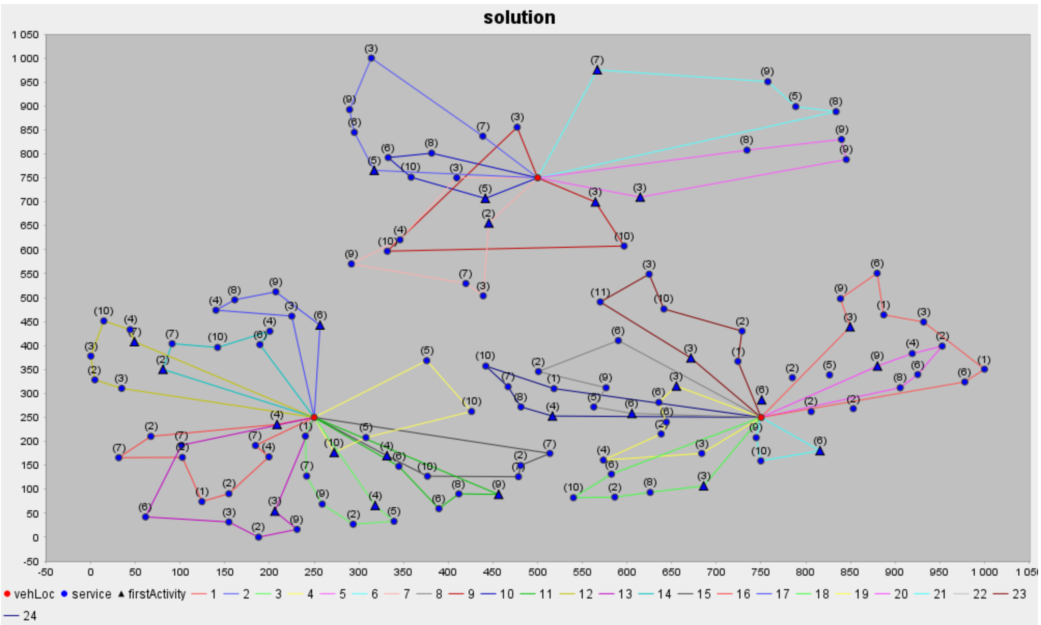

Figure 5.1: An example of the graph output produced by the library. Data unrelated to the business case.

5.1.5 Other VRP libraries

I looked into several libraries also made for solving VRPs. This is far from an exhaustive list of the various free libraries available, but might give some pointers for people who are also interested in VRP solving. These were picked according to the number and quality of links to them on the web, translating into prioritised search engine results.

Other libraries I researched were:

- VRPH: An extensive open source C++ library for solving VRPs. [7]
- Google's Operations Research tools: A collection of various tools for solving many different types of combinatorial problems. [1]
- Open-VRP: A lisp-based open source library for VRPs. [4]
- VROOM: A C++-based VRP library. [6]
- OptaPlanner: An open source library capable of solving various combinatorial problems, including VRPs. [5]

5.2 Development and structure of the program

I did the development using the Netbeans IDE. The Jsprit library was the only external library I needed during the development and it was fetched using Maven, a software project management tool.

The development of the program was a straightforward process. I first used some mock data to try out the Jsprit library and see how it performs with some simple fixed problems. Once I was convinced it was functional and expandable piece of software, I proceeded to apply the full business logic, because I had enough experience related to the rest of the project's aspects to know that I would be able to implement them without insurmountable issues.

To ensure future maintainability, the program was designed to be modular, so if some part of the program needs to changed, the operation will be as easy as possible. The modules of the program are as follows:

1. Routing module, transforming address data into a travelling cost matrix.

2. Job resource demand calculator, abstracting the job resource and man-hour requirements into numbers.

3. The algorithm module, using the travelling cost matrix and job requirements to produce route data.

4. Results visualiser, displaying the results data in a more human-readable form.

5. Results storing module, converting the results into data suitable to be stored in the client company's database. (not implemented)

5.2.1 Routing module

The input of this module is the customer data including upcoming jobs and their locations, and the output is a matrix containing the travelling times between these locations. To determine the travelling times, the module makes queries to the map routing API, as demonstrated in subsection 5.1.1.

As the many-to-many networks require a lot of queries and a high number of queries leads to higher costs, the module is forced to make some optimisations. Nearby targets are forced into the same location because the difference in travelling times would be minimal. Another benefit of reducing the number of queries is performance. Currently most of the program's time is spent querying the travelling times from the routing API.

When receiving a list of job targets, this module iterates through them and creates new locations if the new target's location is too far from existing ones to be merged to them. Afterwards each target knows the id of the location corresponding with itself.

With this reduced set of n locations, the module makes $\frac{n^2}{2}$ queries to the routing API. The paths are considered symmetrical, as the travelling time is rarely significantly different depending on which way it is travelled.

All the routing information is cached after fetching it to avoid having to fetch the same data on successive runs if the same input data is used. This also makes consequent uses of the program faster.

5.2.2 Job resource demand calculator

The estimation as for how long a job will take is currently very coarse. Each installation item takes one hour and each job target has a 15-minute overhead during which contractual business with the customer is done before the work can begin.

If the job duration added to the back and forth travelling time between the target and the depot is greater than 8.5 hours, the program considers the job too large to complete in one route and splits it. It then calculates the maximum number of installation items that can be handled in one day at the target and creates a new "sub-target" which contains the rest of the items. This process is repeated if the new target is again too large to fit on a single day's work schedule.

For the sake of the algorithm, the sub-targets are considered the same as the main targets. However, when outputting solution information, it is important to know which is the sub-targets' main job target item.

5.2.3 The algorithm module

The algorithm module takes the target information and location matrix as input and outputs the solution information ready to be used for whatever purpose.

Using Jsprit's API, the technician's home is set as the depot and the job targets as the nodes to be visited. The travelling costs matrix is then given to Jsprit. Because Jsprit does not natively support having a maximum route time constraint, I created one using its custom constraint API.

5.2.4 Results visualiser

Visualisation of the results is currently limited to the Jsprit library's own plotting functionality, and custom textual output of the routes. By themselves the plots are not helpful at all. For them to be useful, the user would have to see the graph nodes on a map to better see the shape of the routes. The connecting lines between the nodes would also be more informative if they followed the road network instead of being straight lines from one node to another.

To get at least some kind of useful human-readable output, the program prints the route information of each individual technician to a separate file. The routes do not have any specific dates assigned to them, but rather contain information as to what is the earliest possible date they can be done, considering that there needs to be a few days between the delivery of the installation items and the beginning of the job.

As many job targets are too large to fit in one route because they would make the working day too long, they are split into smaller ones ("sub-targets"). These sub-targets contain a reference to the main targets. With these kinds of targets, the number of installation items to do on the route is less than the total number of installation items.

The actual textual output is in Finnish, but here is a sample of the output translated into English and personal information replaced with placeholder data:

```
Technician: Remontti Reiska, Silmupolunsuo 25, 38100  Hämeenkyrö
Total number of targets: 9
Total number of targets including sub-targets: 11
Total number of routes: 6

NEW ROUTE
duration: 454 min
earliest date of installation: Sat Nov 05 00:00:00 EET 2016:
  NEW TARGET:
    target id: 202;
    date of delivery: Tue Nov 01 00:00:00 EET 2016;
    time (min): 26 - 101;
    address: Kalmankatu 6, 33330, Tampere;
    number of installation items: 1, total: 1;
  NEW TARGET:
    target id: 70;
    date of delivery: Fri Oct 14 00:00:00 EEST 2016;
    time (min): 157 - 292;
    address: Aliliidontie 52, 34260, Tampere;
    number of installation items: 2, total: 2;
  NEW SUB-TARGET:
    target id: 272, Main target's id: 140;
    date of delivery: Fri Oct 14 00:00:00 EEST 2016;
    time (min): 338 - 413;
    address: Kalansilmänkuja 2, 33700, Tampere;
    number of installation items: 1, total: 13;
------------------------
NEW ROUTE
duration: 456 min
earliest date of installation: Tue Oct 18 00:00:00 EEST 2016:
  NEW TARGET:
    Target id: 140, date of delivery: Fri Oct 14 00:00:00 EEST 2016;
    time (min): 40 - 415;
    address: Tuomiontie 66, 33700, Tampere;
    number of installation items: 6, total: 13;
...
```

5.2.5 Results storing module

The results are currently only outputted to the user. In order to reach bigger gains in usability and efficiency, the results could be processed automatically. The program could assign specific dates to the routes and store the information to a database. In the optimal case, the routes are generated automatically as new customers are added to the registry, and the generated routes are only approved by a human before put into use. Even the last step could be skipped once the reliability of the route generation reaches a sufficiently good level.

Chapter 6

Evaluation

The most obvious aspect to improve in the program is usability. Currently the tool requires manual fetching of the input data in json format from the client company's system by adjusting the API address url to get the data from a specific date range. This data has to be then saved to a file and put to the program folder.

The program has to be run from command line or alternatively by double clicking on the .jar file if the operating system supports this behaviour. In the latter case, the program does not show any progress information to the user.

6.1 Results

the program was run on a PC running Windows 7 with an Intel 3,4 GHz quad core CPU and 16 GB of 1,6 GHz RAM. The Java version used was 1.8.0_66.

Running time depends greatly on whether the distance information has been cached or if it has to be fetched from MapQuest's routing API. A sample run with 574 job targets split between 42 technicians and an iteration count of 256 showed that without any cached results, the program took 57.7 seconds to run, and on a second go with cached distance data, the program took 9.2 seconds to complete. Profiling the program indicates that most of this time is spent in the library solving functions. About 15 % of time was spent serialising and de-serialising the cached route data and the rest was spent on the actual problem solving.

Because the actual routes are very short (71 % of routes consist of only one job target, 21 % of routes have two targets, 7 % have three and mere 1 % of routes include 4 targets), the inherent benefits of the Jsprit's implementation of the ruin and recreate method are not apparent. A simple savings algorithm probably could reach almost similar solution quality. In the context of this project this does not matter, however, as an already existing library provided the advanced functionality with no extra cost.

The number of iterations used for the solving did not affect the results much. The total time cost of all the routes was used as the measurement of solution quality. As figure 6.1 shows, the benefit of the ruin and recreate algorithm is minimal. When only the driving time is observed, the initial solution with 0 iterations is only 3.1 % slower than the solution reached with 4096 iterations. When the working time at targets is also taken into account, the difference is slightly less than 1 %. The computational cost of a single iteration is constant. Doubling the number of iterations also doubles the solving time. In figure 6.1 the solving time is doubled on every step. It can be seen that the benefits of increasing the number of iterations in the algorithm gradually lower until they become non-existent at 2048 iterations.

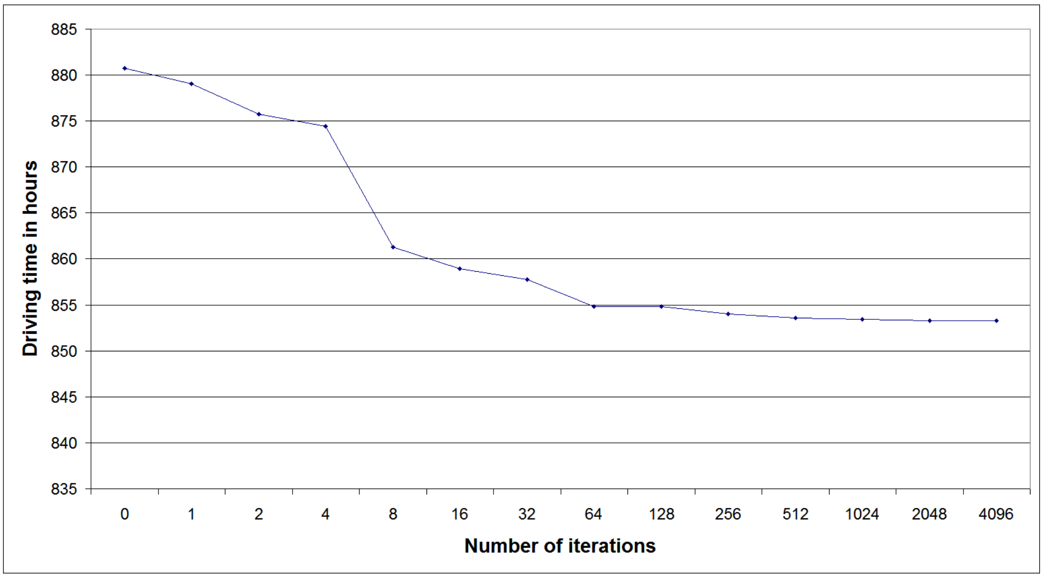

Figure 6.1: The total driving time cost of all routes with varying numbers of iterations. With 0 iterations the solution is just the initial solution created by jsprit.

The small differences should not be downplayed, either. In this case, the difference between the very worst and the best solution translates to 27.5 hours. Saving that many hours per month can be a sizeable savings for a company. However, the difference between 4096 and 64 iterations is only 1,5 hours in total. Finding the optimal spot between performance and solution quality lies somewhere between those two values, in my opinion. 64 iterations is an operation fast enough that it practically does not break the user's workflow, while 4096 iterations causes a very noticeable delay. With the test data and cached location data, the total run time with 4096 iterations was 2 minutes and 26.7 seconds, while with 64 iterations it was 4.5 seconds.

If the routes were produced company-wide and then assigned to the technicians afterwards, the benefits of the algorithm would become more apparent. Currently there are about 10-25 job targets per technician during the one month planning period, leaving a relatively low number of possible combinations to try out. However, using the company-wide system would put more strain on the travelling cost matrix generation. Instead of multiple small all-to-all networks, there would be one big one, requiring additional optimisations to avoid having to make tens of thousands of queries to a routing API per solving.

6.2 Customer's perspective

The customer was able to run the program and make it produce the expected output. The customer was overall pleased with the program's output and found that it could be used for planning routes. The quality of the routes was similar to that of routes produced by manual labour. This was determined by counting the number of days needed to handle a set of jobs.

This program, a preliminary demo of what is possible to do, caused interest in the client company to continue the development the work planning procedures used in the company and automate them further. The next step would be optimising the way the job targets are assigned to the various technicians. This will likely provide much more optimal results, as inefficient allocation of jobs might make it impossible to do the routing process itself well. Automating the whole process would also grant the possibility of objectively comparing different approaches to work planning, as one method will provide a less travelling overhead than another and the results are immediately measureable.

The biggest downside found by the client in the current program was that there can be quite a large gap in the earliest possible installation date between the jobs on a single route. This is because currently the program does not care about the earliest installation date except at results outputting. Aside from this issue, the client did not find problems with the program. A crude workaround solution is to simply use input data with more restricted date filtering. If only job targets aimed to be done within a certain week are included in the input, there cannot be large gaps between the earliest installation dates of targets in a route.

The customer also would have wanted ordering of the routes by dates. Currently the routes are displayed in random order as the routes can be assigned to any date. The scheduling should be designed so that routes that visit the same areas should be on subsequent days.

6.3 Future development

Probably the biggest room for development lies in usability. Using the program for a specific installer or automatically fetching the input data according to user preference would make the program's usage a lot more comfortable. The lack of user interface makes the program potentially confusing for a non-expert.

6.3.1 Business logic

Currently the program uses very rough estimates for determining the time required at a worksite. 1 hour per item and a fixed 15 minute overhead work fine for testing purposes, but more detailed information would provide better and more optimised routes. This would be a rather simple task that would be achievable with the usage of configuration files, for example. This would also make it relatively simple to make the program support new types of installation items and more specifically categorise building types.

The user could enter the average time requirements per installation item type based on past experience on worksites where the item has been installed. The effect of different house types and the floor number of the operation could be taken into account. Furthermore, the relative performance of technicians could be listed, allowing the scaling of the time requirement to get as accurate estimation as possible.

As the program provides only routes for the future, it does not take into account any potential setbacks, such as sick leaves, car malfunctions and so on. As the routes themselves outputted by the program do not have any specific date, but rather only an earliest possible date, it could be thought that a sick leave, for instance will only require postponing the routes in calendar.

The client company's request for following the time windows more strictly is also a clear way to improve the results provided by the program. Considering that the jobs should start as soon as possible after the construction materials have been delivered to the worksite, the algorithm should try to group targets with near delivery dates onto same routes. Currently the algorithm can create routes such that the delivery of the installation materials of the earliest job is weeks before the delivery of the latest job, resulting in a scenario where the materials will lie unused at the worksite of the earlier job for weeks. This would lead to reduced customer satisfaction.

One way to address the aforementioned issue would be to assign a penalty to routes with a big variety in the earliest installation dates. This would result in the algorithm favouring routes where the jobs' earliest installation dates are grouped more tightly together. This would require implementing the aforementioned additional feature to the Jsprit library which might be a lot of work.

Another way would be to group the installation jobs by their earliest installation dates. This would be easy to implement. However, it would be difficult to tell if a certain job would fit better in the next group, resulting in potentially unnecessarily suboptimal routes. This could be countered by running the algorithm multiple times and adjusting the date ranges of the groups between runs. Then the best routes could be picked from the results. However, this would require more complex results analysis due to jobs being found in multiple routes.

6.3.2 Output

The results output is currently just textual representation with the routes and the routes' job locations indented for clarity. A proper visual representation of the routes and suggested dates for them would make benefitting the results much easier. The Jsprit library supports exporting the results in xml format already, so producing a custom machine-readable output is not necessary, unless using a simpler schema or some format other than xml is required.

Performance of the program could be improved by solving the separate technicians' VRPs in parallel. Considering how quick the actual solving process was, I did not see much point in implementing this kind of concurrency to the program.

6.3.3 Programming aspects

The structure of the program would need refactoring, as it was developed for research purposes with a lot of trial and error. Though the codebase is very small, less than 1000 lines, restructuring and otherwise cleaning the code is necessary if this software is to be used in a bigger context.

The program's error handling capabilities are also very limited. Most errors cause the program to crash or produce bad or non-existent results without giving an understandable error message. This is not an issue for research use, but real world usage would definitely benefit from a more solid error handling.

All testing on the program was done by hand. Automated testing would make future development a lot easier.

6.4 Retrospective

Overall, I feel like this project was a success. I reached the goals I had set beforehand and firmly believe that the future development of the program is not only possible, but would increase the amount of potential benefit in work planning.

The implementation process went fine without any unsolvable show stoppers which would have required doing workarounds for the desired outcome.

With highly versatile libraries such as Jsprit, the main problems to tackle involve adapting the business logic to the algorithm's framework.

Chapter 7

Conclusions

There are so many various trades whose operation or parts of it can be abstracted to a VRP that there is much room for improvement in terms of automation in our society. It is no wonder that so much research has been done on the subject. Any sufficiently large company will probably benefit from applying it to various processes unless their everyday problems have absolutely nothing to do with VRPs.

The basic requirement for benefitting from this kind of work planning is having some sort of formalised database to which customer information is being stored. Then it is possible to transform that data and use it to solve the VRP associated with the business at hand. Naturally there are many trades which do not have anything in common with VRPs, but many aspects around the basic necessities of modern human life heavily involve concepts related to VRPs. Delivery of food and other goods from producers to intermediate storages and furthermore to markets are a prime example, and if grocery deliveries straight to the doorstep becomes more common, there will be yet another common application for VRPs.

Even if in this specific case there was not much of a benefit in using an advanced solver, there was no downside to it either. The main benefit in this case was simply the automation process and providing a framework upon which future development can be based. A job which once required human labour has now been partly replaced by a machine. Furthermore, if the client company takes this work planning automation further, it is entirely possible to reach a point where only on specific exceptional circumstances does the process require any human interaction at all.

Based on my experiences with this project, I am certain that automating this kind of work planning would bring great benefit to a lot of companies. Though businesses are different and each has its own unique set of requirements, versatile libraries such as Jsprit could probably be used for their purposes. The biggest work would be adapting the business model into an algorithmic format. A general solution would work only in select cases where the business model of the companies is automatically translated into a certain variation of a VRP. Otherwise customisation is needed, meaning that for small scale operations, automatisation might not be cost effective.

The open source library Jsprit continues the good trend of increasing number of free tools being available to the general public. It is a good example how free software can be used to make human societies more efficient. This will eventually and ultimately lead to humankind having more free time as workforce is replaced by machines. Assuming that this redistribution of work is taken into account on a political level (a huge assumption that does not hold true so far, admittedly), humankind can only benefit from this development.

Bibliography

[1] Google OR Tools. https://github.com/google/or-tools. Accessed: 2016-10-07.

[2] jsprit. https://github.com/graphhopper/jsprit. Accessed: 2016-10-07.

[3] MapQuest. https://developer.mapquest.com/. Accessed: 2016-10-07.

[4] Open-VRP. https://github.com/mck-/Open-VRP. Accessed: 2016-10-07.

[5] OptaPlanner. https://www.optaplanner.org. Accessed: 2016-10-07.

[6] VROOM. https://github.com/VROOM-Project/vroom. Accessed: 2016-10-07.

[7] VRPH. https://sites.google.com/site/vrphlibrary/. Accessed: 2016-10-07.

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

[9] Bell, J. E., and McMullen, P. R. Ant colony optimization techniques for the vehicle routing problem. *Advanced Engineering Informatics 18*, 1 (2004), 41–48.

[10] Blakeley, F., Argüello, B., Cao, B., Hall, W., and Knolmajer, J. Optimizing periodic maintenance operations for schindler elevator corporation. *Interfaces 33*, 1 (2003), 67–79.

[11] Cordeau, J.-F., and Groupe d'études et de recherche en analyse des décisions (Montréal, Q. *The VRP with time windows.* Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

[12] Dantzig, G. B., and Ramser, J. H. The truck dispatching problem. *Management science 6*, 1 (1959), 80–91.

[13] Desaulniers, G., and Groupe d'études et de recherche en analyse des décisions (Montréal, Q. *The VRP with pickup and delivery.* Montréal: Groupe d'études et de recherche en analyse des décisions, 2000.

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

[15] Flood, M. M. The traveling-salesman problem. *Operations Research 4*, 1 (1956), 61–75.

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

[17] Gendreau, M., Laporte, G., Musaraganyi, C., and Taillard, É. D. A tabu search heuristic for the heterogeneous fleet vehicle routing problem. *Computers & Operations Research 26*, 12 (1999), 1153–1173.

[18] Ghoseiri, K., and Ghannadpour, S. F. Multi-objective vehicle routing problem with time windows using goal programming and genetic algorithm. *Applied Soft Computing 10*, 4 (2010), 1096–1107.

[19] Gillett, B. E., and Miller, L. R. A heuristic algorithm for the vehicle-dispatch problem. *Operations research 22*, 2 (1974), 340–349.

[20] Hassanzadeh, A., Mohseninezhad, L., Tirdad, A., Dadgostari, F., and Zolfagharinia, H. Location-routing problem. In *Facility Location.* Springer, 2009, pp. 395–417.

[21] Karaboga, D., Gorkemli, B., Ozturk, C., and Karaboga, N. A comprehensive survey: artificial bee colony (abc) algorithm and applications. *Artificial Intelligence Review 42*, 1 (2014), 21–57.

[22] Kontoravdis, G., and Bard, J. F. A grasp for the vehicle routing problem with time windows. *ORSA journal on Computing 7*, 1 (1995), 10–23.

[23] Laporte, G. What you should know about the vehicle routing problem. *Naval Research Logistics (NRL) 54*, 8 (2007), 811–819.

[24] LAPORTE, G., GENDREAU, M., POTVIN, J.-Y., AND SEMET, F. Classical and modern heuristics for the vehicle routing problem. *International transactions in operational research 7*, 4-5 (2000), 285–300.

[25] MONTOYA-TORRES, J. R., FRANCO, J. L., ISAZA, S. N., JIMÉNEZ, H. F., AND HERAZO-PADILLA, N. A literature review on the vehicle routing problem with multiple depots. *Computers & Industrial Engineering 79* (2015), 115–129.

[26] PISINGER, D., AND ROPKE, S. A general heuristic for vehicle routing problems. *Computers & operations research 34*, 8 (2007), 2403–2435.

[27] REIMANN, M., DOERNER, K., AND HARTL, R. F. D-ants: Savings based ants divide and conquer the vehicle routing problem. *Computers & Operations Research 31*, 4 (2004), 563–591.

[28] SALHI, S., IMRAN, A., AND WASSAN, N. A. The multi-depot vehicle routing problem with heterogeneous vehicle fleet: Formulation and a variable neighborhood search implementation. *Computers & Operations Research 52* (2014), 315–325.

[29] SCHRIMPF, G., SCHNEIDER, J., STAMM-WILBRANDT, H., AND DUECK, G. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics 159*, 2 (2000), 139–171.

[30] SOLOMON, M. M. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations research 35*, 2 (1987), 254–265.

[31] SZETO, W., WU, Y., AND HO, S. C. An artificial bee colony algorithm for the capacitated vehicle routing problem. *European Journal of Operational Research 215*, 1 (2011), 126–135.

[32] TUZUN, D., AND BURKE, L. I. A two-phase tabu search approach to the location routing problem. *European journal of operational research 116*, 1 (1999), 87–99.

[33] YU, B., AND YANG, Z. Z. An ant colony optimization model: The period vehicle routing problem with time windows. *Transportation Research Part E: Logistics and Transportation Review 47*, 2 (2011), 166–181.