Implementation of a Home Health Care Routing and Scheduling Problem

using a learning-based savings algorithm

Institute of Production and Logistics
University of Natural Resources and Life Sciences, Vienna

Masterarbeit

Requirements for the Degree of
Master of Science

by

Christian Lübke

October 2014

Betreuung: Univ.Prof. Mag. Dr. Manfred Gronalt
Betreuung: Ass.Prof. Mag. Dr. Patrick Hirsch
Betreuung: Christian Fikar, Msc.

Declaration

I hereby declare and confirm that this thesis is entirely the result of my own original work. Where other sources of information have been used, they have been indicated as such and properly acknowledged. I further declare that this or similar work has not been submitted for credit elsewhere.

Date: Wien, October 29, 2014

Christian Lübke

Acknowledgements

Apart from the efforts of myself, the success of any project depends largely on the encouragement and guidelines of many others. I would like to express my deepest appreciation to all those who provided me the possibility to complete this master thesis.

I would like to thank my advisors from the University of Natural Resources and Life Sciences, Prof. Manfred Gronalt, Prof. Patrick Hirsch and Christian Fikar. I want to thank Patrick Hirsch for his continuous support, useful comments, remarks and engagement through the learning process of this master thesis. Moreover, I want to give my thanks for correcting my English, when needed. Additionally, I want to thank Christian Fikar for introducing me to the topic, his technical support, feedback for my questions as well as helping me with my work on the code. Furthermore, I want to thank my programming teacher Günter Kiechle. Without his lectures I would have had a much harder time working on my optimization problem.

Naturally, I want to thank my parents and friends at this occasion for supporting me over the last years of study. I hope it was not to exhausting to hear now and then some complaints, when I was not sure how I should go on or how I can motivate myself. Additionally, I want to thanks the lot of proofreaders who were sacrificing some of their free time to help me with the correction of my thesis. For that reason, I want to give my acknowledgements to Stefan Glawischnig, Anna Guggenberger, Sophia Melcher, Valentin Pesendorfer, Sandro Rauscher, Helene Roselstorfer, Sebastian Sölle and Jana Vögl.

Last but not least, I want to thank Gabor Kruchio, for helping me to push this through. It helped a lot that you raised my spirit, when I lost the energy during the process.

Zusammenfassung

Diese Masterarbeit stellt ein Konstruktionsverfahren vor, dass durch das Lernen von vorangegangenen Ergebnissen die weiterführende Lösungssuche adaptiert. Zudem, wird ein Lösungsweg vorgestellt, der dieses Lernverfahren auch für die Auswahl von Routen, im Bereich mobile Pflegedienste, adaptiert. Die in der Arbeit bearbeiteten Probleme sind ein Vehicle Routing Problem (VRP) und ein Problem, das sich mit Personentransport im mobilen Pflegedienst (DARP) beschäftigt. Das zweite Problem, wird im Rahmen eines Projektes auf der BOKU Wien behandelt. Mit dieser Arbeit wird ein kleiner Beitrag dazu geleistet, indem ein alternativer Lösungsweg implementiert wird. Für die Lösung der Aufgabenstellung wird ein modifiziertes Savingsverfahren mit einem Lernmechanismus vorgestellt. Zudem wird der Lernmechanismus für die Auswahl von Gehrouten in dem Algorithmus für das Problem verwendet, in welchem es um die Routenplanung für Pflegekräfte geht. Zu dem Lernmechanismus werden im Zuge der Arbeit verschiedene Möglichkeiten aufgezeigt, um zu lernen. Die Möglichkeiten werden aufgrund der Testergebnisse für das VRP und von Parameter-Tests evaluiert und die beste Variante wird im Anschluss daran auf das DARP angewendet. Die Ergebnisse zeigen, dass der Algorithmus bei kleinen Testinstanzen gute Ergebnisse liefert, jedoch kann er bei grö eren und komplexeren Problemen nur mehr als Konstruktionsverfahren verwendet werden. Auf dieses Verfahren aufbauend wird die Verwendung einer Metaheuristik empfohlen.

Abstract

This master thesis provides a learning-based construction heuristic and a multi-solution approach for a Home Health Care Problem. The problems tackled by this thesis are a Vehicle Routing Problem (VRP) and a routing problem dealing with transport of passengers in the home service industry (DARP). The latter problem is motivated by a project at the University of Natural Resources and Life Science, Vienna. The aim of this thesis is to contribute to this project by providing an alternative solution approach. To solve the mentioned problems, a learning mechanism modifying an extended biased randomized Savings Algorithm is introduced. Furthermore, the learning mechanism is applied to a route-selection procedure, improving the results on the basis of the knowledge obtained from former solutions. This thesis presents different variations on how to adapt the learning mechanism. Consequently, the variations are evaluated by testing different parameter settings for the VRP. To this end, the best variant is implemented for the DARP. The results show that the algorithm is capable of providing good solutions for small problem-instances. However, when dealing with larger and more complex problems, the heuristic should be used to construct initial solutions. Based on these, the application of a metaheuristic solution approach is recommended.

Contents

1 Introduction

The trends in the population development in European countries are changing the traditional patterns of care. We will face an increase in demand for *Home Health Care* (HHC) services and an increase in the number of care-dependent people (Tarricone and Tsouros, 2008). Woodward et al (2004) are showing three reasons for this development: in our society chronic illness is increasing, people who are recovering from surgery or acute illness often need treatment at home and the number of old people is rising due to higher life expectancy.

For this purpose, better routing and scheduling allows home care service providers to save costs and to be more efficient in their decisions. Moreover, HHC providers in urban areas are facing difficulties like congestions, limited parking spaces as well as staff without driver s license. Hence, it is a complicated task to optimize the routing of the nurses and at the moment it is in most cases done manually. Therefore, automatically solving a HHC problem will be an asset for the companies and relieve their daily scheduling. The solution method for that task has to find a route for nurses visiting their clients, considering a multitude of hard and soft constraints.

The work is motivated by a project at the University of Natural Resources and Life Sciences, Vienna. The project includes a major HHC provider, the Austrian Red Cross, and focuses on the conditions given in Austria. The main idea is to deliver nurses to their clients, using a bus and a professional driver. Furthermore, the possibility of walking between clients is considered for the nurses. To solve this problem, Fikar and Hirsch (2014) introduced a solution-approach using a solver-software and a matheuristic. This master thesis will provide a different solution-approach, trying to avoid the use of an expensive solver-software. Overall, the thesis should help service providers to reduce their vehicle fleets and decrease their environmental impact. Further benefits through better routing are less fuel-consumption and more efficient scheduling of the employees. Subsection 1.1 presents the algorithm and goals of this master thesis in more detail.

1.1 Problem de nition

The problem covered in this thesis is based on the work of Fikar and Hirsch (2014). In their paper they present a many-to-many *Dial-a-Ride Problem* (DARP) containing the following differences to the classical DARP:

- The objective of the work is not only to minimize the vehicle drive times but also to minimize the working time of the nurses considering their working time regulations and mandatory breaks.
- The input for the problem is a fixed number of jobs, which all have to be done. The jobs have fixed service durations, hard time windows and requirements on the qualification level of the nurses. In addition, the pickup-time for the nurses depends on when they

have been delivered to the job. This dependency between decisions makes the problem hard to solve.

- The problem allows nurses to walk between certain jobs. Hence, the transportation service does not have to visit all clients in the routing problem.

To present the problem in more detail, Fikar and Hirsch (2014) describe it as follows: the input is a set J of n jobs ($i \quad J$). Moreover, each job has a special qualification requirement $q^{\mathrm{J}}{}_i$ assigned considering the nurses abilities. The staff working for the service provider is defined as a set M of m nurses ($j \quad M$). Similar to the clients, the nurses are associated with a qualification level $q^{\mathrm{M}}{}_j$. Furthermore, a vehicle fleet K of k vehicles ($h \quad K$) is defined for the service provider. The constraint considering the qualification requirements and levels is proposed as $q^{\mathrm{M}}{}_j > q^{\mathrm{J}}{}_i$. Another factor, defined by Fikar and Hirsch (2014) is E, which should ensure employee satisfaction and deviation of qualification level (i.e. a nurse of qualification level $q^{\mathrm{M}}{}_j$ can perform jobs of $[q^{\mathrm{M}}{}_j - E, q^{\mathrm{M}}{}_j]$). Further, the number of downgrades S is limited, which implies that an overqualified nurse does jobs with a lower qualification requirement. The jobs themselves have a hard time window $[e_i, l_i]$, consequently, e_i being the earliest and l_i the latest allowed starting time. Subsequently, the jobs have a service duration d_i and a service start time denoted by B_i. The arrival time for the vehicle and nurse are defined with $A^{\mathrm{M}}{}_i$ and $A^{\mathrm{K}}{}_i$.

The problem definition is on a graph $G = (V, A)$. Moreover, each job can be a potential pickup and delivery location. Therefore, the vertex set $V = \{v_0, v_1, \ldots, v_{2n+1}\}$ is divided in a set $D = \{v_1, \ldots, v_n\}$ for the delivery vertices and a set $P = \{v_{n+1}, \ldots, v_{2n}\}$ for the pickup vertices. Being a DARP, all jobs have to start and end at the depot, which is denoted as v_0 and v_{2n+1}. The arcs $(i, j) \quad A$ consider a walking time $t^{\mathrm{M}}{}_{ij}$ and a driving time $t^{\mathrm{K}}{}_{ij}$. The vehicle have a load of Q_i being at least one, because of the vehicle driver and a maximum capacity C. The time a vehicle can operate is not limited, because it is assumed that there are multiple drivers available. Furthermore, each vehicle can have multiple tours.

Additionally, conditions for the nurses are: (i) maximum working time H; (ii) maximum working time without a break R; (iii) the time for a break r; (iv) W is representing the maximum waiting time between each pickup and delivery; (v) maximum walking duration between two jobs F; (vi) maximum walking between each pickup and delivery U; (vii) and a parameter for the maximal detour between pickup and delivery L . For more information on the constraints refer to Fikar and Hirsch (2014).

The authors introduce a solution approach over two stages which is shown in Figure 1.1.

In Stage 1, *walking-routes* (WR) are created. Promising walking-routes are selected over set-partitioning, to create a set of feasible walking-routes. In the second stage, numerous initial solutions are provided by a construction heuristic. Therefore, Fikar and Hirsch (2014) use an extended *biased randomized savings heuristic* (BR-CWS). The initial-solutions do not have to be feasible. To improve the quality of the solution, the metaheuristic, unified Tabu Search, is implemented at the end of Stage 2. After a defined runtime of the metaheuristic, the WR are tested for small changes (e.g. merging different WRs) to further improve the solution.

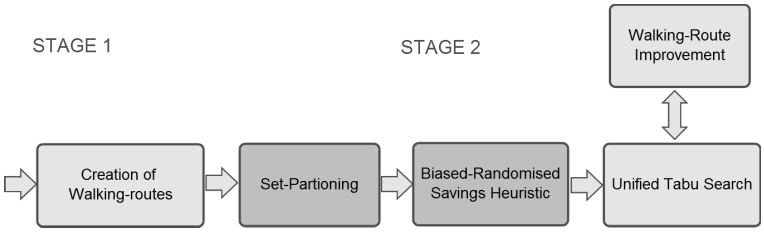

Figure 1.1: Matheuristic by Fikar and Hirsch (2014)

The aim of this thesis is to provide a multi-solution approach that replaces the currently used construction heuristic. The heuristic should provide an adaptable algorithm which is easy to implement and provides good solutions in reasonable computation times. For this purpose, the writing introduces a construction-heuristic, based on the Savings Heuristic by Clarke and Wright (1964) and the Biased Randomization Approach by Juan et al (2013). The heuristic should improve the biased randomized Savings Heuristic by adding memory capabilities. Furthermore, the selection of WR in Stage 1 of the algorithm provided by Fikar and Hirsch (2014) should be processed without using an expensive solver-software. To this end, the idea of using memory capabilities is also used for the selection of WR at the beginning of the process. Figure 1.2 provides an overview of the changes that are applied in this master thesis.

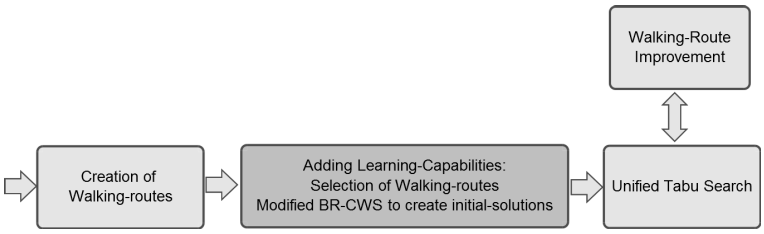

Figure 1.2: Changes to the algorithm provided by Fikar and Hirsch (2014)

In short, the selection of a set with feasible WR via set-partitioning is replaced with a random-based selection using learning capabilities. The former used BR-CWS is extended by memory- and learning-abilities to improve the initial-solution for the algorithm.

This heuristic is not only implemented for the problem definition Fikar and Hirsch (2014) are working on, but also for the test-instances Christofides et al (1979) are providing for different *Vehicle Routing Problems* (VRP). It is easier to test the algorithm and parameter-setting on test-instances which have less constraints and restrictions.

Accordingly, the work should broaden and deepen knowledge in the research area of the VRP and DARP as well as respective solution approaches to solve these problems.

1.2 Structure

The outline of this thesis is as follows. In Chapter 2, the problems related to this paper will be discussed, which are the HHC problem, the VRP and the DARP. Additionally, this chapter will show the solution methods that can be used to solve the problems.

Chapter 3 will present the solution approach. The methods used to solve the problem are modified versions of the Savings Heuristic combined with Biased Randomization. The different versions of the heuristic will be implemented in *C++*. At the end of the chapter, the modifications used on the Savings Heuristic will be introduced in detail. Furthermore, this chapter will present the method used to select WR without using a solver-software and set-partitioning.

In Chapter 4, the different modifications for the Savings Heuristic will be tested. At first, the algorithm is tested on the test-instances for the VRP. The chapter starts with a description of the test-instance. Furthermore, the parameters for the algorithm will be looked at in detail. Consequently, the results provided are shown at the end of the section. Additionally, the same procedure is performed for the DARP including the procedure for the selection of WR.

Chapter 5 concludes the paper with a summary of the findings of this thesis and an outlook to possible future work.

2 Literature

This chapter gives an overview of the basic problems related to the work in this master thesis. Therefore, Section 2.1 contains information to the related work done on HHC problems over the past years. The following sections, Section 2.2 and Section 2.3, present a general outline to the VRP and the DARP. Section 2.4, the last section of this chapter, introduces the solution methods to solve the previously described problems.

2.1 Related Work in HHC

The Home Health Care Routing and Scheduling Problem is a growing research field. It is catching more and more attention, because the number of care-dependent people it is expected to rise over the next years (Tarricone and Tsouros, 2008). Reasons for that are changes in social and cultural behavior, as well as changes in the environment we live in. Furthermore, considering that expected increase for HHC, the limited budgets of the government are playing an important role (Bräysy et al, 2007). Hence, it has become an interesting field of research and there has been a lot of work done on this topic. Therefore, this chapter will give an overview of this research field.

Mankowska et al (2014) describes the HHCRSP as follows. In many situations, providing care for care-dependent people is shifted to home health care companies. For this work, the companies employ a versatile staff, including members with different qualification- and education levels. Moreover, the staff may differ in working time regulations, or in means of transport. The problem itself can be compared to a TSP. On the contrary, to plan the route for each caregiver, different qualification levels and personal needs have to be considered. In addition, the care-dependent people have their own expectations to, which complicates it to find a good solution. Accordingly, it is valuable to provide HHC providers with methods to improve their work and develop sustainable concepts (Rest et al, 2012).

Begur et al (1997) were pioneers of the research-field of Home Health Care. They implemented a decision support system for the home health care problem with the aim to minimize the total travelling time.

Cheng and Rich (1998) presented an approach with a *mixed-integer linear programming model* (MILP). In comparison to this master thesis, they did not differentiate between the qualification-levels of the working-staff. They differentiated between full-time and part-time nurses. The optimal schedule of their solution is minimizing the amount of overtime and part-time working as well as the total distance traveled.

An approach introduced by Bertels and Fahle (2006) is the software *PARPAP*. They combined *constraint programming* (CP) with *linear programming* (LP) and metaheuristics. The metaheuristics they used are Simulated Annealing and Tabu Search. They create an initial solution via CP and improve this solution using metaheuristics.

Bräysy et al (2007) presented a case study to the routing problems, considering home care scheduling, transportation services and delivery of daily meals. For their studies they

relied on Finnish data and presented as outcome that there is a high potential in applying optimization software to save costs in communal routing problems.

Laps Care is a decision support system developed by Eveborn et al (2006) and introduced to help the planners of home care services in Sweden. The system includes a variety of elements (e.g. database, maps) and uses a repeated matching algorithm for the solution method.

A scheduling problem with a homogenous vehicle fleet was proposed by Bredström and Rönnqvist (2008). They describe a VRP with time windows and additional temporal constraints. These constraints are for synchronization and applying a priority for customer visits. The problems are solved with an optimization based heuristic, which is leading to good solutions in short time limits.

The HHC is generalized by Kergosien et al (2009) to a multiple traveling salesman problem with time windows. Adding additional constraints; they solve it with an *integer linear programming formulation* (ILP). However, they were not able to solve real size instances with this approach, but instances up to 40 services only.

Trautsamwieser and Hirsch (2011) have developed a model formulation and a solution approach with the metaheuristic Variable Neighborhood Search for optimizing the daily scheduling of nurses. The problem definition is similar to the problem definition for this thesis, but their approach was under the assumption that each nurse uses a separate vehicle.

The authors Rasmussen et al (2012) modelled the problem as a set partitioning problem, adding additional constraints. They generalize the problem as a *vehicle routing problem with time windows* VRPTW. They introduced an algorithm reaching the solution using a branch-and-price approach.

Hiermann et al (2013) presented a two-step approach to a real-world multimodal home-healthcare scheduling problem. They generate an initial solution with a CP based approach and improve it with one of four metaheuristcs. Furthermore, they compare the different approaches and discuss the performance on their experimental setup.

Implementing a two-phased solution approach based on *Tabu Search*, Rest and Hirsch (2013) solved a real world sized instance within reasonable computation time. For the transport mode they chose to rely on public transport. Therefore, they created a route-network with multiple different vehicles.

Mankowska et al (2014) have proposed a solution method for interdependent services with individual service requirements of the patients. The method takes also into consideration that there are clients that need services that must be performed by two caregivers.

To summarize, a lot of work has been done on HHC, and the examples above are just a small outline of this subject. However, the work which has been done already, does not apply for the problem definition in the paper provided by Fikar and Hirsch (2014). Accordingly, the work in the previous papers can not be applied directly for this master thesis. The reason for that is, that this thesis is partially basing on the same problem definition as in Fikar and Hirsch (2014).

2.2 The Vehicle Routing Problem

The aim of the *Vehicle Routing Problem* (VRP) is to find a set of vehicle routes that connect a pool of customers with a depot. The depot is starting- and endpoint of each vehicle route. Practical applications for these problems are for example delivering newspapers, or the collection of milk products from farms. This section gives a brief overview of this problem, because a lot of papers done for the HHC problem, define their problems as a VRP with additional side constraints. At first, in Subsection 2.2.1, the problem will be introduced and basic graph theoretic knowledge will be explained. The second part of this section, Subsection 2.2.2, will present some basic extensions of the VRP. Additionally, Section 2.4 is devoted to solution methods, which can be used to solve these problems.

2.2.1 Problem de nition

The VRP was introduced by Dantzig and Ramser (1959) and has attracted plenty of interest from researchers for over 50 years. The problem is so interesting for science, because of its significant importance to economy and the problems many distributers are facing in their daily routine. The applications for this problem are delivery and collection of goods as well as transport problems. Additional examples for the VRP are, waste-collection, routing of nurses, maintenance units as well as salespeople.

The problem definition of the VRP can be described on an example for the distribution of goods, like Toth and Vigo (2002a) did in their book. For the delivery of goods each customer has to be visited. The goods are supplied by a fleet of vehicles, whereas each vehicle has a specific capacity. It has to be considered that the vehicles are operated by drivers, which bring specific constraints (e.g. working time) with them. The vehicle routing depends on a road network to perform the movements needed for supplying the customers (Toth and Vigo, 2002a). The solution of a VRP is a set of routes that attend to all needs of the customers. Furthermore, for each route in the solution there has to be one vehicle that starts and ends the route at the depot. The capacity of this vehicle must be higher than, or equal to the demand of the customers served by the supply route (Laporte et al, 2000). A classical aim for the VRP is to minimize the transportation cost of all routes.

To illustrate the definitions for the VRP, Figure 2.1 shows the solution of a problem with 3 vehicle routes complying the demands of 9 customers. The vehicles for this example have a capacity Q of 10. The demands of the customers q_i are written next to the vertices. For simplicity reasons the cost of the edges are not regarded in this figure.

The road network for the VRP can be either symmetric or asymmetric. Consequently, the edges in the graph can be directed or undirected. This depends on the travelling time in both directions or if it is possible to traverse in both directions anyway (Toth and Vigo, 2002a). A symmetric graph, like in our example, is defined with $G = (V, E)$. The arcs for a symmetric graph are undirected and formulated as $E = \{(i, j) : i, j \quad V, i < j\}$. For an asymmetric graph the definitions are $G = (V, A)$ and $A = \{(i, j) : i, j \quad V, i = j\}$.

The vertices $V = \{0, \ldots, n\}$ in the graph are the depot and the locations of the customers, whereas the depot is 0. Additionally, customers $V \ \{0\}$ are associated with a demand q_i and

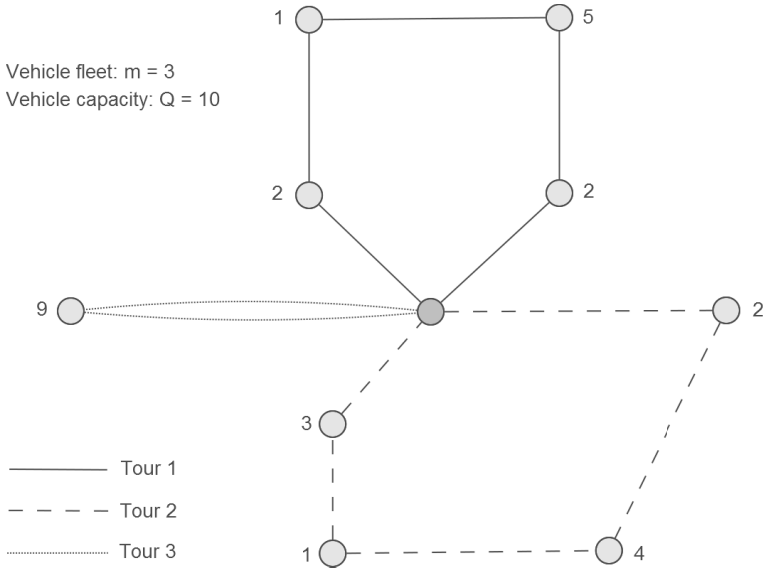

Figure 2.1: Solution of a classical VRP based on Laporte (2007)

the edges of the arcs with a cost of c_{ij} and a cost of c_{ji}, if an asymmetric VRP is given (Cordeau et al, 2007).

Toth and Vigo (2002a) present some further characteristics of the VRP. The vehicle fleet can be homogeneous or heterogeneous. Furthermore, the fleet can differ in the capacity of the vehicles. Moreover, it must be taken into consideration, if the vehicles have to start and end their route at the depot, or if they can end it on a different vertice. The drivers of the vehicles can be seen independent or in association with the vehicles they operate. They can have restrictions in their working time, driving time, or when and how often they have to take a break.

The solution of the VRP can aim for different goals. Goals can be the minimization of the costs associated with the edges between the customers. Further goals can be minimization of the vehicle fleet, or duration of the route considering working times of the drivers (Toth and Vigo, 2002a).

For further details, surveys on the subject of VRP are provided by Toth and Vigo (2002a) and by Cordeau et al (2007).

2.2.2 Basic extensions of the VRP

This section gives an overview over the basic problems studied on the VRP. First, the *capacitated VRP* (CVPR) is explained, because it is the most studied problem and is the simplest one to implement. The graph theoretic model for this problem was already explained in Section 2.2.1 and can be extended for the following problems, because they are related to each other. For further details to the graph theoretical models refer to Toth and Vigo (2002a) or Cordeau et al (2007). Figure 2.2 presents the connection between all the problems

which will be explained, in brief. To illustrate the different problems Figure 2.3 until Figure 2.6 are giving an overview with simple examples.

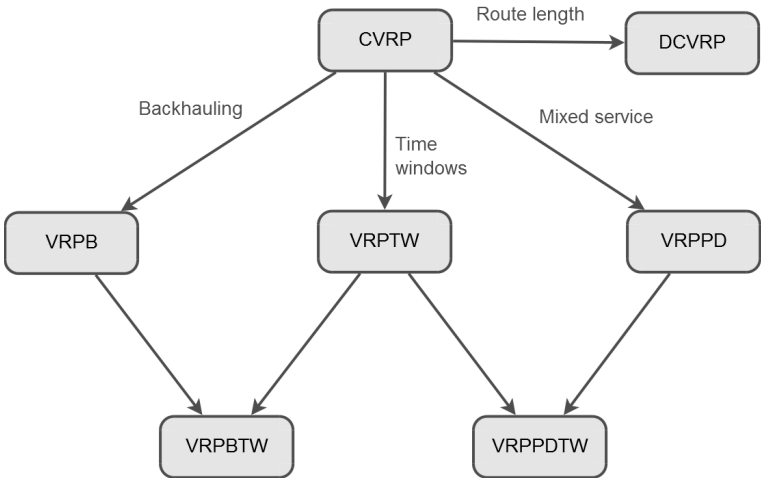

Figure 2.2: The basic problems of the VRP class based on Toth and Vigo (2002a)

The CVPR is the simplest form for the routing problems and has only restrictions for the capacity of the vehicle fleet. The fleet itself is stationed at the depot. All routes in the solution start and end at the depot and the demand of the customers is not allowed to overextend the capacity of the vehicle (Toth and Vigo, 2002a).

In the case of a *Distance-Constrained CVRP* (DCVRP) the problem formulation additionally considers a distance constraint for the maximum route length (Toth and Vigo, 2002a). Furthermore, this constraint can be either the arc-length, or any other cost allocated to the arcs of the graph. An example for the costs and a constraint to the maximum route length can be seen in Figure 2.3. The vehicle route has to fulfill the capacity constraint as well as the time constraint. In the literature the DCVRP is denoted as VRP. To this end, for the next problem definitions can be generalized that they can have distance-constraints, even if it is not denoted in implicitly.

Another extension of the VRP is the *VRP with Time Windows* (VRPTW). Accordingly, these variant can also have distance constraints. In this variant of the VRP the service for each customer has to be done between a specific time window (Cordeau et al, 2002a). The time window is defined in an interval $[a_i, b_i]$. Furthermore, a travel time t_{ij} for each arc connecting the vertices is given. In addition, a service time s_i is defined for each customer. The time window can be defined as a hard, or a soft-constraint. For the first case, the customers have to be visited during the time window and it is not allowed that the vehicle arrives too late. If the vehicle arrives earlier, it has to wait until a_i to start the service (Toth and Vigo, 2002a). In the case of a soft time window, the service can start outside of the time window but has an additional penalty-cost. To illustrate this problem, an example is given in Figure 2.4. From one customer to the next, the *traveltime* and the *servicetime* is added up and results in *arrivaltime* for the next job. The vehicle has to arrive in between the time window at the whole route, or to wait in the case it arrives to early.

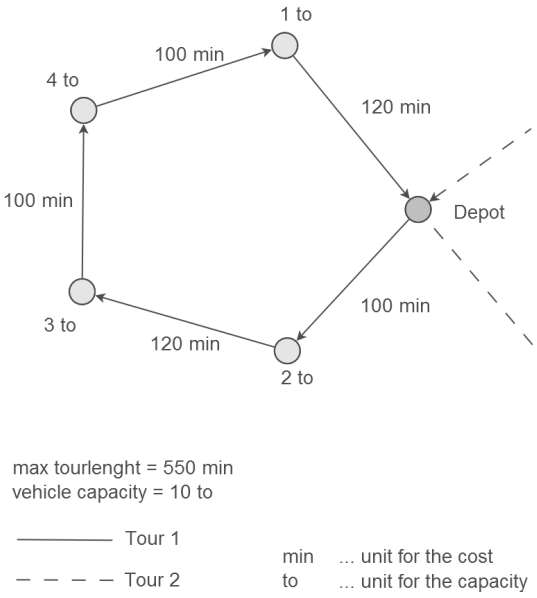

Figure 2.3: DCVRP based on Hirsch and Schweiger (2014)

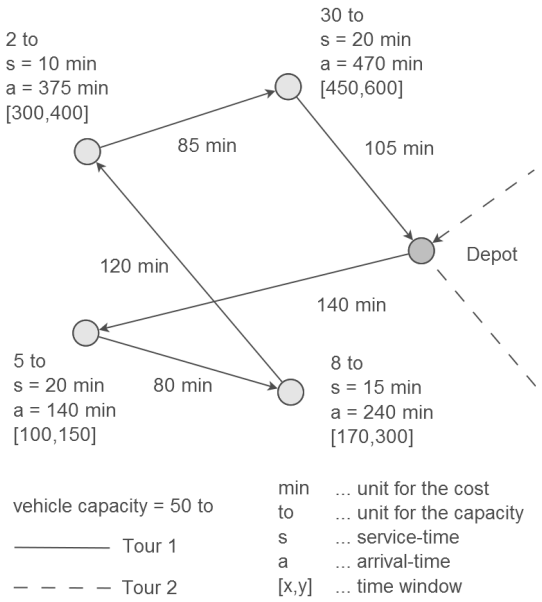

Figure 2.4: VRPTW based on Hirsch and Schweiger (2014)

The *VRP with Backhauls* (VRPB) is another extension of the VRP. Toth and Vigo (2002b) describe that for this problem the customer set is divided in two parts, the linehaul customers and the backhaul customers. Accordingly, the VRPB can have additional distance constraints. The linehaul customers are defined as $L = \{1 \dots n\}$ and have a specific demand to be delivered. On the contrary, the backhaul customers are defined with $B = \{n+1 \dots n+m\}$ and have a specific amount of goods to be picked up. In extension to the VRP the VRPB considers 2 additional constraints. The first constraint is that the capacity of the vehicle must be as high, or higher than the routes total-load for both subsets (L, B). The second one is that the linehaul customers have to be visited first and the backhaul customers, when the vehicle is on the way to the depot again. In the case, that a VRPB is given, which is including time windows, it is called *VRP with Backhauls and Time Windows* (VRPBTW) (Toth and Vigo, 2002a). Figure 2.5 gives an example for the VRPB. As it can be seen in Figure 2.5, the vehicle has to serve the linehaul customers first and can serve the backhaul customers afterwards. For both subsets the maximum capacity of the vehicle has to be met and no distance constraints have been assumed.

The last extension of the VRP discussed in this section is the *VRP with Pickup and Delivery* (VRPPD) (Desaulniers et al, 2002). The following characteristics apply for this problem. Each vertex is associated with a demand d_i and a quantity to be picked up p_i (in some cases only the net difference between these two values is given) (Toth and Vigo, 2002a). Additionally, O_i stands for the vertex which is holding the delivery demand and has to be visited before arriving at the customer i. Consequently, D_i denotes the vertex

which is the location where the goods have to be delivered to after pickup. The customer D_i has to be served after customer i. A special case of the VRPPD is the extension with time windows and in that case it is called *VRP with Pickup and Deliveries and Time Windows* (VRPPDTW) (Toth and Vigo, 2002a). Figure 2.6 shows an example for the VRPPD. The customers have to be visited in the right order, without exceeding the vehicle capacity. For simplicity of the example, no distance constraints have been assumed.

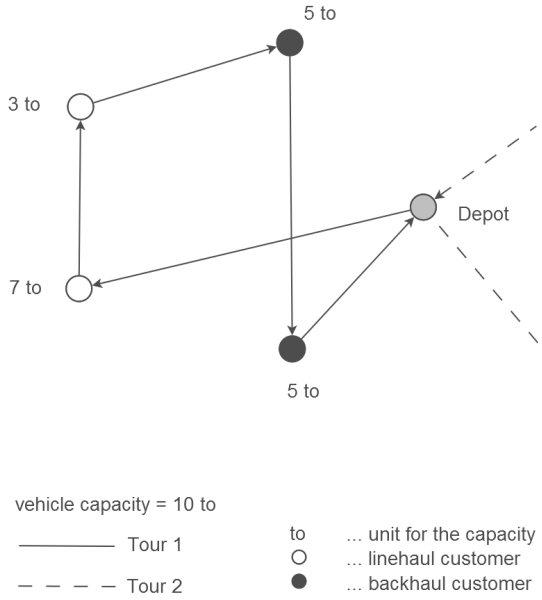

Figure 2.5: VRPB based on Hirsch and Schweiger (2014)

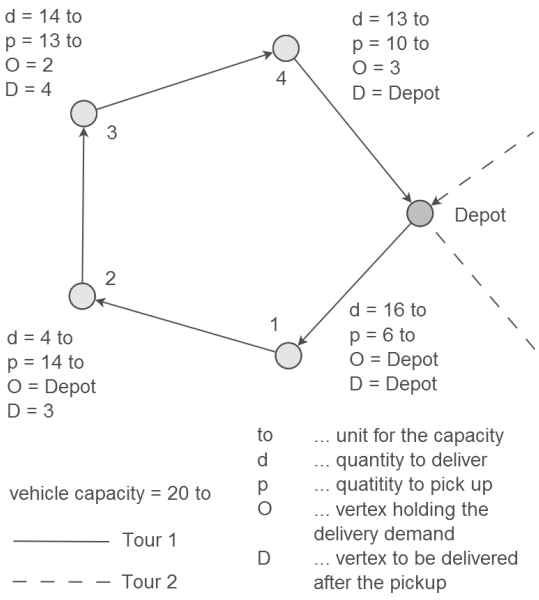

Figure 2.6: VRPPD based on Hirsch and Schweiger (2014)

In summary, the variety of models for VRP are matching parts of our problem definition as well, but we have to take into account that all problems presented in this section concern the transportation of goods (Desaulniers et al, 2002). Therefore, in Section 2.3 a problem formulation will be introduced that is covering the transport of persons, which is called the *Dial-a-Ride Problem*.

2.3 The Dial-a-Ride Problem

The aim of the *Dial-a-Ride Problem* (DARP) is to find a set of vehicle routes for n customers who need a transport from a starting-point to a delivery-destination. The objective for this problem is to minimize the costs of vehicle routes, considering a set of constraints. In comparison to the VRP, this problem considers the user inconvenience as well (Parragh et al, 2008). For this reason, the problem needs to minimize also the ride-time for the users on board, as well as the waiting time for users. Practical applications are for example door-to-door delivery of elderly people or, like presented in this master thesis, the routing

and scheduling in home health care systems. At first, in Subsection 2.3.1, an overview to this topic will be provided and a basic idea of the mathematical problem will be given. The second part of this section, Subsection 2.3.2, will present the main ideas for the single-vehicle and the multi-vehicle DARP.

2.3.1 Problem de nition

The DARP generalizes the VRPPD in some aspects (Healy and Moll, 1995). In addition, the human perspective has to be included for this problems. In some cases the routing of the vehicles would pool the transportation requests of customers, but would leave some customers with long waiting times, or ride times. For that reason, there are more criteria for the problem definition like customer waiting time, customer ride time and delivery time windows. For the most work done to this problem a homogenous vehicle fleet is assumed, but vehicle may differ in size, or in their infrastructure (e.g. transportation of wheelchairs). Moreover, there can be more than one depot for the vehicles (Cordeau and Laporte, 2003).

For some problems it may be good to determine a routing plan and a fleet size which is capable of satisfying all demand. On the other hand, a possible approach to the solution may be to try to maximize the served requests for a fixed vehicle fleet. Hence, Cordeau et al (2007) describe the following two problems for the DARP: minimizing the costs considering all constraints and requests; maximizing the requests executed for a fixed vehicle fleet.

The mathematical formulation for the DARP is presented by Cordeau (2006) in the following way. The variable n defines the number of requests to be served. For the DARP a directed graph $G = (V, A)$ is formulated, whereas the vertex set V is portioned into $\{\{0, 2n+1\}, P, D\}$ where 0 and $2n + 1$ are copies of the depot. $P = \{1, \ldots, n\}$ and $D = \{n + 1, \ldots, 2n\}$ are defined for the pickup and the delivery vertices. For each user i a start node i as well as an end node $n + i$ are forming a request, consequently $i \quad P$ and $n + i \quad D$.

The vertices have further variables which are a load q_i and a service duration d_i. The two depots have a load and a service duration of 0. For the pickup vertices the load is $q_i \quad 0$ and for the delivery vertices the load is $q_i = -q_{i-n}$. The service duration for every node except the depots is $d_i \quad 0$. For each node a time window $[e_i, l_i]$ is given, which presents the earliest and the latest time the transport should arrive. In addition, there is a set of vehicles K with k vehicles. The capacity for each vehicle is Q_k. The arc set is defined as $A = \{(i, j) : i = 0, j \quad P$ or $i, j \quad P \quad D, i =$j and $i = n + j$, or $i \quad D, j = 2n + 1\}$. For the arcs (i, j) connecting the vertices costs c_{ij} and a travel time t_{ij} are associated. At last a maximum ride time for each customer is given which is defined with L.

The detailed formulation for the mixed integer program can be looked up in Cordeau (2006) or Cordeau et al (2007). To explain it in short, Figure 2.7 is giving an overview with a simple example. The objective function for the problem should minimize the total routing cost for all arcs traversed. For this graph $n = 4$, $P = \{1, 2, 3, 4\}$, $D = \{5, 6, 7, 8\}$, consequently the request are 1 to 5, 2 to 6 and so on. The depot is defined with node 0 and node $2n + 1$. The continuous line in the figure shows the vehicle route. The dashed line shows the arcs which are forbidden to go back, because due to the arc set definition described earlier. There are constraints that ensure that each customer is transported only

once and that the pickup happens before the delivery. Furthermore, each vehicle has to start at the depot and has to end the route at the depot again. Further constraints are regulating the maximum ride time, capacity and that time windows are considered.

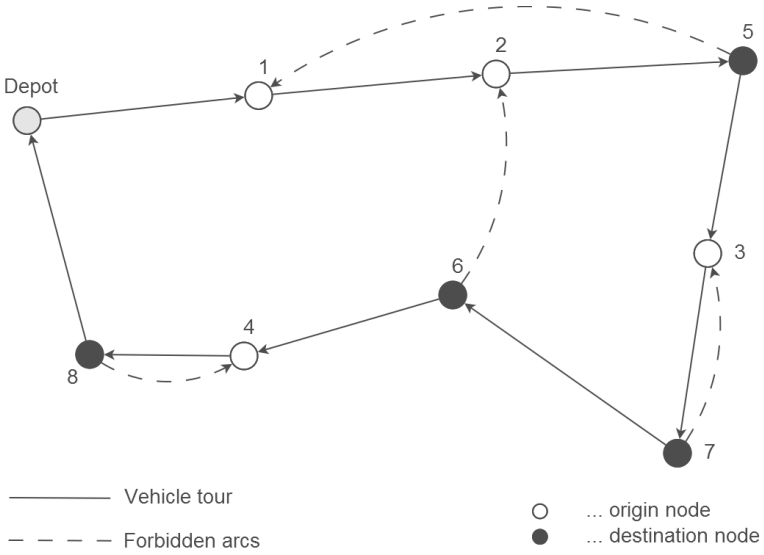

Figure 2.7: Overview of the DARP

2.3.2 Basic notations to the DARP

The DARP can be divided into single-vehicle DARP and multi-vehicle DARP, whereas more research has been done for the latter one. For both versions we can further differ between static and dynamic cases (Cordeau and Laporte, 2007).

For the static case, all requests are known before the scheduling starts. Surveys on the work done for this case can be found in Cordeau and Laporte (2003), Cordeau and Laporte (2007) and Parragh et al (2008). In summary, Parragh et al (2008) concludes that there are exact methods for the static DARP that can solve instances up to 96 transportation-requests. Nevertheless, it is hard to compare the different approaches because there is no standardized data set for the test instances. The tests do not only depend on the problem size, but also on constraints like how narrow the time windows are set. If heuristics or metaheuristics are used to solve the problem, it becomes even harder to compare the different work done in this field. In short, problem types, constraints, and objectives may differ. In summary, it can be said that heuristic methods run fast. Metaheuristic solution methods will perform better in respect to the solution quality (Parragh et al, 2008).

In the case of a dynamic DARP, requests are coming in over the day and the scheduler has to meet the demand in real-time and attach it to already existing routes (Cordeau and Laporte, 2003). For surveys regarding this topic please refer again to Cordeau and Laporte (2003), Cordeau and Laporte (2007) and Parragh et al (2008). In contrast to the static DARPs, there are less solution algorithms presented in the literature. Moreover, there are

almost no exact methods implemented for the dynamic DARP, because for that case reaching optimal results may not be needed (Parragh et al, 2008). Most of the solution methods used are heuristic approaches, accordingly to their short response times. The response time is also the reason that metaheuristic solutions are not optimal. Therefore, Parragh et al (2008) summarize that there have only been a small number of solution approaches and the most of them are repeated calls of solution methods for the static case. In addition, they conclude that there are no standardized data sets as well to compare the different approaches researched for this topic.

The problem covered in this work is a static many-to-many multi-trip Dial-a-Ride Problem. Taken together, there has already been a lot of research done for the DARP, but the differences to the problem described in Fikar and Hirsch (2014) are the constraints for the nurses and that there is an additional option included, which is that the nurses can walk between certain customers.

2.4 Solution Methods

The problems for this paper can be generalized as a Travelling Salesman Problem (TSP) (Cordeau et al, 2007). Moreover, they are NP-hard (Cordeau et al, 2007), which means that the problem can not be solved in reasonable time by an exact approach on a standard computer. The reason for that is the non-polynomial increase of the possible solutions, when the problem size is increased (Juan et al, 2010). The computational time rises exponential with the problem size and so only small instances can be solved with an optimal solution (Cordeau et al, 2002b). Therefore, heuristics and metaheristics are often used to get solutions for this problems. Accordingly, this chapter will introduce some solution methods, and discuss their field of application.

The most famous classes in complexity theory are *P (polynomial)* and *NP (nondeterministic polynomial)*(Whitley and Watson, 2005). The complexity of the most efficient algorithm for a specific problem assigns the complexity to the problem (Landgraf, 1995). Whereas, the problem set for P can be solved in polynomial time on a deterministic Turing Machine, the problems which are NP-hard increase their computational time with increasing problem size drastically. The problem size which can be solved exactly for VRP is for instances of up to 100 customers (Laporte, 2007). The computational time is also depending on side constraints the problem has. Therefore, practical applications for the exact algorithms are limited by the size of the problem instances which makes them inadequate and heuristics are often used instead.

The outline of this chapter is as follows. Section 2.4.1 gives an overview to the exact approaches for routing problems. The second part, Section 2.4.2, presents heuristic methods. The methods will be differed in construction- and improvement heuristics. In addition, a lot of space will be given to explain the Savings Heuristic introduced by Clarke and Wright (1964), because it is the baseplate for the further solution approach discussed in Chapter 3. In Section 2.4.3 metaheuristic solution methods will be introduced, which have become more and more popular over the last decades. Moreover, metaheuristic solution methods improve the solution quality by searching through the most promising regions in the solution space

and not getting stuck at local optima. However, for these methods the computing time is much higher than for classical heuristic methods (Laporte et al, 2000).

2.4.1 Exact Models

Exact approaches result in an optimal solution, or may report that the solution is infeasible (Engeler, 2002). The approaches are based on partial enumeration. Engeler (2002) divides the exact algorithms into: branch-and-bound; and dynamic programming. Branch and bound approaches are calculating the solutions depending on a decision tree, whereas dynamic programming is searching for the shortest path in the graph (Engeler, 2002). For more details to exact algorithms for VRP, Laporte (1992) are providing a survey. The following itemization shows two examples for exact approaches, though there are more exact approaches for VRPs and DARPs:

- *Branch and Bound*

  For problems with increasing size complete enumeration is not possible anymore and other alternatives have to replace this method (Dowsland, 2005). The branch and bound algorithm is working with the behavior that it is partitioning the complete problem into smaller sub-problems and eliminating those that are not leading to a good solution. Engeler (2002) describes the branch and bound algorithm as a decision tree for which the branches are defined through rules (e.g. vehicle type).

  Accordingly, to the reason that this algorithm is trying to eliminate non promising solutions there has to be a method to ensure that. The process for that purpose is called pruning, Dowsland (2005) describes it as follows for a minimization problem. To evaluate the solutions found, they are compared to a lower and an upper bound. Consequently, a solution has to be in between those two bounds to not get eliminated. The upper bound should provide a value where we want to do at least as good as this bound to take the solution into consideration. The upper bound is usually defined through the best solution found so far. As lower bound an estimated value for completing the sub-problem is taken. To use this algorithm for a maximization problem, the rules of upper and lower bounds have to be reversed.

  Other approaches and combinations have been introduced for this approach, which is for example the Branch and Price algorithm. This algorithm is a hybrid between Branch and Bound and column generation methods. Dantzig and Wolfe (1960) used a specific form of problem reformulation to provide a tighter LP relaxation bound. Another variation is the Branch-Price-and-Cut solution approach, which is described in detail in Trautsamwieser and Hirsch (2014).

- *Dynamic Programming*

  Another exact algorithm is *dynamic Programming* (DP), which breaks down the problem into simpler sub-problems (Dowsland, 2005). It divides the problem in different stages and it processes all options available at a stage before going on to the next stage (Dowsland, 2005). The stages in DP are linked over a recursive relationship.

Dowsland (2005) describes that the DP implementation has four main characteristics, which are *stages*, *states*, *decisions* and *policies*. Considering these characteristics: at each stage of the problem, for each feasible state, a decision has to be made how to achieve the next stage. The decisions for every stage are then combined into an overall optimal policy (Dowsland, 2005).

This method can only be applied to small problem instances, because the computational time is rising exponentially. For that reason, this method is used to calculate sub-problems (e.g. TSP), or as a subroutine for the branch and bound (Engeler, 2002).

2.4.2 Heuristics

Since there is no way to compute big NP-hard problems with exact approaches in a reasonable time, heuristics are used to explore regions in the solution space (Cordeau et al, 2002b). Although, the solutions resulting from heuristic search methods may not be optimal, finding a good solution may be a benefit for a lot of applications. The following definition by Rayward-Smith et al (1996) provides a good description for heuristics:

'A heuristic technique (or simply heuristic) is a method which seeks good (i.e. near-optimal) solutions at a reasonable computation cost without being able to guarantee optimality, and possibly not feasibility. Unfortunately, it may not even be possible to state how close to optimality a particular heuristic solution is.

Heuristics can be divided into constructive and improvement heuristics which will be discussed in the following two sections (Laporte, 2007). For a construction heuristic the goal is to find a quick solution which can be optimized afterwards with an improvement heuristic.

Construction Heuristics

There are a lot of construction heuristics and their main purpose is to build a solution from scratch. Laporte et al (2000) propose that there are two main techniques that are used for creating VRP solutions with heuristics. The first technique is to merge already existing routes and to examine how much each possible merge can save for the overall tour-length. The second technique is to gradually assign vertices to routes considering the insertion cost. The three most popular and well known heuristics will be discussed subsequently:

- *The Savings Heuristic*

  The Savings Heuristic was developed by Clarke and Wright (1964) (CWS) and has become one of the most used heuristics, since then (Cordeau et al, 2002b). The CWS can be implemented for both, directed or undirected graphs (Laporte et al, 2000).

  At the start of the implementation of the algorithm, n back and forth routes $(0, i, 0)(i = 1, 2, \ldots, n)$ are created (Laporte, 2007). Figure 2.8 shows the initial routes, which have to be created first. The next step is to calculate the savings value for every possible merge of the initial tours. How they have to be merged is shown in Figure 2.9 and defined with $S_{ij} = c_{i0} + c_{oj} - c_{ij}$. Taken together, the arcs from $(i, 0)$ and $(0, j)$ are

removed and replaced with (i, j). Starting with the highest savings value the algorithm can progress with two version. The Savings Heuristic can be either, sequential, or parallel.

For the parallel version (best feasible merge) applies the following. The algorithm starts with the highest savings value s_{ij} and it has to be determined if a merge between this two routes is feasible or not. If the merge is feasible, arcs $(i, 0)$ and $(0, j)$ are deleted and arc (i, j) is added. Then the next highest savings value are taken and are tested on feasibility. The algorithm stops, if no positive savings value is left.

For the sequential version (route extension) every route $(0, i, \ldots, j, 0)$ is taken into consideration (Laporte et al, 2000). The first saving s_{ki} or s_{jl} that can feasible merge the current route with another route is determined, however, the second route has to start at $(0, l)$ or end at $(k, 0)$. That step is continued for as long as no merge for the current route is found. The next step, is to apply the same operation for the next possible route, setting this one the current route. When no route merge is feasible the algorithm stops.

Laporte et al (2000) and Cordeau et al (2002b) prefer the parallel version, because the expected results are better. Moreover, Laporte and Semet (2002) provide a comparison for both versions on the instances Christofides et al (1979) provided to test CVRP and DCVRP solution methods. In summary, this comparison shows that the parallel version dominates the sequential one in each test-instance.

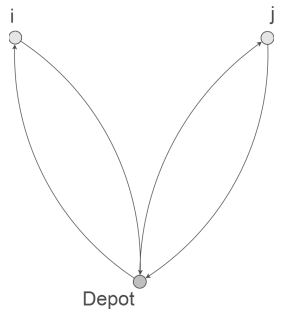

Figure 2.8: Creating initial tours

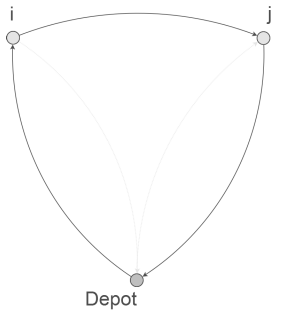

Figure 2.9: Merging two initial tours

Positive aspects for the CWS are simplicity and speed (Cordeau et al, 2002b). However, negative points for this algorithm are that it is not that accurate and inflexible. A problem is that it works on a greedy principle and has no means of undoing bad choices once they are done (Cordeau et al, 2002b).

Several ideas to improve the CWS have been proposed. The main problem is that the heuristic produces good routes at the start, but less rewarding ones towards the end (Laporte and Semet, 2002). Gaskell (1967) and Yellow (1970) advanced the algorithm by setting a parameter λ for $S_{ij} = c_{i0} + c_{0j} - \lambda c_{ij}$ to get more compact routes (Cordeau et al, 2002b). Hence, the higher λ, the more value gets the new arc connecting i and j. For further variants refer to Laporte and Semet (2002).

Another extension for the CWS is the biased randomization introduced by Juan et al (2013). The problem for the CWS is that the choices of the first merges have a strong impact on the results. The main idea is to give the algorithm a random behavior and produce multiple solutions (Juan et al, 2013). To implement this behavior, Juan et al (2013) introduce a parameter that is regulating the steepness of a geometric distribution. Basing on the distribution each savings value is allocated to a probability to get chosen. The steepness of the distribution is adjusting the probability. In the case = 0 each savings value has the same chance of being chosen. On the other hand if = 1, the savings list is processed from top to bottom as in the standard CWS. By running the algorithm multiple times with different random-numbers, for the selection of the savings values, different solutions can be created and the solution can be improved by easy adaptions of the parameter .

- *The Sweep Algorithm*

  The Sweep Algorithm was developed by Gillett and Miller (1974) and is one of the most elementary types of petal heuristics. The algorithm is best described by seeing how it works. Therefore, Figure 2.10 shows the main idea. The vehicle capacity Q for this example is 10. Starting at the depot a half-line is constructed (Laporte, 2007). Another half-line is rotating around starting at the first one. With increasing angle customers are included in the current route. The route is finished when constraints are reached and no further customer can be added to the route.

  Compared to the CWS, this algorithm is simpler. But concerning other factors, like accuracy and speed, the CWS has shown better results in the comparison provided by Cordeau et al (2002b). Furthermore, the method has also a greedy nature, which makes it difficult to add extra constraints. To sum up, the sweep algorithm is not flexible (Cordeau et al, 2002b).

- *The Fisher and Jaikumar algorithm*

  The third algorithm introduced in this work is the Fisher and Jaikumar (1981) algorithm, which is a two-phase process (Cordeau et al, 2002b). The first step is to create a feasible cluster of customers, which is solved by a *generalized assignment problem* (GAP). In more detail, a seed is located in areas where routes are likely to lie and according to that the GAP is solved. The GAP itself has the objective to minimize the distance between customers and the seed (Laporte, 2007). In addition, constraints like vehicle capacity have to be considered as well. In the second step, a vehicle route is scheduled. The solution in this step is reached by solving a TSP (Cordeau et al, 2002b). Figure 2.11 shows an example by Fisher and Jaikumar (1981) how seeds may be set and the construction of the routes may look at the start of the algorithm. The dashed lines form cones for the customers and the solid lines respectively for the vehicles. Moreover, constants for this figure are a vehicle fleet with a size of $K = 3$ and a capacity $b_k = 30$. The demand for the customers a_i is standing beside the vertices.

  In comparison to the sweep algorithm, the routes for this method can intersect with each other again. Hence, the evaluation by Cordeau et al (2002b) for this algorithm

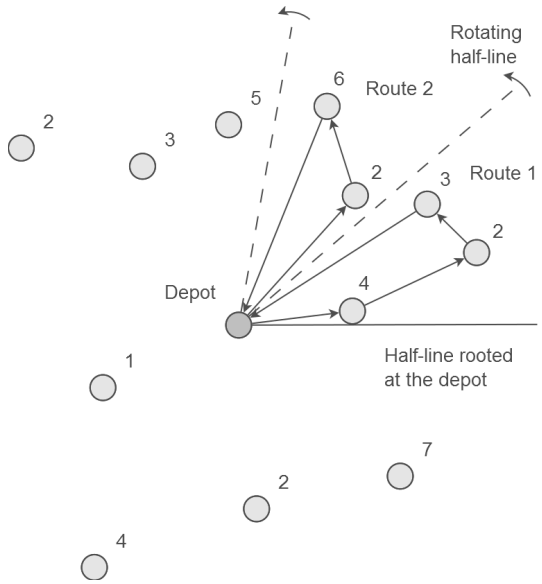

Figure 2.10: Sweep algorithm based on Laporte (2007)

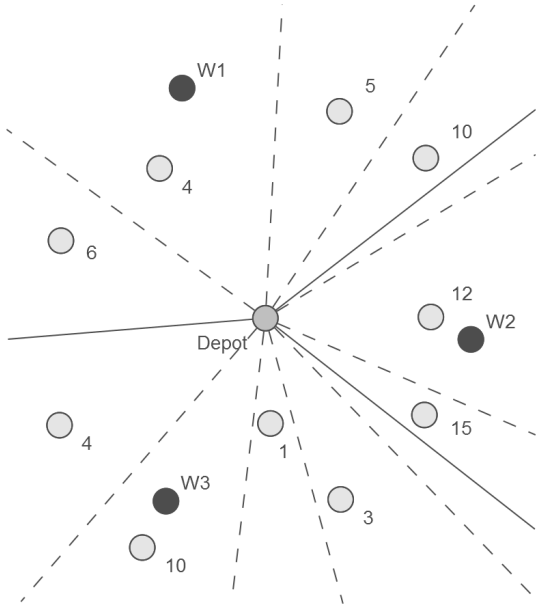

Figure 2.11: Algorithm by Fisher and Jaikumar (1981)

is that it is not easy to implement and depending on how the seeds have been chosen and the Lagrangian process has been implemented. Moreover, another difficulty is the solution of the GAP, because it is NP-hard (Laporte, 2007). It is also not easy to add additional constraints.

Improvement Heuristics

To improve the solutions of the construction heuristics, improvement heuristics can be implemented. They can only start based on an already existing solution and are consequently changing it (Hirsch and Schweiger, 2014). Improvement heuristics can work in two different ways for the VRP. Therefore, Laporte (2007) differs improvement heuristics in *intra-route heuristics* and *inter-route heuristics*.

For the first one, the routes themselves are optimized by changing the order in which the vehicle serves the customers. Example for this are TSP improvement heuristics like *2-opt* or *3-opt*. For this algorithm edges connecting the routes are removed and afterwards the route is connected again.

In contrast, inter-route exchanges mean that customers are moved to different vehicle routes. For this problems and different exchange habits good surveys are provided by Thompson and Psaraftis (1993), Van Breedam (1994) and Kinderwater and Savelsberg (1997). Whereas, Thompson and Psaraftis (1993) present a method which is called *b-cyclic, k-transfer*. In short, exchange between b routes is done, transferring k customers. Van Breedam (1994) discusses and classifies exchange cases between two different routes

(e.g. *string cross* – for this operation two routes are exchanged by crossing two vertices).

However, one negative point for improvement heuristics has to be mentioned. They are searching in their neighborhood and may get stuck at a local optimum (Hirsch and Schweiger, 2014). In conclusion, it is possible that the global optimum can not be reached, resulting in a poor solution. For a better understanding, Burke and Kendall (2005) are presenting an example on the local search algorithm *hill climbing*. For Figure 2.12, we assume to maximize a benefit in a solution space. The main idea is to look for good solutions in the neighborhood of the current solution. However, moving to a new solution is only possible if it is better than the current solution. As can be seen in the figure, the algorithm can not find the global solution if starting at the position assigned in this example. The reason for that is, that it is not possible to leave the local optimum and the algorithm terminates.

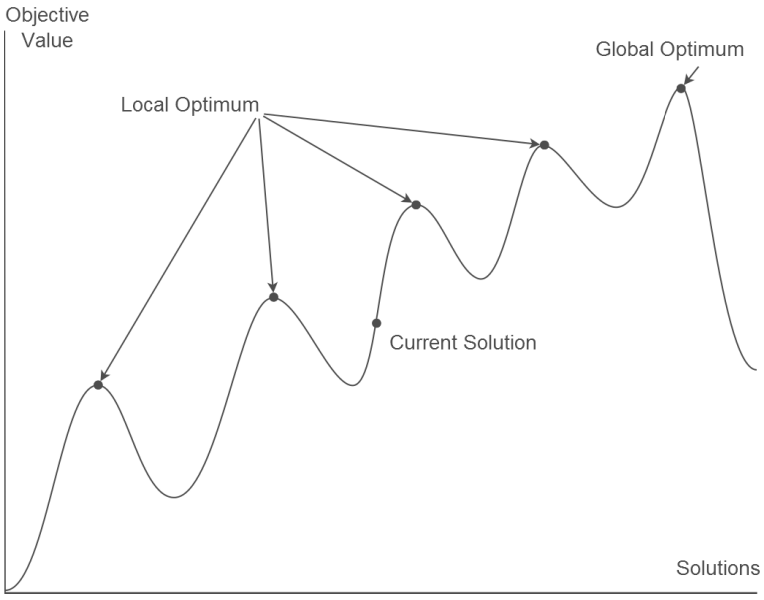

Figure 2.12: Hill climbing getting stuck in local optimum based on Burke and Kendall (2005)

2.4.3 Metaheuristics

Over the last two decades, a lot of effort has been put into developing methaheuristics (Cordeau et al, 2002b). In comparison to the classical heuristics described in Section 2.4.2, metaheuristics allow solutions to get worse, or even infeasible during the search process (Gendreau et al, 2002). Therefore, metaheuristics deliver better results and do not get stuck so easy in local optima, but they sacrifice speed for that benefit. The following definition by Glover and Laguna (1997) provides a good description to the term metaheuristic:

'A meta-heuristic refers to a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality. The heuristic guided by such a meta-strategy may be high level procedures or may embody

nothing more than a description of available moves for transforming one solution into another, together with an associated evaluation rule.

Metaheuristics can be divided in *local search*, *population search* and *learning mechanism* (Laporte, 2007). There is no consensus to the topic of categorizing metaheuristics accordingly. In this master thesis the different metaheuristics will be divided in local search and population search and for both types two examples will be given to give a brief overview, over this large topic. For further information refer to Burke and Kendall (2005).

Local Search

Starting at an initial solution s_0 the local search metaheuristics are moving from one solution s_t to the next solution in the neighborhood $N(s_t)$ (Laporte, 2007). The neighborhood is defined by all solutions that can be reached from the current one, over the algorithmic changes to the routing. The search terminates, with the best known solution s^* after reaching a stopping criterion (Laporte, 2007). The stopping criterion can be a pre-defined number of iterations, or a number of iterations after s^* does not improve anymore. Thus, there are a lot of algorithms counting to that framework, but only two popular ones will be described, namely the *Tabu Search* and the *Variable Neighborhood Search* (VNS):

- *Tabu Search*

  This algorithm stands out as one of the best choices to solve VRP (Cordeau et al, 2002b). Moreover, this method has already obtained optimal and near optimal solutions for various practical applications (Glover, 1990). For tabu search a process is needed, in which a set of moves is changing a solution. Moreover, the attractiveness of the moves has to be evaluated.

  Glover (1990) describes that Tabu Search is incorporating three thoughts. First, memory structures are needed which can hold different characteristics (e.g. evaluation criteria). Second, methods have to be implemented to control the search process (e.g. tabu restrictions). Last, different time spans for the memory functions have to be considered. Hence, the Tabu Search algorithm is working with short term and long term memory functions. Furthermore, mechanisms, as diversification, are aiming for solutions which are not used frequently. On the other hand, mechanisms like intensification, are looking out for good solutions and intensifying the solution space of them (Laporte, 2007).

  The algorithm starts with an initial solution and is searching in the neighborhood $N(s_t)$ for the best solution s_{t+1} (Laporte, 2007). To prevent the search from cycling, a set with for forbidden solutions $T(s_t)$ at iterations t is provided. To create a solution a move is done and consequently declared tabu for iterations. The algorithm terminates after a stopping criterion (e.g. number of iterations) (Hirsch and Schweiger, 2014).

- *Variable Neighborhood Search*

  The VNS is a metaheuristic using local search procedures for solving combinatorial and global optimization problems (Hansen and Mladenovic, 2003). Therefore, it is system-

atically changing the neighborhood during the local search for solutions (Hansen and Mladenovic, 2003). The algorithm starts with a predefined set of neighborhood structures $N_k, (k = 1, \ldots, k_{max})$. Consequently, the search is starting at N_1 and switching to N_2 for the second neighborhood during the process. Looking for local optima in the neighborhood, the algorithm proceeds to the next neighborhood, if it finds no improvements (Hiermann et al, 2013). The search terminates at the point where no improvements can be found considering all predefined neighborhoods. Different search strategies for the solution are described by Hiermann et al (2013):

- *Random:* the next neighbor is selected randomly
- *Next-Improvement:* if a better neighbor than the current best solution is found, it is selected
- *Best-Improvement:* the best neighbor of all possible neighbors is selected
- *Best-Of-Improvement:* this is best improvement for a subset of neighbors

Population Search

Two examples for population search metaheuristics are the *Genetic Algorithm* and the *Ant Colony Optimization*.

- *Genetic Algorithm*

  Introduced by Holland (1975), the genetic algorithm is inspired by the evolution in nature and applies that to optimization problems (Hirsch and Schweiger, 2014). The algorithm evolves a population of solutions (chromosomes) over different means. The mechanisms to evolve the chromosomes are mimicking natural phenomena like mutation, or inversion (Gendreau et al, 2002). Further mechanisms are explained in Hirsch and Schweiger (2014) and shown in Figure 2.13. Gendreau et al (2002) describes the algorithm presenting a simple example:

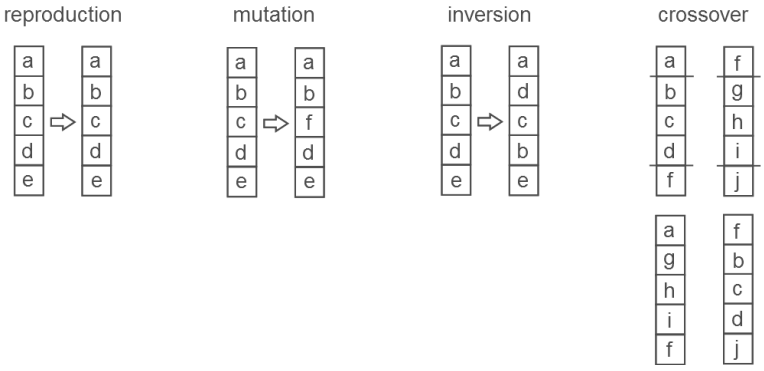

Figure 2.13: Inheritance mechanisms for the GA based on Hirsch and Schweiger (2014)

At the start of the algorithm, an initial population is randomly created. The population is defined as $X^1 = \{x^1{}_1, \ldots, x^1{}_N\}$. For iterations $t = 1, \ldots, T$ the first three steps

described in the following itemization after Gendreau et al (2002) have to be applied k times considering ($k < N$ 2). At the end, Step 4 has to be processed. The parameters defined earlier are T, for the number of generations, and k, for the number of selections per generation (Gendreau et al, 2002).

- *Step 1 reproduction:* Two parental chromosomes are selected probabilistically biased in considering the best chromosomes
- *Step 2 recombination:* Two offsprings are generated using one of the crossover operators described earlier in Figure 2.13
- *Step 3 mutation:* At the mutation step a random mutation can be applied with a small probability
- *Step 4 - generation replacement:* Creating a new population by removing the $2k$ worst solutions and replacing them with the newly generated solutions

- *Ant Colony Optimization*

  This algorithm is based on the natural behavior of ants (Gendreau et al, 2002). Hirsch and Schweiger (2014) describe this method as one of the most intelligent versions of swarm-systems. Accordingly, the *ant colony optimization* (ACO) is depending on the collaboration of all ants as a colony. Moreover, it mimics that ants are almost blind and depending on pheromones to coordinate themselves and build ant trails (Hirsch and Schweiger, 2014). The more interesting paths, which are shorter or leading to highly frequented places, are marked with a larger amount of pheromones (Gendreau et al, 2002). This behavior leads to an efficient road network for the ants. Therefore, the idea has been taken up to create a metaheuristic corresponding to good solution paths (Gendreau et al, 2002).

3 Method

The aim of this thesis is to implement a multi-solution approach for two different problem definitions. The following methods are considered. To create an initial solution, the savings algorithm by Clarke and Wright (1964) is implemented. The reasons why this algorithm has been chosen as well as the algorithm-description, are described in Section 3.1. Using the CWS as foundation of a biased randomized behavior is implemented to create a pool of different solutions where the best one can be chosen as the result. Consequently, the biased randomized behavior is explained in Section 3.2. Moreover, one objective of this work is to provide learning mechanisms for the construction heuristic as well as for the selection of *walking routes* (WR) in the work of Fikar and Hirsch (2014). Therefore, in Section 3.3 different versions of the CWS are described. In addition, Section 3.4 is presenting the method for the selection of WR.

3.1 Savings Algorithm

The solution approach presented in this master thesis is based on the Savings Heuristic proposed by Clarke and Wright (1964). The reasons for that are that it is the most popular heuristic in practice and also easy to implement (Laporte and Semet, 2002). Moreover, the algorithm is adaptable with minimal efforts to various constraints and different problem definitions, which is needed for this thesis. The most important positive aspect of this algorithm is that it has a very fast computational time. This is beneficial, because the algorithm introduced with this thesis generates many solutions. Furthermore, it has to be taken into consideration that this algorithm has to deal with two problems. On the one hand, the approach is implemented for a VRP and used to solve benchmark-instances. On the other hand, a real life application of a many-to-many DARP has to be solved. Another reason for choosing the CWS was that the algorithm of Fikar and Hirsch (2014) is already using a biased randomized CWS. As suggested by Juan et al (2010), also for this work the parallel version of CWS is implemented, because better results are expected.

On the contrary, the CWS is not very accurate and flexible. In the case that good solutions are needed, or that a large amount of money is at stake it might be good to consider a search approach that is going through the solution space more far-reaching (Cordeau et al, 2002b).

The CWS is implemented as shown in Algorithm 1. The input received is a set of vertices containing coordinates, demands and in some instances also service times. Moreover, constraints as the vehicle number, vehicle capacity and maximum ride-time have to be considered. Furthermore, the distance-, or cost-matrix is read in by the algorithm.

Starting the algorithm, the savings-value is calculated for all vertices, $i, j = 1, \ldots, n$ and $i = j$. The formula used is $s_{ij} = c_{i0} + c_{0j} - c_{ij}$. The next step is to create n vehicle tours $(0, i, 0)$ for $i = 1, \ldots, n$. The savings list has to be ordered in a decreasing fashion. Next, the CWS algorithm is applied and a solution is obtained.

Algorithm 1 The savings algorithm

```
procedure CWS(aVertexSet, aConstraintSet, distanceMatrix)
    savingsList = getSavingsList(distanceMatrix, aVertexSet);
    vectorOfTours = getInitialTours(aVertexSet);
    savingsSol = getSavingsSol(vectorOfTours, savingsList, aVertexSet,
    aConstraintSet, distanceMatrix);
    return savingsSol;
end procedure
```

The solution for the parallel CWS algorithm is calculated in the following fashion and shown in Algorithm 2. Starting with the highest value in the savings-list the list is processed until it is empty. The savings-value defines the vertices where the routes have to be merged. Consequently, *verticeI* has to be the end of *routeI* and *verticeJ* the beginning of *routeJ*. For a symmetric cost matrix it would not matter if the vertices are at the end or the beginning of the routes and therefore solutions would be improved easily by adding the possibility to reverse the routes. Additionally, a symmetric cost matrix would halve the size of possible merges. At the end of the function the constraints for the routes are tested for feasibility.

Algorithm 2 Parallel version

```
procedure getSavingsSol(vectorOfTours, savingsList, aVertexSet, aConstraintSet,
distanceMatrix )
    while !savingsList.empty() do
        curValue = select first value in savingsList;
        verticeI = curValue.A;
        verticeJ = curValue.B;
        routeI = getRoute(vectorOfTours, verticeI);
        routeJ = getRoute(vectorOfTours, verticeJ);
        if both routes can be merged and all constraints are satisfied then
            mergeRouteIJ(routeI, routeJ, vectorOfTours);
        end if
        remove curValue;
    end while
    return savingsSol;
end procedure
```

3.2 Biased Randomisation

To improve the performance of the CWS, *biased-randomization* (BR) introduced by Juan et al (2013) is implemented. To change the pattern of selecting the savings-values and to create different solutions, a geometric distribution is advancing the iterator for the savings-list. The steepness of the distribution can be adjusted by parameter , whereas if = 0 every position has the same chance of getting selected and if = 1 the highest rated position is chosen.

Algorithm 3 presents the modified savings algorithm adapted with the BR. In comparison to Algorithm 1 a set of parameters, *algParameter*, is passed to the function. This set of parameters can include the maximum iterations of the for-loops, or the number of solutions, which should be saved. Moreover, different seeds are provided by the parameter-set, because the random number generator that is used for the algorithm should be tested on different random numbers. Furthermore, the parameter-setting for is defined in the parameter set as well, or tested with a for-loop for different values.

Algorithm 3 Biased randomized CWS

```
procedure BR CWS(aVertexSet, aConstraintSet, distanceMatrix, algParameter)
    savingsList = getSavingsList(distanceMatrix, aVertexSet);
    vectorOfTours = getInitialTours(aVertexSet);
    for i = 0; i < maxRuns; i++ do
        rng = generation of a random number;
        distribution.param(beta);
        savingsSol = getBRSavingsSol(vectorOfTours, savingsList, aVertexSet,
        aConstraintSet, distanceMatrix, rng);
        if savingsSol < bestSol then
            bestSol = savingsSol;
        end if
    end for
    return bestSol;
end procedure
```

Accordingly, Algorithm 2 which calculates the savings solution is adapted as well. In the former code the savings value at the first place in the list was taken every time. In the adapted function that can be seen in Algorithm 4, the chosen savings value is depending on the geometric distribution and the numbers provided from the random number generator.

Algorithm 4 Biased randomized parallel version

```
procedure getBRSavingsSol(vectorOfTours, savingsList, aVertexSet, aConstraintSet,
distanceMatrix, rng)
    while !savingsList.empty() do
        curValue = select first value in savingsList;
        randomx = distribution(generator);
        advance curValue for randomx;
        verticeI = curValue.A;
        verticeJ = curValue.B;
        routeI = getRoute(vectorOfTours, verticeI);
        routeJ = getRoute(vectorOfTours, verticeJ);
        if both routes can be merged and all constraints are satisfied then
            mergeRouteIJ(routeI, routeJ, vectorOfTours);
        end if
        remove curValue;
    end while
    return savingsSol;
end procedure
```

The algorithm iterates over the defined maximum iterations and prints the best results for the corresponding test-instances at the end.

3.3 Savings Algorithm with learning-capabilities

This section introduces different versions of the algorithm provided by this thesis. Overall, the versions have a similar approach, but differ in how the iterations and loops are implemented. Generally, they can be divided into two groups, which are explained in the following two flow charts in Figure 3.1 and Figure 3.2.

Concerning the algorithm in Figure 3.1, the idea is to generate a specified number of solutions. The number of iterations is defined in a for-loop. The data for every obtained solution is saved to its corresponding arcs (i, j). The experience gained from the previous leads to a new ranking of the arcs (i, j). With these modifications the biased savings-algorithm is repeated, for a specified number of iterations.

The other version is using a nested for-loop to implement memory capabilities. This can be seen in Figure 3.2. It is similar to the first version, but it is more adaptive and adjusts the algorithm more often to the previous solutions. The outer loop of this algorithm determines how often the learning mechanism should be processed. Accordingly, the inner loop is regulating how much iterations each learning mechanism should process before terminating.

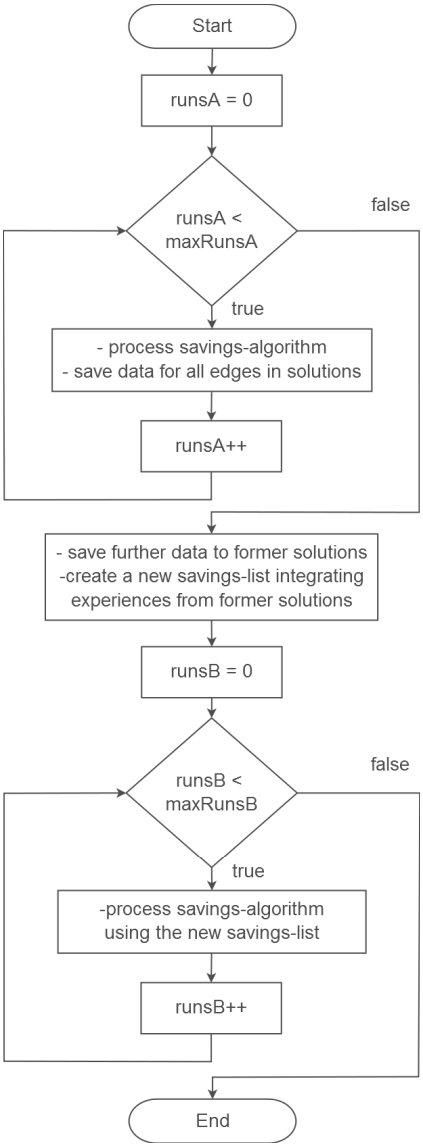

Figure 3.1: Consecutive for-loops

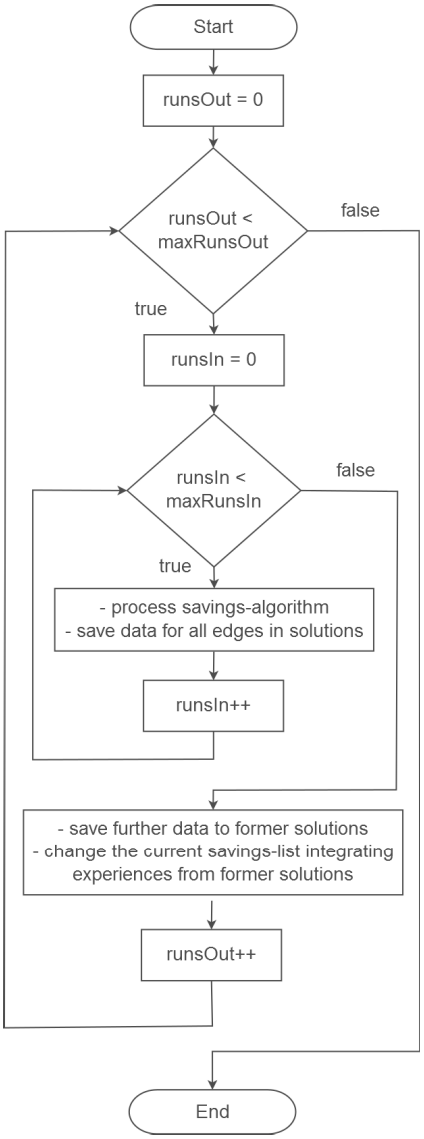

Figure 3.2: Nested for-loops

However, this being the main difference, the alternative versions are working with the same functions that will be explained in brief. To save the data to all edges, following data is stored: (i) tour-length for each tour in which the edge is part of the solution, (ii) the arithmetic mean of all solutions provided by the algorithm, (iii) and a counter of how often each edge is in the solutions. A part of the data is saved right after the savings-algorithm was processed. The other part is saved after a set of iterations is finished and the for-loop (depending on the version: for-loop A; or the inner for-loop for the nested version) ends. The data processed last are the arithmetic mean over the length of all tours and the best known length of each edge. Using this data, there are three ways to precede:

- The first approach is to recreate the savings-list, which is shown in Algorithm 5. Working step by step through the current savings-list each edge (i, j) is tested, if it is in any solution or not. If the edge is in a solution, the current savings-value gets multiplied with the quotient of the arithmetic mean for all solutions and the arithmetic mean of the corresponding edge. In the other case, the current savings-value is reused for the new savings-list.

Algorithm 5 New Savings-list

```
procedure getNewList(savingsList, newList, edgeDataMatrix, arithmeticMean)
    for it=savingsList.begin(); it < savingsList.end(); it++ do
        currentSavings-value = it -> first;
        i = it->second.vertexA;
        j = it->second.vertexB;
        if edgeDataMatrix[i][j].counterInSolution > 0 then
            diffPercentage = arithemticMean / edgeDataMatrix[i][j].arithmeticMean;
            newSavingsValue = currentSavingsValue * diffPercentage;
            newList.insert(newSavingsValue);
        else
            newList.insert(currentSavingsValue);
        end if
    end for
end procedure
```

After generating the new savings-list, the biased randomized savings-algorithm described in Algorithm 4 can start.

- In the second approach the savings-list is adapted similarly to the first approach. However, it differs in calculating the factor that is multiplied with the current savings-value to get the new savings-value. In this version the factor is calculated by dividing the arithmetic mean for all solutions and the best known tour-length of the corresponding edge.

- The third variant to manipulate the selection of the savings-values is not by manipulating the savings list, but by manipulating the iterator which is selecting the next value. Therefore, an additional option to use the already obtained data is introduced. After advancing the iterator to the savings-value that is chosen the position of the iterator is reevaluated and compared to the neighbors in the savings-list using the arithmetic means of the corresponding edges. An example for this function can be seen in Algorithm 6.

Algorithm 6 Best Neighbor

```
procedure getBestNeighbor(savingsList, edgeDataMatrix, currentIterator)
    bestEdge = currentIterator;
    nextEdge = currentIterator;
    compareToNeighbours;
    for i = 0; i < compareToNeighbours; i++ do
        nextEdge++;
        if edgeDataMatrix[bestEdge][].val > edgeDataMatrix[nextEdge][].val then
            bestEdge = nextEdge;
        end if
    end forreturn nextEdge
end procedure
```

In summary, the described variants are shown in Table 3.1. Furthermore, this table defines the abbreviations for the different versions that will be used in the following chapter, Chapter 4.

Table 3.1: Overview of the different solution-approaches

Consecutive for-loop Using two consecutive for-loops, whereas both loops have a pre-defined runtime		
C-AM	C-BL	C-BN
New factor to adapt the savings-values. The factor is calculated by dividing the arithmetic mean for all solutions with the arithmetic mean of the corresponding edge	New factor to adapt the savings-values. The factor is calculated by dividing the arithmetic mean for all solutions with the best length of the corresponding edge	Comparing the current savings-value to the neighbors in the savings-list. The criteria for the evaluation is the arithmetic mean of the corresponding edge
Nested for-loop Using two nested for-loops and testing different adjustments for both loops		
N-AM	N-BL	N-BN
New factor to adapt the savings-values. The factor is calculated by dividing the arithmetic mean for all solutions with the arithmetic mean of the corresponding edge	New factor to adapt the savings-values. The factor is calculated by dividing the arithmetic mean for all solutions with the best length of the corresponding edge	Comparing the current savings-value to the neighbors in the savings-list. The criteria for the evaluation is the arithmetic mean of the corresponding edge

3.4 Selection of Walking-Routes with learning-capabilities

One of the objectives for this master thesis is to obtain solutions for the DARP introduced in the work of Fikar and Hirsch (2014) without using set-partioning.

At the start of the algorithm a set of walking-routes *WR* is created. Based on *WR*, an initial set of walking-routes *WR* is constructed, which includes each job only once. A set-partitioning model selects *WR* by minimizing an objective-function, which is considering walking-route durations and driving distances from the depot to delivery and pickup vertices. Each job is allowed to be selected only once.

The second approach introduced by Fikar and Hirsch (2014), is to select the WR on a random behavior. For this variant each WR has the same chance of being selected.

The aim of this thesis is to select the walking routes randomly based on a list with a geometric distribution and to relinquish from the use of a solver-software. The idea is to learn from previous results, which walking-routes may be beneficial. Therefore, the same method is applied as for the savings algorithm. Each selected walking-route gets the corresponding information to the solution-details saved and after a defined run-time, the set of walking-routes is updated to the previous results.

To describe the algorithm in more detail, the parameters for the algorithm are similar to the parameter used for the Savings Algorithm, except is replaced with to adjust the steepness of the geometric distribution.

At the start of the selection each WR has the same probability of being taken, therefore each *WR* starts with an initial value of 0. The first iterations of the algorithm are the learning phase to gather informations. For each run of the algorithm, the objective value is saved to the corresponding *WR*. After a specified runtime, which is depending on the settings of the for-loops, the arithmetic-mean of the objective value for each *WR* is calculated. If a *WR* has not been in any solution it is assigned to the arithmetic-mean of the objective value for all previous iterations. Furthermore, the set of walking-routes *WR* is sorted considering the assigned arithmetic-mean of the walking-route. As a result, *WR* with better objective values are favored for future runs, because they are ranked higher in the list.

4 Computational experiments

This chapter presents the results obtained by the test runs. The different versions of the modified Savings Heuristic are implemented for two different problems.

At first, the algorithms are tested on a VRP. For these problems the Christofides et al (1979) benchmark instances have been chosen, which are often used to test construction heuristics. The results and computational experiences of these tests are presented in Section 4.1.

The second tests are performed on a many-to-many DARP. These tests solve the same instances that are used by Fikar and Hirsch (2014). Accordingly, the results shown in Section 4.2 are compared to the best results in the paper of Fikar and Hirsch (2014).

4.1 VRP-Tests

This section describes the results for the VRP. At the beginning of this chapter, in Subsection 4.1.1, the Christofides et al (1979) benchmark instances are described. Subsection 4.1.2 presents the parameter used for the algorithm. The last subsection, Subsection 4.1.3 shows and discusses the obtained results. The solution approach is coded in *C++* with Microsoft Visual Studio 2012. Computational results were gathered on an Intel Core i7-4770K, 8 GB RAM, with MS-Windows 8 as operating system. The computation time for the algorithm is a few seconds, depending on the instance size. The instances used for the tests are available at http://neo.lcc.uma.es/vrp/vrp-instances/capacitated-vrp-instances/.

4.1.1 Instances

To test the different approaches on VRP-instances, the Christofides et al (1979) benchmark instances have been used. The test set contains different CVRPs and DCVRPs. The problem size is reaching from 50 up to 199 customers, whereas for the latter no exact method delivers results in reasonable time. Therefore, the solutions of the algorithm are compared to the best known values received with metaheuristics (Laporte et al, 2000).

An overview of the test-instances can be seen in Table 4.1. The test-instances are dedicated to a shortcut, starting with *CB 1* and counting subsequently to *CB 14*. Table 4.1 presents the number of vertices n, not including the depot, the capacity of the vehicles as well as the restriction for tour-length and service-times are described. To compare the results, the best known solutions are added to the information as well. The distance-matrix for the instances is symmetric, but considering the implementation for the DARP it is implemented to work on an asymmetric distance matrix as well.

Table 4.1: Christofides Benchmark Instances. Best known results by Groër et al (2011)

abbr.	n	type	capacity	length	service time	BKS
CB 1	50	CVRP	160	-	-	524.61
CB 2	75	CVRP	140	-	-	835.26
CB 3	100	CVRP	200	-	-	826.14
CB 4	150	CVRP	200	-	-	1,028.42
CB 5	199	CVRP	200	-	-	1,291.29
CB 6	50	DCVRP	160	200.0	10.0	555.43
CB 7	75	DCVRP	140	160.0	10.0	909.68
CB 8	100	DCVRP	200	230.0	10.0	865.94
CB 9	150	DCVRP	200	200.0	10.0	1,162.55
CB 10	199	DCVRP	200	200.0	10.0	1,395.85
CB 11	120	CVRP	200	-	-	1,042.11
CB 12	100	CVRP	200	-	-	819.56
CB 13	120	DCVRP	200	720.0	50.0	1,541.14
CB 14	100	DCVRP	200	1040.0	90.0	866.37

4.1.2 Parameter setting

This subsection discusses the parameters used for the algorithm and how they change for the different test runs. The first itemization will present the parameters for both approaches, the approach with the consecutive for-loop as well as the approach with the nested for-loop:

- $\boldsymbol{\beta}$ *(double beta)*: The parameter beta is regulating the steepness of the geometric distribution that is used for the randomization of the Savings Heuristic. This parameter is a decimal-number between 0 and 1. If $= 0$, each savings-value has the same probability to get chosen of the savings algorithm. For the case $= 1$, the savings-list is processed always starting with the highest savings-value.

- ***Maximal Runs*** *(int maxRuns)*: The maximal runs parameter defines how many solutions the algorithm can create before it terminates. The setting of this parameter is for all instances 450. The reason for choosing this amount runs is the computational time, however, more runs would improve the solution.

- ***Seed*** *(int seed)*: For each test-instance 10 test-runs with different seeds are processed. The seed regulates which numbers are generated by the random number generator. Therefore, changes in the seed affect the solutions of the corresponding test-runs.

Two additional parameters for the consecutive for-loop determine the runtime of the loops:

- ***Maximal Runs for Loop A*** *(int maxRunsA)*: The maximum runs for the first for-loop, For-loop A, are defined with $maxRunsA = 225$.

- ***Maximal Runs for Loop B*** *(int maxRunsB)*: Consequently, the maximum runs for the second for-loop, For-loop B, are defined with $maxRunsB = maxRuns - maxRunsA$.

Concerning the test-runs, the loop size of the consecutive for-loop is defined equally for Loop A and Loop B. Accordingly, other parameter-settings are viable as well, but have not been tested in this thesis.

For the nested for-loop two parameters, which are regulating how much iterations each loop has to execute, are implemented:

- ***Maximal Runs for the outer loop*** *(int maxRunsOut)*: The maximum runs for the outer loop are predefined. For each test-run, 5 iterations with a different setting are done. Therefore, $maxRunsOut = \{2, 4, 8, 16, 32\}$.
- ***Maximal Runs for the inner loop*** *(int maxRunsIn)*: The number of runs for the inner for-loop is defined with $maxRunsIn = maxRuns\ maxRunsOut$. Because the variable for the run time – condition has to be an integer number, for $maxRunsOut = \{4, 8, 16, 32\}$ in sum only 448 runs are done.

4.1.3 Results

The variants of the algorithm as described in Chapter 3, are first tested with the following settings for the parameters. is set for all test-runs to 0.5. In addition, all tests are performed 10 times with different random numbers. The reason for that is, to observe the performance of the different solution approaches.

Table 4.2 shows the solutions for the Savings Algorithm, the biased randomized Savings Algorithm, the three different variants of the algorithm with consecutive for-loops as well as the three different variants of the algorithm with nested for-loops. The best result is highlighted for every row. As the table indicates, the variants with learning-mechanism are outperforming the approaches using the *CWS* and the *BR-CWS*. Furthermore, it can be seen that especially the variant *N-AM* is performing well and delivering the best results of all six variations of the algorithm.

Table 4.2: Results for the consecutive for-loop with = 0.5

			Consecutive for-loop			Nested for-loop			Comparison	
Instance	CWS	BR-CWS	C-AM	C-BL	C-BN	N-AM	N-BL	N-BN	%	BKS
CB 1	592.09	548.49	547.27	542.64	546.46	550.19	**537.75**	547.27	2.51	524.61
CB 2	904.68	865.93	859.22	861.64	865.93	857.56	**853.11**	862.81	2.14	835.26
CB 3	888.27	856.34	855.71	**849.38**	855.71	850.28	856.34	856.34	2.81	826.14
CB 4	1,096.26	1,066.72	1,066.72	1,066.05	1,065.64	1,069.47	1,066.30	**1,065.40**	3.60	1,028.42
CB 5	1,426.02	1,372.20	**1,354.65**	1,361.78	1,369.71	1,356.84	1,363.14	1,364.34	4.89	1,291.45
CB 6	622.63	579.87	587.42	576.71	589.29	**570.05**	582.78	580.48	2.63	555.43
CB 7	972.39	945.28	939.63	939.63	945.29	**934.13**	943.77	945.29	2.69	909.63
CB 8	963.67	920.64	909.17	907.87	919.48	**896.94**	915.79	911.37	3.58	865.94
CB 9	1,300.82	1,231.94	1,232.34	1,231.11	1,230.97	**1,224.77**	1,236.78	1,227.70	5.35	1,162.55
CB 10	1,538.76	1,489.79	1,476.81	1,464.63	1,480.97	1,468.15	**1,459.96**	1,483.19	4.59	1,395.85
CB 11	1,093.23	1,054.82	1,054.82	**1,054.73**	1,057.86	1,055.42	1,057.34	1,054.82	1.21	1,042.11
CB 12	892.46	829.37	826.26	830.40	830.40	**826.26**	**826.26**	829.37	0.82	819.56
CB 13	1,588.40	1,570.49	1,569.50	1,569.06	1,569.88	**1,569.00**	1,570.55	1,569.50	1.81	1,541.14
CB 14	956.91	871.61	869.31	870.50	872.64	**866.95**	869.27	871.61	0.07	866.37
Average gap BKS %			3.53	3.30	3.91	3.07	3.39	3.64		

After the first tests it can be stated that the approach with nested for-loops is more promising than the approach with consecutive for-loops. To verify this assumption, both variants are tested with different . As it can be seen in Table 4.3 and Table 4.4, test-runs with iterating from 0 to 1 in steps of 0.1 were performed. All test-runs were performed 10 times with different random numbers.

Table 4.3 is showing the results for the approach with consecutive for-loops. It shows that the difference to the best known solution is between 0.25 and 4.66 percent. The most promising results are reached by the algorithms *C-AM* and *C-BL*, whereas *C-AM* shows the best results for the average GAP to the best known solution considering all instances.

The parameter for this attempt can be discussed as follows: the optimal beta for the different instances is varying between 0.1 and 0.5. The beta-setting remains relatively stable in the three different versions of the consecutive for-loop. Moreover, the most successful -setting is between 0.1 and 0.3. The reason for that is explained in the following subsection with more detailed -tests.

Table 4.3: Results for the consecutive loop with different

	C-AM			C-BL			C-BN		
Inst	Sol		%	Sol		%	Sol		%
CB 1	538.74	0.3	2.69	**538.20**	0.6	2.59	538.74	0.2	2.69
CB 2	**850.64**	0.1	1.84	860.74	0.1	3.05	853.51	0.1	2.18
CB 3	855.72	0.5	3.58	**847.52**	0.2	2.59	**847.52**	0.2	2.59
CB 4	1,064.90	0.3	3.55	**1,062.68**	0.2	3.33	1,063.17	0.3	3.38
CB 5	**1,346.14**	0.3	4.23	1,350.88	0.1	4.60	1,357.32	0.1	5.10
CB 6	570.03	0.1	2.63	569.61	0.1	2.55	**561.22**	0.1	1.04
CB 7	**932.72**	0.3	2.54	939.43	0.2	3.28	935.25	0.2	2.82
CB 8	896.05	0.1	3.48	**895.50**	0.5	3.41	903.10	0.6	4.29
CB 9	1,204.16	0.6	3.58	**1,202.26**	0.2	3.42	1,218.81	0.2	4.84
CB 10	1,462.92	0.2	4.80	**1,460.90**	0.3	4.66	1,471.56	0.4	5.42
CB 11	**1,052.78**	0.3	1.02	1,058.01	0.1	1.53	1,054.55	0.1	1.19
CB 12	**826.26**	0.5	0.82	873.47	0.3	6.58	**826.26**	0.2	0.82
CB 13	**1,567.40**	0.2	1.70	**1,567.40**	0.1	1.70	1,568.68	0.2	1.79
CB 14	**868.50**	0.2	0.25	931.86	0.4	7.56	**868.50**	0.3	0.25
Average GAP BKS %			2,62			3,63			2,74

Table 4.4 shows the results of the respective -tests for the approach with the nested for-loop. Taken together, it can be shown that this approach reaches better results than the approach with consecutive for-loops. The worst result is achieved for instance *CB 4* and is 2.85 percent away from the best known solution. In one case the algorithm has reached the best known solution and for a second case it differs only by 0.07 percent. In summary, the best variation is algorithm *N-AM*.

For the test-runs the best are varying again from 0.1 to 0.6. For the most cases the is between 0.1 and 0.2. The adjustment for the outer for-loop is varying as well, but only reaching 32 iterations for 3 cases. The most solutions are not exceeding 8 iterations on the outer loop.

Table 4.4: Results for the nested for-loop and different as well as different for-loop-conditions for the outer for-loop (LR)

	N-AM				N-BL				N-BN			
Inst	Sol	Param.		%	Sol	Param.		%	Sol	Param.		%
			LR				LR				LR	
CB 1	**531.75**	0.2	8	1.36	**531.75**	0.5	8	1.36	538.11	0.1	16	2.57
CB 2	**842.36**	0.1	8	0.85	846.21	0.1	2	1.31	851.17	0.1	16	1.90
CB 3	844.42	0.2	2	2.21	846.57	0.2	4	2.47	**839.69**	0.1	8	1.64
CB 4	1,060.18	0.2	4	3.09	**1,057.77**	0.1	2	2.85	1,063.17	0.3	2	3.38
CB 5	**1,324.56**	0.2	4	2.56	1,337.33	0.1	4	3.55	1,356.57	0.2	2	5.04
CB 6	**555.43**	0.2	2	-	557.89	0.2	4	0.44	575.37	0.1	2	3.59
CB 7	**927.67**	0.1	4	1.98	931.74	0.1	2	2.43	929.04	0.1	16	2.13
CB 8	**882.52**	0.1	2	1.92	884.86	0.1	4	2.19	896.39	0.1	32	3.52
CB 9	1,196.55	0.1	4	2.92	1,197.99	0.2	2	3.05	1,216.38	0.2	2	4.63
CB 10	**1,433.82**	0.2	2	2.72	1,449.57	0.2	4	3.85	1,463.88	0.4	4	4.87
CB 11	1,050.60	0.2	2	0.81	**1,050.40**	0.2	2	0.80	1,052.43	0.2	32	0.99
CB 12	**823.09**	0.4	32	0.43	826.26	0.2	8	0.82	826.26	0.2	2	0.82
CB 13	1,567.14	0.6	2	1.69	**1,564.95**	0.2	2	1.54	1,567.65	0.3	16	1.72
CB 14	**866.95**	0.5	8	0.07	868.50	0.2	8	0.25	868.50	0.3	2	0.25
Average GAP BKS %				1,62				1,92				2,65

Beta Test

To test the adjustment of in more detail, gets iterated in steps of 0.01 from 0.01 until 0.99. To visualize the testing procedure, Algorithm 7 gives an overview of how the parameter change in the loops and how the solutions are saved.

For the tests only three problem instances have been selected. The reason for that is that otherwise it would exceed the scope of work for the master thesis. Test instance *CB 7* was selected, because it was the only instance which has exceeded the vehicle number restriction, as a result of poor route merging during the algorithm. Moreover, it has to be noted that the Savings Algorithm is usually not designed to solve problems with a fixed vehicle fleet. The instance *CB 10* was chosen, because the solution-cost was high and being one of the larger instances, with $n = 199$. *CB 14* was tested, because it is already close to the best known solution and it would be interesting to see if it is possible to reach it, if parameters are adjusted accurately. In summary, all three test-instances chosen have distance-constraints.

Algorithm 7 N-AM

```
procedure BETA TEST
    for beta = 0.01; beta < 1; beta++ do
        maxRuns = 450;
        maxLearnruns = 1; //outer loop
        for i = 0; i < 5; i++ do
            maxLearnruns = maxLearnruns*2;
            innerRuns = maxRuns / maxLearnruns;
            process algorithm;
            save best result;
        end for
        print Solution for beta;
    end for
    return bestSol;
end procedure
```

In addition, only the algorithm *N-AM* was used for testing, because it has already performed best on all three instances and might provide the most promising results.

Table 4.5 gives an overview of the best solutions for the detailed -test and the parameters used to obtain the results. Table 4.5 shows that the solution for *CB 7* has improved by 1 percent. Also the routing has improved and therefore the vehicle number is meeting the amount of routes now. *CB 10* has only slightly improved the solution in comparison to the former reached values for a of 0.5. Instance *CB 14* has not improved the solution at all and is still 0.07 % away from the best known result.

Table 4.5: Solution for the -tests

	N-AM				
Inst	Sol	Parameter		%	Comparison
			LR		to = 0.5 in %
CB7	918.46	0.07	32	0.97	1.98
CB10	1,432.59	0.06	4	2.63	2.72
CB14	866.95	0.50	8	0.07	0.07

The graphs in Figure 4.1, Figure 4.2 and Figure 4.3 show the -test in more detail. The results of the test will be described subsequently in the following itemization:

- Figure 4.1 shows how the solution cost for *CB 7* is changing according to the different tested. The best solution is reached with a of 0.07. The graph is fluctuating considerably and therefore the next best solutions are at different (e.g. 0.27, 0.23, 0.41, 0.51, ...). The coefficient of determination shows that the regression line does

not approximate the data points well, but a trend can be seen that the solution cost rises with increasing .

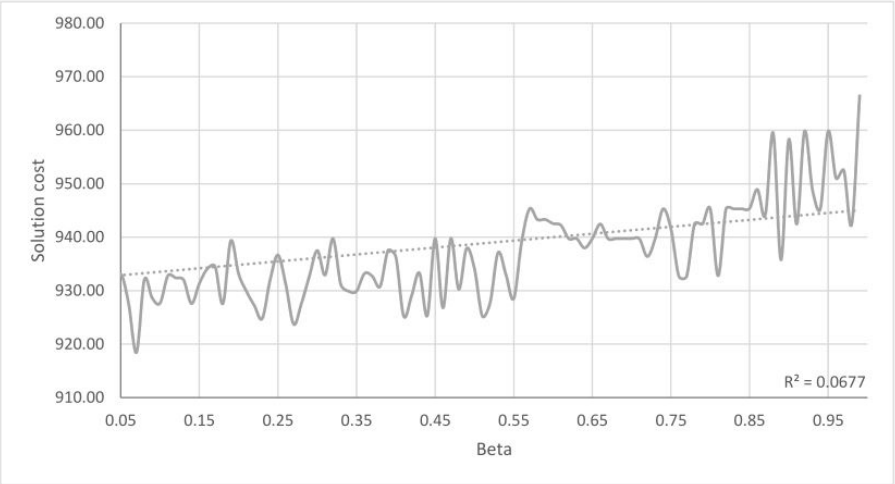

Figure 4.1: Testing for Christofides_ 7

- Figure 4.2 presents the graph for *CB 10*. The solution is slightly getting worse with increasing . The best result is reached with a of 0.06. Other promising are within the range from 0.05 to 0.22. Moreover, this graph is fluctuating as well as the graph in Figure 4.1. The regression line fits for 38 % of the data, but the model can not explain the correlation for the variables. For Figure 4.2, it can be seen that the solution cost is increasing with rising .

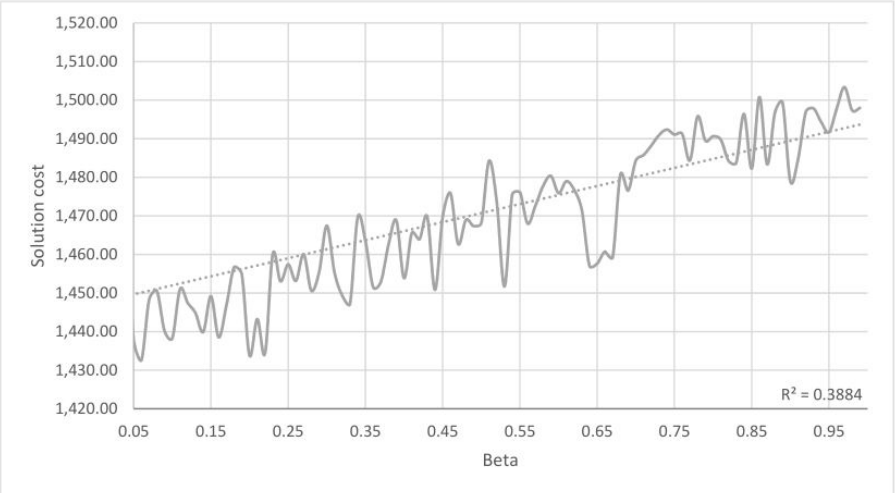

Figure 4.2: Testing for Christofides_ 10

- The graph in Figure 4.3 presents the results for *CB 14*. It is steadier than the graphs for the two other test-instances. The best solution is reached with different betas reaching from 0.22 until 0.52. The coefficient of determination shows that the regression line does not approximate the data points.

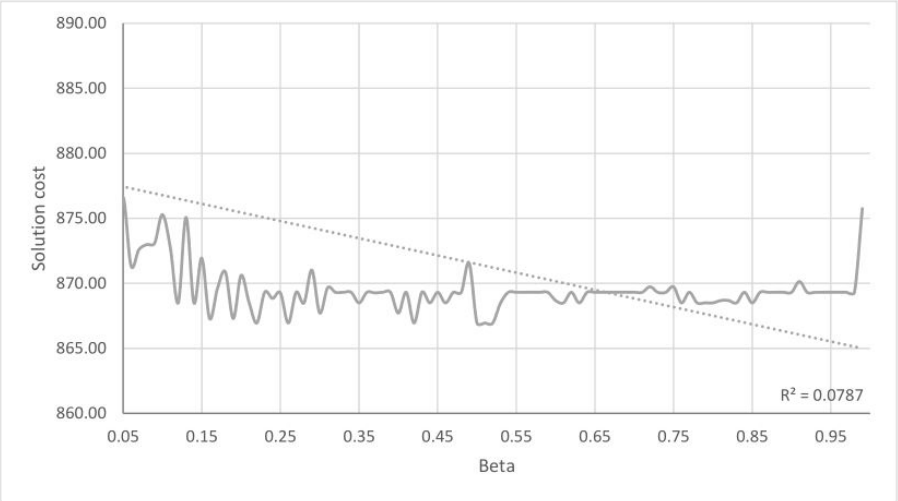

Figure 4.3: Testing for Christofides_ 14

To discuss the influence of the -adjustment Table 4.6 is providing a comparison between two different settings. The first 20 savings-values for the test-instance *CB 1* are shown in the table. For each savings-value the following data is shown:

- ***Vertices***: The first 20 edges in the savings-list are presented in the first column.
- ***Sav-List***: This column presents the corresponding savings-value for each pair of vertices.
- ***New-List***: This column presents the adapted savings-value for each pair of vertices. For this example the savings-value was adapted after 225 runs to gather information. The factor to adapt the values was calculated by dividing the arithmetic mean over all solutions with the arithmetic mean for the corresponding edge.
- ***In Sol***: In this column it is counted how often the vertice-pair has appeared in the 225 solutions.
- ***Sum***: The counter for *in Solution* can be summed up for reversed vertices (e.g. 35-36 and 36-35 excluding each other).

Table 4.6: Comparison between = 0.5 and = 0.1 for test-instance *CB 1*

	= 0.5				= 0.1			
	Sav-List	New-List	In Sol	Sum	Sav-List	New-List	In Sol	Sum
35-36	77.26	77.28	149		77.26	78.03	74	
36-35	77.26	77.39	66	215	77.26	77.93	65	139
20-35	64.79	64.77	150		64.79	64.99	76	
35-20	64.79	64.81	66	216	64.79	64.64	65	141
3-36	64.41	64.39	68		64.41	64.33	68	
36-3	64.41	64.42	147	215	64.41	64.44	59	127
20-36	63.85	63.46	7		63.85	63.55	52	
36-20	63.85	63.15	11	18	63.85	63.45	53	105
19-40	63.48	63.48	140		63.48	63.83	97	
40-19	63.48	63.47	84	224	63.48	63.45	78	175
3-35	61.92	61.84	6		61.92	61.69	51	
35-3	61.92	62.41	4	10	61.92	63.44	46	97
40-41	60.60	60.59	139		60.60	60.89	81	
41-40	60.60	60.58	80	219	60.60	60.33	70	151
33-39	58.36	58.37	137		58.36	58.25	75	
39-33	58.36	58.31	51	188	58.36	58.36	73	148
33-45	58.32	58.28	85		58.32	58.33	103	
45-33	58.32	58.35	140	225	58.32	58.36	115	218
19-41	57.39	57.69	4		57.39	57.11	53	
41-19	57.39	58.18	3	7	57.39	57.21	57	110

As indicated in the table, the manipulation of the savings-value is changing the value only slightly. Hence, the value moves for some steps in the ranking of the list. A remarkable point is that for a of 0.5 the algorithm is preferring certain edges and therefore, some edges are disadvantageous. For example, the selection of edge $(35-36)$ and edge $(36-3)$ eliminates the possibility of choosing edge $(20-36)$. On the other hand, if is set at 0.1, the selection is more randomized and all edges have a better chance of being selected.

Runtime Test

To improve the results, tests are provided, which increase the iterations of the algorithm. The parameter *maxRuns* is increased stepwise by 225 iterations, starting with 450 iterations and ending with 3600 iterations. To visualize the testing procedure Algorithm 8 gives an overview of how the parameters change within the loops and how the solutions are saved.

Algorithm 8 N-AM

```
procedure MaxRuns Test
    maxRuns = 225;
    for i = 0; i < 15; i++ do
        maxRuns = maxRuns + 225;
        beta = betaStart;
        for j = 0; j < betaRange; j++ do
            beta = beta + 0.01;
            maxLearnruns = 1; //outer loop
            for k = 0; k < 5; k++ do
                maxLearnruns = maxLearnruns*2;
                innerRuns = maxRuns / maxLearnruns;
                process algorithm;
                save best result;
            end for
        end for
        print Solution for maxRuns;
    end for
    return bestSol;
end procedure
```

The test-instances remain the same as for the -tests before, but the is tested for a smaller range. The reason for that is to minimize the computational time and that the tests before already highlighted the most promising -adjustments. To summarize the tests for and *maxRuns*, Table 4.7 provides an overview over the best results and the corresponding parameter. The table shows that the result for *CB 7* remained the same as for the betatest. *CB 10* improved the solution. Accordingly, the algorithm is reaching a result which is only 2.23 percent away from the best known result. For *CB 14* can be seen again that no improvements could be achieved by increasing the run time.

Table 4.7: Solutions for and *maxRuns*

Inst	N-AM					Comparison
	Sol	Parameter		%		
		LR	Iter		to	= 0.5 in %
CB7	918,46	0,07	32	450	0,97	1,98
CB10	1426,74	0,07	2	3600	2,21	2,72
CB14	866,95	0,5	8	450	0,07	0,07

- Figure 4.4 shows the results for *CB 7*. The solution cost for this test-instance remains steady and does not change considerably. The fact that the solution with the least

iterations performed best is coincidence. The explanation for that is, that by increasing the runtime of the algorithm, the corresponding loop-sizes are increased as well. Therefore, the random numbers provided by the generator are different. Accordingly, the random-numbers generated for each test-run differ and this kind of result can occur. The -range for this test was 0.06 – 0.30. The graph for the is fluctuating and no trend is visible.

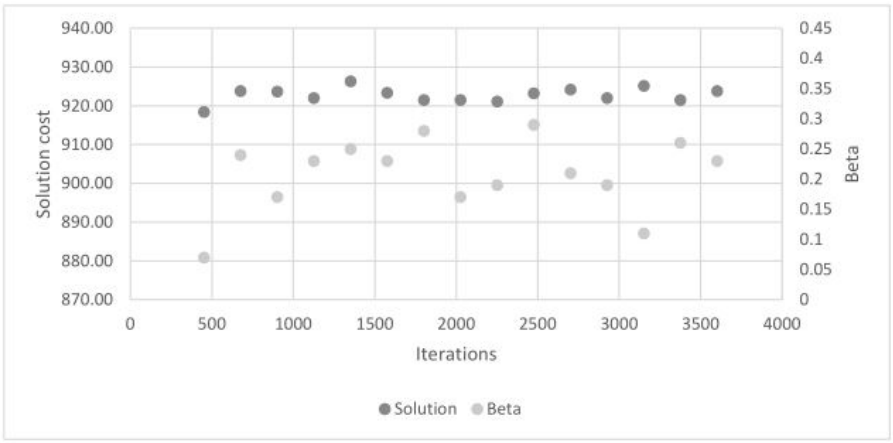

Figure 4.4: Testing and *maxRuns* for Christofides _ 7

- Figure 4.5 presents the results for *CB 10*. The tested -range for this results was form 0.06 - 0.21. The solution cost shows a decreasing trend with increasing run-time. The graphs for as well as for the solution cost are fluctuating.

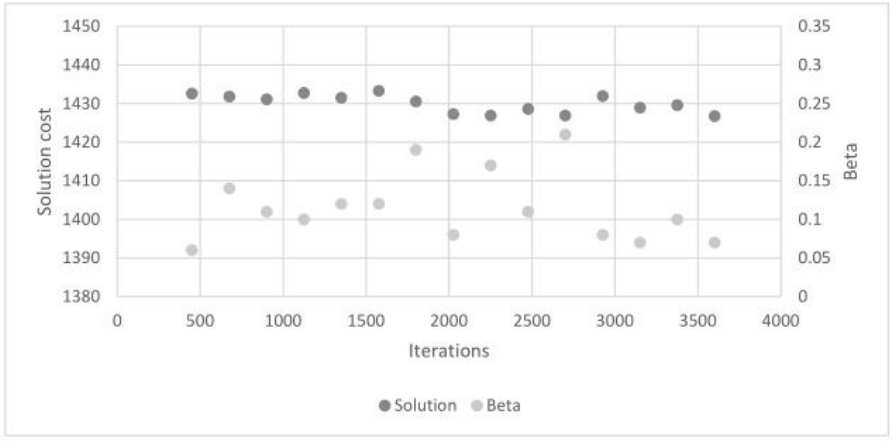

Figure 4.5: Testing and *maxRuns* for Christofides_ 10

- Figure 4.6 shows that the solution-cost for *CB 14* remains the same with increasing run-time. The beta-range for this test was 0.43 - 0.58. Hence, it can be stated that the algorithm finds the best solution faster with increasing run-time because the is decreasing over run-time.

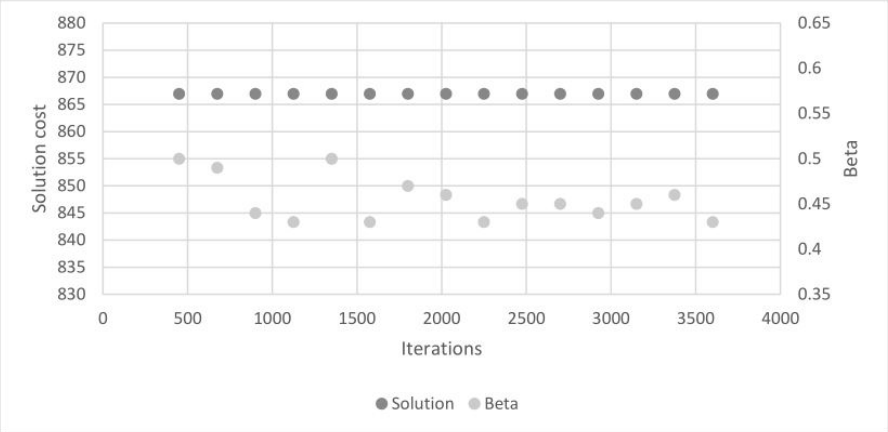

Figure 4.6: Testing and *maxRuns* for Christofides_ 14

4.2 DARP-Tests

The section describes the results for the many-to-many DARP discussed in Fikar and Hirsch (2014). Subsection 4.2.1 describes the test instances used for this section. Subsection 4.2.2 presents the parameter used for the algorithm. At the end of this chapter, in Subsection 4.2.3 the obtained results for this problem are discussed and compared to the results of Fikar and Hirsch (2014). The solution approach is coded in *C++* with Microsoft Visual Studio 2012. For two solution approaches FICO Xpress-BCL is used to solve the set-partitioning problem to select the walking routes. The system info of the computer used for the implementation is: Intel Core i7-3930K, 32 GB RAM, with MS-Windows 7 as operating system and 6 threads operating in parallel. The computation time for the algorithm is under one minute, depending on the instance size. The instances used for the tests are available at https://www.wiso.boku.ac.at/en/production-and-logistics/research/instances/.

4.2.1 Instances

To test the algorithm, Fikar and Hirsch (2014) have provided 30 test-instances. They were created based on real-world data and by using statistical probabilities. Therefore, the instances are not traceable to any personal information of patients. The halve of the instances are based on urban areas and are therefore indicated with *U*. Accordingly the other instances are based on sub-urban regions and indicated with *S*. The problem sizes n are set to 75, 100 and 125 jobs. Consequently, each problem size is dedicated 5 times to urban and 5 times to sub-urban areas. The number of vehicles m is set to 2 for all test-instances. The number of nurses m is varying, depending on the instance size, from 24 to 40 nurses. Each instance is defined over three files which are: (i) a distance-matrix for walking, (ii) a distance-matrix for driving and (iii) a description for the different jobs. The distances were calculated using map-data and assuming a walking speed of 3.6 km/h. The job description defines the time window for the job with earliest and latest start time e_i, l_i, the required qualification level ${t^{\mathrm{J}}}_i$ for the nurse associated with the job and the service time d_i. Furthermore, the vehicle capacity C is 6. Considering the driver the vehicle can transport 5 nurses. Additionally, conditions for the nurses are: maximum working time H of 600 min; maximum working time without a break R of 360 min; the time for a break r equals 30 min; W is set to 15 min and representing the maximum waiting time between each pickup and delivery; maximum walking duration between two jobs F is set to 10 min; maximum walking between each pickup and delivery U is 20 min; and a parameter for the maximal detour between pickup and delivery L equals 15 min.

4.2.2 Parameter setting

In this subsection, the parameters which are adjusted while testing the algorithm, are discussed.

- $\boldsymbol{\alpha}$ *(double alpha)*: The parameter is regulating the steepness of the geometric distribution which is used for the selection of the walking-routes. This parameter is a

decimal-number between 0 and 1 and tested in steps of 0.1 for all test-instances.

- β *(double beta)*: In accordance with Fikar and Hirsch (2014) the parameter is set to 0.45. It is adjusting the steepness of the geometric distribution, which is used to randomize the savings-list.

- ***Maximal Runs*** *(int maxRuns)*: This parameter defines how much solutions the algorithm creates before it terminates. In contrast to the tests performed in Section 4.1, maxRuns is set to 4500 iterations. The reason for that is, that far better results are expected, because the problem is more complex and the results are depending on more parameter. Moreover, the walking-routes are selected of a large pool and with increasing loop-size the algorithm is able to test out more combinations.

- ***Maximal Runs for the outer loop*** *(int maxLearnRuns)*: The maximum runs for the outer loop are predefined with 4 iterations. Test results before showed that this setting provides good results.

- ***Maximal Runs for the inner loop*** *(int maxIterations)*: The iterations for the inner for-loop are defined with *maxRuns/maxLearnRuns*. Using the current parameter setting, 1125 runs are performed for each inner loop.

- ***Seed*** *(int seed)*: For each test 10 runs with different seeds are processed. The seed regulates which numbers are generated of the random number generator.

4.2.3 Results

The solution approach for the many to many DARP is only attended with algorithm *N-AM*, because it has shown the most promising results for the tests performed in Section 4.1. The algorithm is tested for different settings ranging from 0 to 1 and increasing in steps of 0.1. Table 4.8 shows the results for this tests. The tests were done considering four variants: the first two columns contain results without set-partioning to select the walking routes; the other two variants are basing their consecutive search-process on preselected walking routes obtained with a solver software. The variants used are the BR-CWS and N-AM. Overall, the results are compared to the best results provided in the work of Fikar and Hirsch (2014).

As can be seen in Table 4.8, the results are printed in different styles. If feasible solutions were found they have been saved and are printed in black. The best feasible solution is highlighted and printed bold. If no feasible solution was found, the best infeasible solution is printed in grey and marked with an asterisk. It has to be mentioned that it does not matter for the following metaheuristic if the solution is feasible or not, because it optimizes the routing anyhow. The more important fact is that the objective value is as small as possible, which is easier to obtain, if infeasible solutions are considered as well.

As a result, it can be seen that for the two tests without selecting the walking-routes in advance, N-AM reaches far better results than the variant with the BR-CWS which is choosing the walking-routes randomly. Therefore, it can be seen that the learning mechanism

for choosing the walking-routes shows good results. Another interesting fact is that the algorithm finds a feasible solution for all 30 test-instances.

The two test-runs which are selecting the walking-routes using a solver software are showing that the solution improves another step. With these approaches the initial-solution is already between 5 and 10 % close to the best results shown in Fikar and Hirsch (2014).

Table 4.8: Best feasible results for Biased Randomization and Nested-For Loop Modification of the Savings Heuristic

	Without Set-Partitioning		Set-Partitioning		Comparision	
Inst	BR	N-AM	BR	N-Am	Best Known	%
U1-n75-k2-m12-8-4	802*	642	**562**	**562**	521	7.87
U2-n75-k2-m12-8-4	877*	632	566	**564**	532	6.02
U3-n75-k2-m12-8-4	812*	641	**558**	564	526	6.08
U4-n75-k2-m12-8-4	812	631	568	**566**	539	5.01
U5-n75-k2-m12-8-4	880	663	**607**	609	571	6.30
U1-n100-k2-m16-10-6	1190*	889	**752**	784	702	7.12
U2-n100-k2-m16-10-6	1210*	869	754	**744**	707	5.23
U3-n100-k2-m16-10-6	1127*	**832**	741*	740*	700	18.86
U4-n100-k2-m16-10-6	1216*	848	**720**	724	690	4.35
U5-n100-k2-m16-10-6	1100*	883	**775**	**775**	736	5.30
U1-n125-k2-m20-12-8	1511*	1042	**854**	855	817	4.53
U2-n125-k2-m20-12-8	1494*	1076	893*	**991**	834	18.82
U3-n125-k2-m20-12-8	1714*	1067	**886**	911	838	5.73
U4-n125-k2-m20-12-8	1617*	1084	874	**867**	817	6.12
U5-n125-k2-m20-12-8	1566*	1057	**842**	847	791	6.45
S1-n75-k2-m12-8-4	1164*	906	**862**	953	825	4.48
S2-n75-k2-m12-8-4	1122*	860	**801**	812	755	6.09
S3-n75-k2-m12-8-4	1195*	867	**831**	**831**	793	4.79
S4-n75-k2-m12-8-4	1181*	902	**863**	888	792	8.96
S5-n75-k2-m12-8-4	1130*	937	936	**932**	821	13.52
S1-n100-k2-m16-10-6	1701*	1222	**1087**	1105	1027	5.84
S2-n100-k2-m16-10-6	1701*	1181	**1061**	**1061**	998	6.31
S3-n100-k2-m16-10-6	1672*	1195	**1102**	1148	1039	6.06
S4-n100-k2-m16-10-6	1691*	1154	1056	**1054**	983	7.22
S5-n100-k2-m16-10-6	1695*	**1309**	1194*	1194*	1096	19.43
S1-n125-k2-m20-12-8	2242*	**1640**	1309*	1316*	1224	33.99
S2-n125-k2-m20-12-8	2437*	**1488**	1341*	1343*	1185	25.57
S3-n125-k2-m20-12-8	2552*	**1754**	1475*	1419*	1201	46.04
S4-n125-k2-m20-12-8	2434*	**1546**	1383*	1392*	1225	26.20
S5-n125-k2-m20-12-8	2468*	**1587**	1337*	1320*	1213	30.83

To complete the comparison between these four solution-approaches Table 4.9 is providing an overview, which shows only the best solution obtained. For this table, it does not matter if the solution is feasible or not.

Table 4.9: Best results for Biased Randomization and Nested-For Loop Modification of the Savings Heuristic without considering feasibility

	Without Set-Partitioning		Set-Partitioning		Comparision	
Inst	BR	N-AM	BR	N-Am	Best Known	%
U1-n75-k2-m12-8-4	802	642	**562**	**562**	521	7.87
U2-n75-k2-m12-8-4	877	632	566	**564**	532	6.02
U3-n75-k2-m12-8-4	812	641	**558**	564	526	6.08
U4-n75-k2-m12-8-4	749	631	**562**	**562**	539	4.27
U5-n75-k2-m12-8-4	808	662	602	**601**	571	5.25
U1-n100-k2-m16-10-6	1190	878	**737**	753	702	4.99
U2-n100-k2-m16-10-6	1210	859	**741**	744	707	4.81
U3-n100-k2-m16-10-6	1127	832	741	**740**	700	5.71
U4-n100-k2-m16-10-6	1216	848	**720**	724	690	4.35
U5-n100-k2-m16-10-6	1100	883	**775**	**775**	736	5.30
U1-n125-k2-m20-12-8	1511	1042	**854**	855	817	4.53
U2-n125-k2-m20-12-8	1494	1076	893	**886**	834	6.24
U3-n125-k2-m20-12-8	1714	1067	**886**	907	838	5.73
U4-n125-k2-m20-12-8	1617	1084	874	**867**	817	6.12
U5-n125-k2-m20-12-8	1566	1053	**842**	847	791	6.45
S1-n75-k2-m12-8-4	1164	889	841	**838**	825	1.58
S2-n75-k2-m12-8-4	1122	860	**801**	809	755	6.09
S3-n75-k2-m12-8-4	1195	857	**831**	**831**	793	4.79
S4-n75-k2-m12-8-4	1181	892	**850**	854	792	7.32
S5-n75-k2-m12-8-4	1130	914	881	**854**	821	4.02
S1-n100-k2-m16-10-6	1701	1205	**1087**	1088	1027	5.84
S2-n100-k2-m16-10-6	1701	1166	**1044**	**1044**	998	4.61
S3-n100-k2-m16-10-6	1672	1195	**1102**	1120	1039	6.06
S4-n100-k2-m16-10-6	1691	1154	1056	**1054**	983	7.22
S5-n100-k2-m16-10-6	1695	1248	**1194**	**1194**	1096	8.94
S1-n125-k2-m20-12-8	2242	1488	**1309**	1316	1224	6.94
S2-n125-k2-m20-12-8	2437	1445	**1341**	1343	1185	13.16
S3-n125-k2-m20-12-8	2552	1563	1475	**1419**	1201	18.15
S4-n125-k2-m20-12-8	2434	1515	**1383**	1392	1225	12.90
S5-n125-k2-m20-12-8	2468	1454	1337	**1320**	1213	8.82

In Table 4.9 the overall results are far better, when infeasible routes are compared with the best known solution, obtained by Fikar and Hirsch (2014). In comparison to Table 4.8 it can

be seen that the results of the variant *N-AM* have improved by considering the best infeasible solution. Moreover, comparing the approaches with set-partitioning and this variant, it can be shown that for the small test-instances the difference in the obtained results is not high. Overall, good results are obtained for most of test-instances, excluding test-instances *S2* *S4* for 125 jobs.

To evaluate the solutions a function, adding penalty factors, is used that is allowing infeasible solutions using following factors: (i) violations in capacity, (ii) time windows and (iii) ride times as well as (iv) how many nurses more than available are needed for the service. Therefore, the penalty costs are added to the objective value.

To conclude this chapter, Table 4.10 gives an insight of how the learning mechanism works on three test-instances. The table shows how the algorithm improves the solutions of the four learning-iterations.

Table 4.10: Learning iterations for three test-instances

U1-n75-k2-m12-8-4	U1-n100-k2-m16-10-6	U1-n125-k2-m20-12-8
First Run		
786	1057	1378
726	1015	1363
705	988	1349
689	984	1208
678	971	1172
	963	1157
	956	1144
	952	1127
	941	
Second Run		
672	938	1126
664		1081
Third Run		
no improvements	915	no improvements
Fourth Run		
no improvements	no improvements	no improvements

Table 4.10 presents another behavior. It shows that after the first iteration of the learning cycle, the algorithm still improves the results. In the third iteration the algorithm can only find an improvement for the second test-instance.

5 Discussion and Outlook

In this master thesis six different attempts to a learning-based Savings Algorithm were implemented for two different problem definitions. On the one hand, multiple test-instances for a VRP were solved using the learning-based Savings Algorithm. On the other hand, a many to many DARP tackling a HHC problem was solved. In addition, a detailed description of both problems was provided. Furthermore, the method as well as the parameters to adjust the algorithm were described at great length. Moreover, this thesis introduced a possible way to replace the usage of a solver-software in the algorithm provided by Fikar and Hirsch (2014) for the selection of walking-routes.

The first tests on the algorithm dealt with a VRP. Therefore, the different variants are used to solve the Christofides et al (1979) test-instances. The usage of the algorithm improved the results in comparison to attempts with a CWS or a biased randomized CWS. The results provided by the different variants of the algorithm were in all cases close to the best known solution for the test-instances. For the further progress of the work the different approaches were evaluated and compared to each other. The approach with the most promising results, *N-AM*, was selected for more detailed analysis considering the implementation and the algorithm of Fikar and Hirsch (2014). Subsequently, the parameters were tested in more detail and correlations between the parameter-setting and the objective value were presented. In conclusion, the implementation of the algorithm is adaptable with less effort and easy to reproduce. Therefore, only two parameters have to be adjusted for the corresponding problem. Another positive aspect of the solution approach is the fast computational time, allowing it to generate a multitude of solutions in a short time.

The next step was to test the application of the algorithm for the DARP. It was tested on the instances and problem-definition provided by Fikar and Hirsch (2014). For this problem, not only a modified Savings Algorithm was introduced, but also an algorithm to replace the set-partitioning, which is used to select the walking-routes at the moment. The results for the final comparison showed that the approach provides promising solutions. Nevertheless, the results obtained by the solution approach with the pre-selection of promising walking-routes via set-partitioning showed better results. In summary, depending on the needs of the following metaheuristic, the algorithm can replace the solver-software, for the cost of a setback in the objective value of the solution.

In summary, the algorithm provided by this thesis can be applied for problems the home health care providers are facing nowadays. It is capable of providing good solutions for smaller test-instances, but the solutions fall off in quality with increasing problem-size and more narrow constraints. Moreover, the algorithm is a good choice for a construction-heuristic to create initial solutions for an ensuing metaheuristic. Therefore, it can be an asset for decision-makers to make use of this approach to evaluating their planning and scheduling. Considering the approach presented in Fikar and Hirsch (2014), benefits obtained by the results can be a decrease in the vehicle fleet as well as a relief for the staff members (e.g. no need of searching for parking spots and time to relax during the driving time).

A negative aspect of the algorithm is that the solution is strongly depending on the parameter-setting. Hence, different problem-instances need a fine-tuning of the parameters to obtain the best results. Another aspect, which has to be considered is that the algorithm is favoring edges while merging the routes. The reason for that is the selection-procedure of the Savings Algorithm.

Future work could consist of adding penalties to modify the selection of edges for the route-merging-process. Additionally, to improve the results obtained by the algorithm an improvement-heuristic with fast computational time could be added at the end of the algorithm. A possible method for that could be a *k-opt algorithm*. Hence, using an improvement-heuristic would only be beneficial for the VRP. For more complex problems, like the DARP, a metaheuristic is needed.

Bibliography

Begur SV, Miller DM, Weaver JR (1997) An integrated spatial dss for scheduling and routing home-health-care nurses. Interfaces 27(4):35–48

Bertels S, Fahle T (2006) A hybrid setup for a hybrid scenario: Combining heuristics for the home health care problem. Computers & Operations Research 33(10):2866–2890

Bredström D, Rönnqvist M (2008) Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. European Journal of Operational Research 191(1):19–31

Bräysy O, Wout D, Pentti N (2007) Municipal routing problems: a challenge for researchers and policy makers? Nautilus Academic Books, Zelzate p 330–347

Burke KE, Kendall G (2005) Search methodologies: Introductory tutorials in optimization and decision support techniques. Springer Science+Buisness Media, Inc Springer: USA

Cheng E, Rich JL (1998) A home health care routing and scheduling problem. Technical Report CAAM TR98-049, Rice University

Christofides N, Mingozzi A, Toth P (1979) The vehicle routing problem. Combinatorial Optimization Wiley, Chichester

Clarke G, Wright JW (1964) Scheduling of vehicles from a central depot to a number of delivery points. Operations Research 12(4):568–581

Cordeau JF (2006) A branch-and-cut algorithm for the dial-a-ride problem. Operations Research 54(3):573–586

Cordeau JF, Laporte G (2003) The dial-a-ride problem (darp): Variants, modeling issues and algorithms. Quarterly Journal of the Belgian, French and Italian Operations Research Societies 1(2):89–101

Cordeau JF, Laporte G (2007) The dial-a-ride problem: models and algorithms. Annals of Operations Research 153(1):29–46

Cordeau JF, Desaulniers G, Desrosiers J, Solomon MM, Soumis F (2002a) VRP with time windows. In: Toth P, Vigo D The Vehicle routing Problem SIAM Monographs on Discrete Mathematics and Appliactions SIAM Publishing: Philadelphia pp 157–193

Cordeau JF, Gendreau M, Laporte G, Potvin JY, Semet F (2002b) A guide to vehicle routing heuristics. Journal of the Operational Research Society pp 512–522

Cordeau JF, Laporte G, Savelsbergh MW, Vigo D (2007) Chapter 6 vehicle routing. In: Barnhart C, Laporte G (eds) Transportation, Handbooks in Operations Research and Management Science, vol 14, Elsevier, pp 367–428

Dantzig G, Ramser J (1959) The truck dispatching problem. Management Science 6(1):80–91

Dantzig GB, Wolfe P (1960) Decomposition principle for linear programs. Operations Research 8(1):101–111

Desaulniers G, Desrosiers J, Erdmann A, Solomon MM, Soumis F (2002) VRP with pickup and delivery. In: Toth P, Vigo D The Vehicle routing Problem SIAM Monographs on Discrete Mathematics and Appliactions SIAM Publishing: Philadelphia pp 225–242

Dowsland KA (2005) Classical techniques. Burke, K Edmund and Kendall, Graham Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques Springer Science+Buisness Media, Inc Springer: USA pp 19–68

Engeler K (2002) Mehrdepot-Tourenplanung mit Zeitfenstern. Josef Eul Verlag Lohmar Köln

Eveborn P, Flisberg P, Ronnqvist M (2006) Laps care–an operational system for staff planning of home care. European Journal of Operational Research 171(3):962–976

Fikar C, Hirsch P (2014) A matheuristic for routing real-world home service transport systems facilitating walking. Journal of Cleaner Production (0):–

Fisher ML, Jaikumar R (1981) A generalized assignment heuristic for vehicle routing. Networks 11(2):109–124

Gaskell T (1967) Bases for vehicle fleet scheduling. Operational Research Quarterly 18(3):281–295

Gendreau M, Laporte G, Potvin JY (2002) Metheuristics for the capacitated vrp. Toth, Paolo and Vigo Daniel The Vehicle routing Problem SIAM Monographs on Discrete Mathematics and Appliactions SIAM Publishing: Philadelphia pp 109–128

Gillett BE, Miller LR (1974) A heuristic algorithm for the vehicle-dispatch problem. Operations Research 22(2):340–349

Glover F (1990) Tabu search: A tutorial. Interfaces 20(4):74–94

Glover F, Laguna M (1997) Tabu Search. Kluwer Academic Publishers, Norwell, MA, USA

Groër C, Golden B, Wasil E (2011) A parallel algorithm for the vehicle routing problem. INFORMS Journal on Computing 23(2):315–330

Hansen P, Mladenovic N (2003) Variable neighborhood search. Handbook of Metaheuristics Kluwer Academic Pusblisher, New York pp 145–184

Healy P, Moll R (1995) A new extension of local search applied to the dial-a-ride problem. European Journal of Operational Research 83(1):83–104

Hiermann G, Prandtstetter M, Rendl A, Puchinger J, Raidl G (2013) Metaheuristics for solving a multimodal home-healthcare scheduling problem. Central European Journal of Operations Research pp 1–25

Hirsch P, Schweiger M (2014) Optimierungsmodelle und natürliche Ressourcen. Institut für Produktionswirtschaft und Logistik - BOKU Wien

Holland JH (1975) Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. Oxford, England: U Michigan Press

Juan A, Faulin J, Ferrer A, Lourenço H, Barrios B (2013) Mirha: multi-start biased randomization of heuristics with adaptive local search for solving non-smooth routing problems. TOP: An Official Journal of the Spanish Society of Statistics and Operations Research 21(1):109–132

Juan AA, Faulin J, Jorba J, Riera D, Masip D, Barrios B (2010) On the use of monte carlo simulation, cache and splitting techniques to improve the clarke and wright savings heuristics. Journal of the Operational Research Society 62(6):1085–1097

Kergosien Y, Lenté C, Billaut J (2009) Home health care problem: an extended multiple tratravel salesman problem. Multidisciplinary international conference on scheduling: theory and applications (MISTA 2009) pp 85–92

Kinderwater G, Savelsberg M (1997) Vehicle routing: handling edge exchanges. Aarts, EHL and Lenstra, JK, Local Search in Combinatorial Optimization

Landgraf A (1995) Vehicle routing: Implementierung und Analyse der "Parallel Savings Based Heuristic". Technische Universität Wien

Laporte G (1992) The vehicle routing problem: An overview of exact and approximate algorithms. European Journal of Operational Research 59(3):345 – 358

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

Laporte G, Semet F (2002) Classical heuristics for the capacitated VRP. In: Toth P, Vigo D The Vehicle routing Problem SIAM Monographs on Discrete Mathematics and Appliactions SIAM Publishing: Philadelphia pp 109–128

Laporte G, Gendreau M, Potvin JY, Semet F (2000) Classical and modern heuristics for the vehicle routing problem. International Transactions in Operational Research 7(4-5):285–300

Mankowska D, Meisel F, Bierwirth C (2014) The home health care routing and scheduling problem with interdependent services. Health Care Management Science 17(1):15–30

Parragh S, Doerner K, Hartl R (2008) A survey on pickup and delivery problems. Journal für Betriebswirtschaft 58(2):81–117

Rasmussen MS, Justesen T, Dohn A, Larsen J (2012) The home care crew scheduling problem: Preference-based visit clustering and temporal dependencies. European Journal of Operational Research 219(3):598–610

Rayward-Smith V, Osman I, Reeves C, Smith G (1996) Modern Heuristic Search Methods. John Wiley

Rest KD, Hirsch P (2013) Time-dependent travel times and multi-modal transport for daily home health care planning. TRISTAN VIII - The Eight Triennial Symposium on Transportation Analysis, San Pedro de Atacama, CHILE, JUN 09-14

Rest KD, Trautsamwieser A, Hirsch P (2012) Trends and risks in home health care. Journal of Humanitarian Logistics and Supply Chain Management 2(1):34 – 53

Tarricone R, Tsouros A (2008) Home care in europe: the solid facts. Regional Office for Europe of the World Health Organization, Copenhagen

Thompson PM, Psaraftis HN (1993) Cyclic transfer algorithm for multivehicle routing and scheduling problems. Operations Research 41(5):935–946

Toth P, Vigo D (2002a) An overview of vehicle routing problems. In: Toth P, Vigo D The Vehicle routing Problem SIAM Monographs on Discrete Mathematics and Appliactions SIAM Publishing: Philadelphia pp 1–26

Toth P, Vigo D (2002b) VRP with backhauls. In: Toth P, Vigo D The Vehicle routing Problem SIAM Monographs on Discrete Mathematics and Appliactions SIAM Publishing: Philadelphia pp 195–224

Trautsamwieser A, Hirsch P (2011) Optimization of daily scheduling for home health care services. Journal of Applied Operational Research 3(3):124–136

Trautsamwieser A, Hirsch P (2014) A branch-price-and-cut approach for solving the medium-term home health care planning problem. Networks

Van Breedam A (1994) An analysis of the behavior of heuristics for the vehicle routing problem for a selection of problems with vehicle-related, customer-related, and time-related constraints. PhD disseratation, University of Antwerp

Whitley D, Watson JP (2005) Compexity theory and the no free lunch theorem. Burke, K Edmund and Kendall, Graham Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques Springer Science+Buisness Media, Inc Springer: USA, PA, pp 317-339

Woodward CA, Abelson J, Tedford S, Hutchison B (2004) What is important to continuity in home care?: Perspectives of key stakeholders. Social Science & Medicine 58(1):177 – 192

Yellow P (1970) A computational modification to the savings method of vehicle scheduling. Operational Research Quarterly 21(2):281–283

List of Figures

List of Tables

List of Algorithms

Abbreviation

CP	Constraint Programming
CVRP	Capacitated Vehicle Routing Problem
CWS	Clark and Wright Savings Heuristic
DARP	Dial-a-Ride Problem
DCVRP	Distance-Constrained Capacitated Vehicle Routing Problem
GAP	Generalized Assignment Problem
HHC	Home Health Care
HHCRSP	Home Health Care Routing and Scheduling Problem
ILP	Integer Linear Programming
LP	Linear Programming
MILP	Mixed Integer Programming Model
VNS	Variable Neighborhood Search
VRP	Vehicle Routing Problem
VRPB	Vehicle Routing Problem with Backhauls
VRPB	Vehicle Routing Problem with Backhauls and Time Windows
VRPPD	Vehicle Routing Problem with Pickup and Delivery
VRPPDTW	Vehicle Routing Problem with Pickup and Delivery and Time Windows
VRPTW	Vehicle Routing Problem with Time Windows
WR	Walking-Routes