Insert here your thesis' task.

Czech Technical University in Prague
Faculty of Information Technology
Department of Theoretical Computer Science

Bachelor's thesis

Vehicle routing problem, its variants and solving methods

Michal Polívka

Supervisor: RNDr. Tomáš Valla, Ph.D.

10th May 2015

Acknowledgements

I would like to thank my supervisor Tomáš Valla for the time, support and thoughts throughout the thesis. I would also like to thank my family and friends for all their support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as school work under the provisions of Article 60(1) of the Act.

In Prague on 10th May 2015 .....

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Polívka, Michal. *Vehicle routing problem, its variants and solving methods.* Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2015.

Abstrakt

Tato práce studuje Vehicle Routing Problem (VRP) a jeho varianty. Pro kapacitní verzi problému (CVRP) implementujeme varianty používaných algoritmů a jako naše řešení navrhneme a implementujeme modifikaci dvou local search metod, které ještě nebyli na tento problém nikým použity. Na třech sadách instancí provedeme měření a porovnání těchto algoritmů s nejlepšími známými výsledky a free/open-source programy.

Klíčová slova optimalizace, lokální prohledávání, routovaní vozidel, CVRP, VRP

Abstract

This thesis studies Vehicle Routing Problem (VRP) and its variants. We also implement local search based algorithms for a capacitated version of the problem (CVRP). We propose and implement two algorithms, which are based on recently presented local search methods, that have not yet been applied to CVRP. On three sets of benchmark instances, we test our algorithms and compare them against best-known solutions and other solvers.

Keywords vehicle routing, optimization, local search, CVRP, VRP

Contents

List of Figures

Chapter 1

Introduction

In this thesis we focus on vehicle routing problem, its variants, free/open-source solvers and various solution methods and algorithms.

This first chapter gives an introduction to Vehicle Routing Problem (VRP). We describe a motivation and objectives of this thesis. We also outline basic definition and notation of the problem. After that we introduce the variants of the problem and in the last section we mention and review available free/open-source solvers.

In the second chapter we focus on Capacitated Vehicle Routing Problem (CVRP), which we first introduce together with a literature review and commonly used algorithms. In the next section we acquaint the reader with our implemented algorithms and strategies. We present three sets of benchmark instances, which are then used to compare the strength of our algorithms against the best known results and free/open-source solvers. Apart from commonly used algorithms we propose and implement two algorithms, which are based on recently presented local search methods, that have not yet been applied to CVRP.

1.1 Motivation and objectives

Vehicle routing problem has been extensively studied since 1959 (Dantzig and Ramser [16]), when it was defined under the name Truck Dispatching Problem. The problem appears in many forms in real life and solution to the problem can save companies a lot of money. It is an important problem in the fields of transportation, distribution, and logistics.

The problem can be simply describe as having a fleet of vehicles, number of customers and depot, each vehicle starts from a depot, visits some customers and returns back to the depot. At the end all customers should be served exactly once, the objective function should be minimal and the solution should be feasible with regard to the constraints. As it is classified as a combinatorial optimization and integer programming problem, the goal is to minimize

an objective function. This function can vary according to the exact problem specification, but mostly it is minimizing the sum of route lengths. The problem belongs to the class of NP-hard problems, therefore for large real-world problems with many customers the exact solutions are not applicable. Some part of the research is still developing new exact algorithms, which are able to solve larger problems, but other part is studying heuristic and meta-heuristic algorithms, which show excellent results in reasonable time. We study some of these heuristic and meta-heuristic algorithms and we apply them to a specific variant of the problem known as Capacitated Vehicle Routing Problem (CVRP). We have implemented some of the known algorithms and developed their variants.

We have also applied variants of two new algorithms (LAHC, SCHC), which are based on local search and have not been yet applied to CVRP. On random benchmark instances, we have obtained with these two algorithms an average gap A (Definition 1) equaled to 1.19 and 1.62 percent from best-known solutions. We have shown, that these algorithms perform better, than widely used SA, whose average gap is 2.61. We compare our algorithms with other free/open-source solvers obtaining the minimal average gap.

1.2 Definition

The definitions of the problem vary depending on the constraints, graph and objective function. The general problem is usually defined the same way as the CVRP variant. It is defined on a complete undirected graph $G = (V, E)$, where V is a set $\{0, \ldots, N\}$ of vertices. Each vertex i excluding vertex 0 represents a customer and its demand d_i. There are $N - 1$ customers. Vertex 0, where every route $R_i = \{v_0, v_1, \ldots, v_{\ell_i+1}\})$, $v_i \in V$, and ℓ_i is the length of the route R_i, starts and ends is called *depot* ($v_0 = v_{\ell_i+1} = 0$). Union of all R_i is equal to V and the intersection is equal to v_0, v_{ℓ_i+1}. E is a set of edges $e_{i,j}$, for $i, j \in V$ and every edge $e_{i,j}$ has assigned a cost $c_{i,j}$. The cost of the edge is symmetrical. A fleet of m vehicles, all of them having the same capacity K is dispatched from the depot. The input is G, N, K and we have to determine the output m, R, which satisfies all constraints. Specifically, we have to determine a number of vehicles m and their routes R_i, $i \in \{0, \ldots, m-1\}$, such that the solution is feasible with respect to Constraint 1–4.

- The basic constraint is on capacity, where the total demand of route does not exceed the vehicle capacity (Constraint 1)

  Constraint 1

$$\sum_{v \in R_i} d_v < K$$

- Each customer is visited only once by exactly one vehicle (Constraint 2–3)

  Constraint 2

$$\bigcup_{i=0,\ldots,m-1} R_i = V$$

  Constraint 3

$$R_i \cap R_j = \{0\}, for\{i = 0, \ldots, m-1\}, \{j = 0, \ldots, m-1\}, i \neq j$$

The solution consists of a set of m cycles sharing the depot. In many variants of the problem a common objective function is to minimize the total cost $C(S)$ of the solution S (Constraint 4). However the objective function can vary according to the problem specification.

Constraint 4

$$C(S) = \sum_{i=0}^{m} C(R_i), \text{where} C(R_i) = \sum_{k=0}^{l_i} c_{k,k+1}$$

1.3 VRP variants

Over the period of more than 50 years of research there has been described many variants of the previously defined problem. We mention few of them to show how wide and complex the problem is. For further information we advise the reader to read a book about VRP (Golden, Raghavan, and Wasil [21], Toth and Vigo [35]).

The oldest and most studied variants are:

Capacitated Vehicle Routing Problem (CVRP)
: The problem has been defined in previous section and we can find it in real-life problems, where shipment of products is involved. We study this problem in the next chapter.

Vehicle Routing Problem with Time Windows (VRPTW)
: VRPTW (Solomon [33]) is a variant, where the objective is to serve the demands of customers in predefined time windows. There has been variants with soft windows allowed (we can deliver in different time with some penalty).

Vehicle Routing Problem with Pick-Up and Delivery (VRPPD)
: VRPPD (Solomon and Desrosiers [34]) is similar to CVRP, but we have two types of items. One, which we need to deliver from the depot to customers and the other to deliver from customers to the depot.

Other studied variants include some of these features:

- Dynamic: some customers are not known in advance and are added in real-time.
- Heterogenous: there exist different types of vehicles with different capacities.
- Multiple depots: there exist M depots, which can or does not need to have assigned vehicles and we solve the problem as in CVRP case. There can be a variant, that a vehicle have to return back to the same depot from which started the route.
- Periodic: we have a list of customers to serve and times, when we can operate and we need to cut and solve the problem in D days.
- Split delivery: we can split the delivery from customer into more cars, so that one customer can be visited by more than one car.
- Stochastic: one or more components are random (customer i presented with probability p_i, random demands, random cost c_{ij},...). First solution is generated before knowing the values and second solution corrects that solution after knowing the random variables.
- With backhauls: similar to VRPPD, but with a restriction, that all demands on route r_i has to be completed before any pick-ups are made on that route.
- With satellite facilities: these can replenish a vehicle on a route, so that the vehicle can continue to serve customers until specified.

Because of the complex structure of real-life problems, there exists numerous combinations of these and other variants.

1.4 Free/Open-source solvers

There has been few free/open-source solvers, which can solve different types of VRP with different number of customers. For CVRP, we present at the end of this chapter results obtained on benchmark instances from some of these algorithms compared with our implemented algorithms.

We present a summary of the solvers, which implement some more complicated and promising algorithms (we have excluded some, which implement only basic heuristics) and their brief description obtained from the cited sources in addition with some extra findings.

Jsprit [31] Jsprit is a java based, open-source toolkit for solving rich traveling salesman (TSP) and vehicle routing problems (VRP). It is core meta-heuristic algorithm uses ruin-and-recreate principle (it destroys part of the solution and recreates it in another way). For more information about the algorithm visit the cited source. The development of the library is active and it is used by ODL Studio [3] software.

Open-VRP [6] Open-VRP is a framework to model and solve VRP-like problems for students, academics, businesses and hobbyist alike. It includes implementation of greedy heuristics and tabu search. It is written in LISP and there has not been any activity in development for more than 3 years.

OptaPlanner [32] OptaPlanner is a constraint satisfaction solver written in Java. It supports many different problems and includes their examples. It is still very active, but during our tests it has not proved very effective in solving CVRP. However, due to the fact that it is a general solver it gave quiet good results.

SYMPHONY [1] SYMPHONY is an open-source solver for mixed-integer linear programs (MILPs) written in C and hosted by COIN-OR website. It also supports a parallel execution.

VRP Spreadsheet Solver [17] The Microsoft Excel workbook VRP Spreadsheet Solver is presented as an open source unified platform for representing, solving, and visualising the results of VRP. It unifies Excel, public GIS and meta-heuristics. It is said, that it can solve VRP with up to 200 customers. However, it is available only to VeRoLog members, which we could obtain only by registering and attending some VeRoLog workshop or conference.

VRPH [22] VRPH is an open source library of heuristics for the capacitated Vehicle Routing Problem (VRP). It includes several example applications that can be used to quickly generate good solutions to VRP instances containing thousands of customer locations. It has been developed as part of Chris Groer is dissertation while at the University of Maryland with advisor Bruce Golden. It is currently hosted on COIN-OR website, but there is not any new version since the first release in 2009.

Vroom [30] Vroom contains different libraries and frameworks especially for solving a dynamic VRP. It has been implemented as a part of Ph.D. dissertation and there is not any sign of development going on.

Chapter 2

Capacitated Vehicle Routing Problem

2.1 Introduction

Capacitated Vehicle Routing Problem (CVRP) is one of the most studied variants of VRP. It is known to be NP-hard problem, which generalizes Travelling Salesman Problem (TSP).

The general VRP problem definition uses the definition of CVRP, which we have defined in chapter 1. The objective is to minimize the sum of the route lengths (Definition 4). Here we present a minimal version of a general integer programming formulation that has been described in Bodin, Golden, Assad, and Ball [8].

Input constants

- m = number of vehicles
- n = number of customers, indexed 1 to N, 0 is depot
- K = capacity of vehicle
- d_i = demand of customer i
- c_{ij} = cost of traversing edge (i,j)

Variables

- x_{ij}^k = 1, if vehicle k travels directly from customer i to customer j; 0, otherwise

Integer Linear Program for CVRP:

$$\min \sum_{k=1}^{m} \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} \cdot x_{ij}^{k} \tag{2.1}$$

Subject to:

$$\sum_{k=1}^{m} \sum_{i=1}^{n} x_{ij}^{k} = 1, \text{ for } j \in \{1, \ldots, n\} \tag{2.2}$$

$$\sum_{i=1}^{n} x_{it}^{k} - \sum_{j=1}^{n} x_{tj}^{k} = 0, \text{ for } k \in \{1, \ldots, m\}, t \in \{1, \ldots, n\} \tag{2.3}$$

$$\sum_{i=1}^{n} d_i \cdot \sum_{j=1}^{n} x_{ij}^{k} \leq K, \text{ for } j \in \{1, \ldots, m\} \tag{2.4}$$

Condition 2.1 signifies, that the total distance is to be minimized. With condition 2.2 we ensure, that each customer is served by exactly one vehicle. Condition 2.3 represents route continuity. If a vehicle node visits customer, it must also leave him. The capacity constraint is satisfied by condition 2.4.

2.2 Literature

The CVRP literature is very rich. It is very difficult to sort all the published articles and it would take plenty of pages to touch only the surface (google scholar shows around 548 000 articles on VRP and 16 100 results covering CVRP). We summarize the most important algorithms and we forward the reader to other sources for more details.

There is a book published on VRP problem (Golden, Raghavan, and Wasil [21]), which also includes chapters on CVRP. It focuses on the last advances, new VRP variants and approaches. It provides a unified presentation of research results from the year 2000 till 2008.

In 2014, the first book has been replaced by a second edition (Toth and Vigo [35]), whose text is either completely new or significantly revised. We have not been able to obtain and study this book and we only mention it, because from it is abstract it seems like a complete state-of-the-art coverage of VRP, which is very rare or impossible to find otherwise.

2.3 Benchmarks

There are many benchmark instances for the problem, most of them having up to 200 customers. For the purpose of testing we have chosen 3 different benchmark sets. The summary and other benchmark sets can be found in Uchoa, Pecin, Pessoa, Poggi, Subramanian, and Vidal [36].

Christofides, Mingozzi and Toth (CMT) (1979) [13]

This set contains 14 instances with a range of customers between 50 and 199. They have random structure and in instances 11 to 14 the customers are clustered in groups. Some of the instances also have a constraint on maximum length of a route. All the instances in this set has been solved to optimum by very sophisticated solvers and most of modern meta-heuristic approaches can solve most of them to optimal value. We have chosen this set, because it has quite small number of customers and some of open-source solvers have been tested on it too. With a little playing with the arguments of our algorithms we were able to solve all instances to optimum.

Li et al. (2005) [27]

This set contains 12 instances with a range of customers between 560 and 1200. It includes 3 largest instances ever published in research papers: 1040, 1120 and 1200 customers. This set has been very popular, because of its size. However the set is extremely artificial and all the customers are located in concentric circles around the depot. The routes in best known solutions have almost identical shapes, or the solution is somehow symmetrical. The optimum has not been proved in any of them yet. From all 3 benchmark sets our algorithms had most problems with this one and has on average differ less than 6% from the best known value. We think that the big gap from the best known solution is mainly because of the artificial structure and positioning of the customers. Our algorithms are based on local improvements, which are difficult to make if all neighbours of a customer have similar distances to that customer.

Uchoa et al. (2014) [36]

This is a recently published set, which consists of 100 instances with a range of customers between 100 and 1000. All the customers are generated in random clusters. This new set was designed in order to provide a more comprehensive and balanced experimental setting and is intended to be primarily used in the next years. If we compare our algorithm to the state-of-the-art algorithms compared in Uchoa, Pecin, Pessoa, Poggi, Subramanian, and Vidal [36], it appears, that in some instances we can compete with them and with bigger problem size our running time is much less. More closely we will look on this in the comparison section.

The VRP community tends to compete in providing the best-known solution for each instance. CVRPLIB [5] is a good resource, which have the input data for all popular benchmarks from the literature accompanied with their best-known solutions. They also organize a challenge for exact methods offering 300–500 US dollars as a prize. There exists many research groups and time from time there is a new challenge, for example [5].

2.4 Example

Input is specified in input file with TSPLIB extension ([4]). Customers are points allocated in a 2-D coordination system.

Output is both graphical and textual. Textual output contains a route number and a permutation of costumers, which are in that route in the order of traversing the route. At the end is specified total cost of the solution.

Graphical output of a solution consists of routes, each having a different colour. Customers are illustrated as points in 2-D coordination system. Figures 2.1–2.3 show a graphical output for benchmark instances described in 2.3. Figure 2.1 shows an optimal solution consisting of 5 routes to a problem with 50 customers.

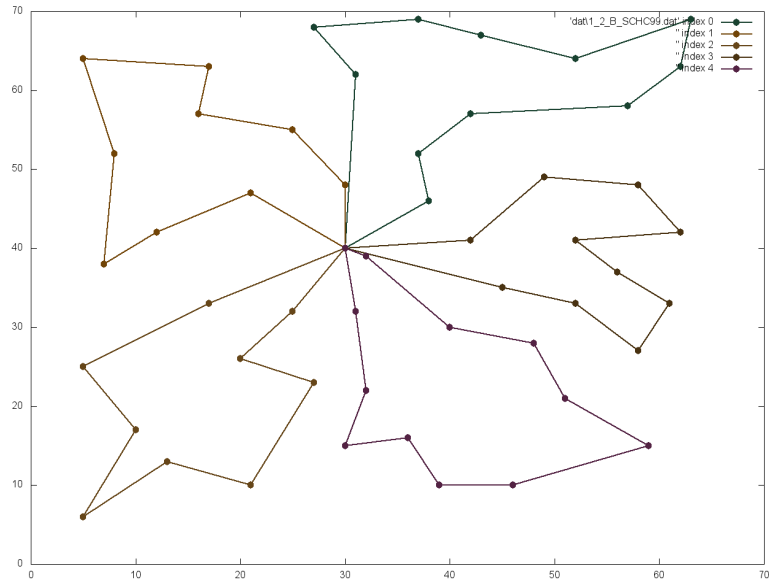


Figure 2.1: CMT1 optimal solution

Figure 2.2 shows a best-known solution consisting of 11 routes to a problem with 100 customers, which are being clustered. In this problem, there was also a restriction on a maximum route duration. The optimality of this solution has not been proved.

Figure 2.3 shows a complex solution consisting of 43 routes to a problem with 1000 customers. The gap between this and the best known solution is only 1,63%.

2.5 Algorithms and strategies

In the beginning of algorithm development for CVRP, researchers started with exact algorithms and some basic heuristics, however later it showed, that for

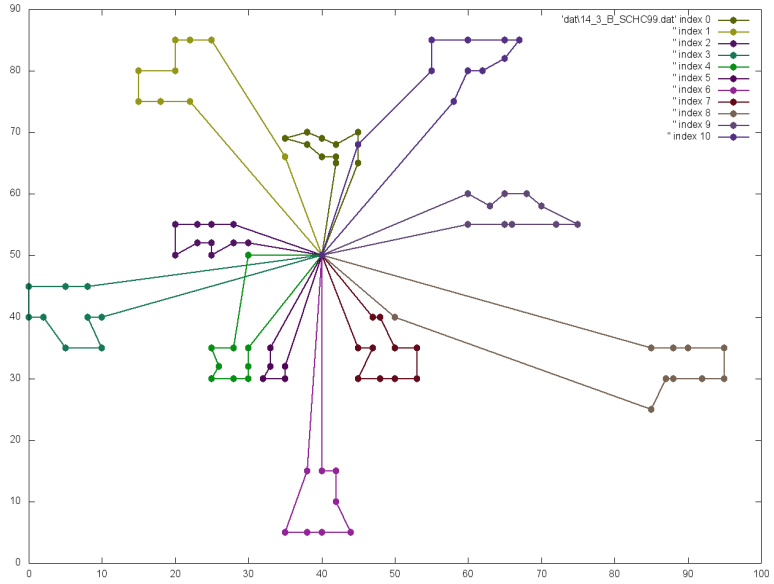


Figure 2.2: CMT14 solution

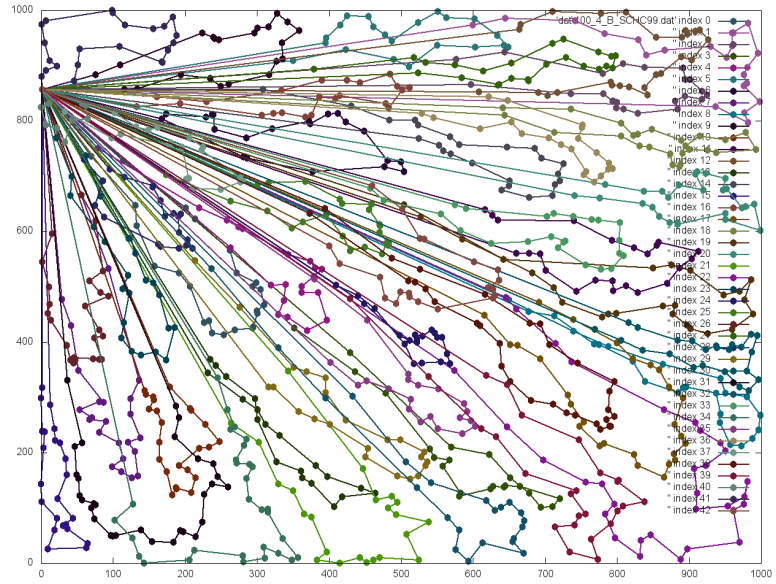


Figure 2.3: Uchoa-X-n1001-k43 solution

the real-life problems the exact algorithms are usually slow and the basic heuristic do not give a very good solution. So the research went in direction of meta-heuristics. Firstly it concentrated on local search kind of strategies, but recently more and more research is in population search algorithms. There has also been some improvement in exact approaches, but they still can not and will not be able to cope in near future with large data-sets. We provide a brief information about all the methods used for solving CVRP and later an in-depth description of methods and strategies we have used in the thesis.

2.5.1 Exact algorithms

Because of the NP-hard nature of the problem (Lenstra and Kan [26]), exact algorithms can not solve large instances. They can solve instances from the literature with up to 135 customers (Fukasawa, Longo, Lysgaard, de Aragão, Reis, Uchoa, and Werneck [18]). They are very complex and solved with specialized solvers using mixed-integer linear model formulation. We have not implemented these methods, but we mention them for completeness. The two common strategies are:

- Branch and bound: state-space search. Let set of solutions form a tree. The algorithm explores this tree and in every branch checks the lower and upper estimated bounds. If it is not satisfied, the solution is discarded.
- Branch and cut: generalization of branch and bound where, after solving the LP relaxation, and having not been successful in pruning the node on the basis of the LP solution, we try to find a violated cut. If one or more violated cuts are found, they are added to the formulation and the LP is solved again. If none are found, we branch.

To our acquired knowledge from Uchoa, Pecin, Pessoa, Poggi, Subramanian, and Vidal [36], there has been recently published a Branch-Cut-and-Price algorithm (BCP) (Pecin, Pessoa, Poggi, and Uchoa [29]), which is very complex and can solve most instances up to 275 customers. The number of customers, that can solve also depends on Q and the length of the routes. The computational time can vary from minutes to hours.

2.5.2 Heuristics

These include classical heuristic methods which have been developed the earliest and which are recently used as a basic solutions for other algorithms. They search a limited search space, but their advantage is speed and in most cases simplicity. The typically produce good quality solutions, but not such good as meta-heuristics, because they easily get stuck in local minima. There exists two kinds – constructive and improvement heuristics. Constructive methods perform well in constructing a basic solution.

- Clarke and Wright saving algorithm: the most commonly used algorithm, which is very fast and performs quite well on CVRP. Full description and pseudo-code can be found in implemented strategies and algorithms section.

- Route first, cluster second: we first construct a giant Travelling Salesman Problem (TSP) tour and then we cut the tour into feasible vehicle routes. The cutting phase if performed by solving a shortest path problem on an acyclic graph. Having a graph consisting from the TSP path and adding feasible edges, feasible edges, where edge (i, j) is the cost of having in that route nodes in between.

- Cluster first, route second: we first cluster the customers so that they are close to each other and their weights does not exceed car is capacity. This is obtained by solving a Generalized Assignment Problem (GAP). Then each cluster is one route, so we solve it by TSP. There are more ways how to form clusters and an interesting one is called Sweep Algorithm, which forms the clusters by rotating a ray centered at the depot by using smallest angle from previous customer, till they exceed capacity of a vehicle.

Improvement heuristics use exchange and relocation operators in local search. They are also so-called neighbourhoods.

- Local search – Iteratively and locally improves objective function. It will be described more in the implemented algorithms and strategies section.

- Neighbourhoods – Neighbourhood is a solution, which can be obtained from the current solution applying a local operator (exchange, swap, etc,...). There exists numerous local operators and we later in the text provide a description and reasons, which we have used and why.

2.5.3 Meta-heuristics

The key-point of meta-heuristics is to explore deeper promising regions of a search space. They include mechanisms, which make sure, that they do not get stuck in local optima and if they do, they have a chance to get out of it. The quality of solutions is very high in a relatively short time, but they are more sensible to parameters setting. There has recently been plenty of new meta-heuristics and we name just few of them and will later provide an in-depth description of those we have implemented.

- Genetic algorithms: today they are receiving plenty of attention and their performance is comparable or better than local search based ones. They make a simultaneous multiple way search. Our focus was not on these and the reader can read more about these in Goldberg [20].

- Local search based: these include local search variants, which are able to get out of local minima and search intensively around promising solutions. We provide our implementation and comparison of some of these.

2.5.4 Choosing strategies

Strategies used in local search based algorithms to decide which neighbourhood should be picked as a new solution.

- First fit: picks the first neighbouring solution, that satisfies the constraints. The advantage is, that it usually finds a solution quickly. However, if there are not many feasible solutions it can end up searching through all of them. The disadvantage is, that it picks always the same neighbour.

- Best fit: tries all the possible neighbouring solutions and picks the best one. The advantage is, that it always picks the best neighbourhood. The disadvantage is, that it always picks the same neighbourhood and has to search all neighbourhoods, which in G costs at least $O(N^2)$ operations, where N is the number of customers.

- Random fit: randomly chooses the neighbourhood solution, that satisfies the constraints. The advantage is, that it adds randomization to the algorithm and so we have a bigger chance not to end up in the same local minimum. However, if we have little feasible neighbourhoods left, it can take more time to find the feasible one.

2.6 Implementation

We have implemented various heuristics and meta-heuristics algorithms. We tested them on many benchmark instances from which we choose three most distinct ones. We have also compared our implementation to free and open-source solvers. Two of our implemented algorithms (Late Acceptance Hill-Climbing algorithm and Step Counting Hill-Climbing algorithm) have been developed very recently and we have not found any articles relating them to the CVRP problem. We show that they have a big potential in solving VRP problems and obtaining a very good solution in a reasonable time.

2.6.1 Neighbourhood operations

Neighbourhood is a solution, which we can obtain by applying an operation to the current solution. We have implemented 4 neighbourhood operations, which are used in some of the following algorithms to improve the solution. With each neighbourhood operation we provide a description and a reason why do we choose it. We first define a predecessor and successor of a node

in a route, before applying a neighbouring operation. Let i be a node from {1,...,N}. Having a route, which contains i and which permutation of nodes symbolizes the order of traversing these nodes from left to right, let the predecessor i_p be a node just before i and the successor i_s be a node just after i in that permutation.

- Swap procedure (Figure 2.4) takes two different nodes i, j from {1,...,N} and swaps these nodes. Having a sub-route (i_p, i, i_s) and sub-route (j_p, j, j_s), we swap the nodes i, j, so that we get sub-routes (i_p, j, i_s) and (j_p, i, j_s). This operation is the easiest and it is used in many problems, which are solved using local search. In CVRP it allows us to get rid of between-route x-crossings as we can see in figure below. It is both in and between-route operation.

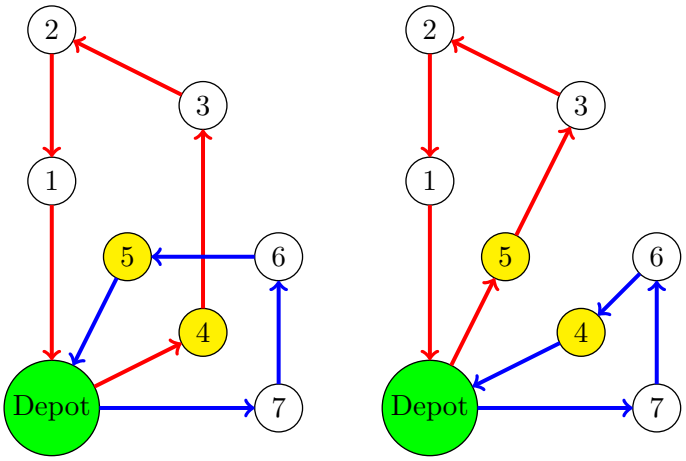


Figure 2.4: Before and after the procedure Swap(4,5)

- Reverse edge procedure (Figure 2.5) takes two different nodes i, j from $\{1, \ldots, N\}$, which have a common route and reverses the sub-route from i to j including both i and j. Having a sub-route $(i_p, i, i_s, \ldots j_p, j, j_s)$ we reverse the sub-route and obtain $(i_p, j, j_p, \ldots i_s, i, j_s)$ (if the number of nodes in the sub-route are ≤ 3, we apply the swap procedure with parameters i, j). In heuristic solutions we usually see x-crossings, which can be fixed by applying this operation without violating capacity constraints. It is an in-route operation.

- Delete and insert procedure (Figure 2.6) takes a node i from $\{1, \ldots, N\}$, deletes it and inserts it in the best possible place among the routes. Let $C(S)$ be the cost of the solution after deleting node i (sub-route (i_p, i, i_s) changes to sub-route (i_p, i_s)). Let cost $C(S^*)$ be the cost after inserting i between nodes j, j_s, for $j \in 0, \ldots, N$. We insert i between j, j_s, so that $C(S^*)$ is minimal. It allows us to change the number of customers

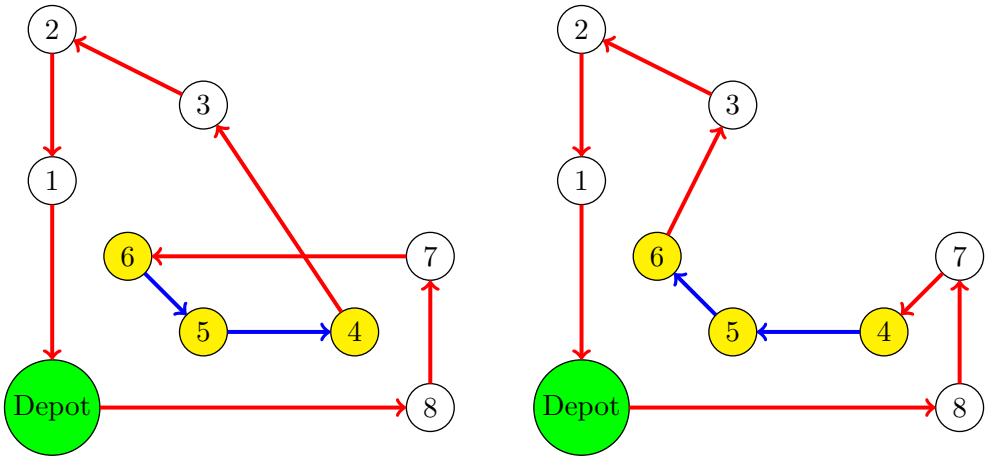


Figure 2.5: Before and after the procedure ReverseEdge(4,5)

in routes and decrease the number of routes. It is both in-route and between-route operation.

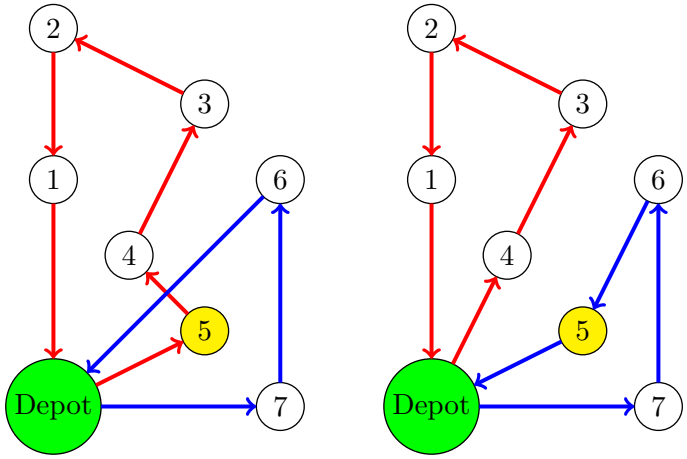


Figure 2.6: Before and after the procedure DeleteAndInsert(5)

- Double delete and double insert procedure takes two different nodes i, j from $\{1, \ldots, N\}$, which are then deleted at once and then reinserted at the best place in the opposite routes (the same way as in the "Delete and insert procedure"). This operation is useful if we have very tight capacity constraints and can not use single delete and insert operation. It seems very similar to swap operation, but it allows us to both restructure the number of customers in routes and to swap and insert at best possible place in one operation. Let $C(S)$ be the cost of current solution and let $C(S_n)$ be the cost of the solution S_n after applying this operation. We can also get to solution S_n through solutions $S_0, \ldots, S_n$ by applying

swap operations. However, it can happen, that S_i, for $i \in \{0, \ldots, n\}$, will end up in local minima and we will not be able to reach S_n.

2.6.2 Closest search

Closest search is an important, but not well studied strategy used in local search algorithms. It searches candidates for possible neighbourhood operations only in predefined closest distance. In our tests we have used a predefined value $20 \cdot \log(n)$. We have also tested our algorithms by picking the neighbour using the closest search in 50, 75 and 99 percent cases. In the result section, we show, that using the closest search in 99 percent cases is the best choice.

2.6.3 Savings algorithm

Savings algorithm published in 1964 (Clarke and Wright [14]) is found in many algorithms as a way to compute basic solution. There has also been few parameterized variants, which are summarized in Corominas, Garcıa-Villoria, and Pastor [15]. We have implemented a version with lambda parameter ($d[i][j]$ on line 5 of Algorithm 1 would change into $lambda \cdot d[i][j]$, where $lambda$ is a random real number in from the interval (0,2)), but it has not shown any improvement our meta-heuristic algorithms. It only helps to obtain different basic solutions. However, as most of our meta-heuristics have a random element build-in, we do not need to obtain for them a random basic solution.

We first illustrate the basic savings concept (Figure 2.7), that we can obtain saving if we connect two routes R_1, R_2 into one route R_3. Let $C(R_1)$ be the cost of R_1 and $C(R_2)$ be the cost of R_2. The cost of R_3 is calculated as follows: $R_1 + R_2 - c_{i,0} - c_{0,j} + c_{i,j}$, where $c_{a,b}$ is the cost of traversing $e_{a,b}$. The saving cost is only the difference between the current solution and the solution after applying the operation. It is calculated on line 5. After we have

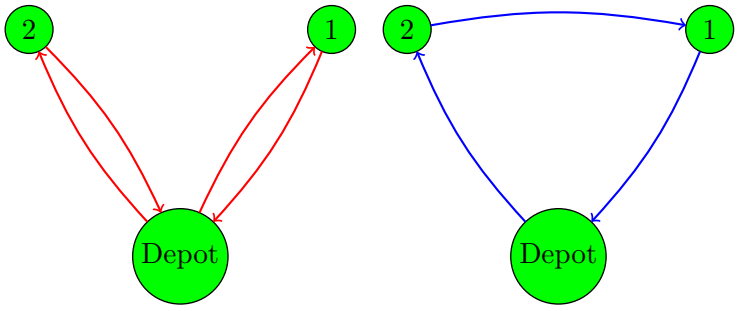


Figure 2.7: Savings concept

calculated all possible combinations of savings $V_{i,j}$, for $i, j \in \{1, \ldots, K\}$, we sort them in descending order and starting from the beginning we always get one $V_{i,j}$ with two customers i, j. If they are not in the same route, we decide if

Algorithm 1 Savings algorithm

```
1:  procedure algorithmSavings
2:      Initialize each customer in his own route
3:      for i = 1, ..., K do
4:          for j = 1, ..., K do
5:              s[it].cost ← d[i][0] + d[0][j] − d[i][j]
6:              s[it].i ← i
7:              s[it].j ← j
8:              it ← it + 1
9:      Sort s in descending order
10:     for all x in s do
11:         a ← x.i
12:         b ← x.j
13:         if ConstraintsViolated (a,b) then
14:             continue
15:         if a.next = depot AND b.next = depot then
16:             reverseRoute (b)
17:         if a.next = depot AND b.prev = depot then
18:             connectRoutes (a,b)
```

we can connect them or not and how to connect them. We can connect them only if they are at the beginning or end of a route (($i_p = 0$ or $i_s = 0$) and ($j_p = 0$ or $j_s = 0$)). There can only be two situations. Either both of them are at the end of the route ($i_p = 0$ and $j_p = 0$) or one of them is at the beginning and one at the end (($i_p = 0$ and $j_s = 0$) or ($j_p = 0$ and $i_s = 0$)). Because on lines 3–4 we have allowed savings $V_{i,j}$ and $V_{j,i}$, we now work only with one saving, where $j_p = 0$ and $i_s = 0$ (Algorithm 1, line 17). If $j_s = 0$ and $i_s = 0$, we first apply the reverse route procedure on the whole route, which contains node j. Finally if $j_p = 0$ and $i_s = 0$, we connect the routes by deleting $e_{0,j}$ and $e_{i,0}$, and by introducing $e_{i,j}$.

The algorithm proceeds with other pairs of nodes and is cutting number of routes, which are contained in final solution. In general, the number of routes in final solution is minimal, or very close to the minimum possible.

2.6.4 Hill-Climbing algorithm

We first describe general local search algorithm (LS) and then changes made to it. Every LS needs a basic solution S_b, which becomes current solution S. It then iteratively performs specified operation O and obtains a new solution S'. We use the neighbouring operations described earlier. We decide if the new solution is feasible and should be used or not (if yes, we replace S with S') and repeats the whole process. Every iteration we try to make a local change O to S.

The solution function have at least one global minimum and many local minima. Because of the nature of LS, we can not prove optimality of the result. LS can converge and stop in local minima. Hill-Climbing (HC) is a basic

Algorithm 2 Hill-Climbing algorithm

```
1:  procedure algorithmHillClimb
2:      S ← feasible solution
3:      C(S) ← cost(S)
4:      while 1 do
5:          S′ ← BestFeasibleNeighbourOf(S)
6:          C(S′) ← cost(S′)
7:          if C(S′) < C(S) then
8:              S ← S′
9:              C(S) ← C(S′)
10:         else
11:             break
```

algorithm based on local search. It always chooses feasible neighbourhood solution (based on choosing strategy – best, first or random fit) that improves the objective function and if it can not find one, it exits. Generally, we get a slight improvement of S_b, but as HC quickly converges to local minima and has no way to get out, the improvement is negligible to the improvement of following algorithms.

2.6.5 Late Acceptance Hill-Climbing algorithm

The Late Acceptance Hill-Climbing (LAHC) (Bykov [10], Burke and Bykov [9]) is a new meta-heuristic invented and presented by Yuri Bykov in 2008. HC accepts a new solution if it is better than the current one. LAHC accepts a new solution if it is not worse than a solution, which was current n iterations backward. The advantage of LAHC is, that it has only one initialization parameter pL, which we have to set-up based on the input data.

In the time of writing this thesis google scholar has not showed any results of applying LAHC on CVRP and showed only 118 results on applying LAHC to other problems. Just to compare, HC shows 66 100 results. The important element and only parameter is the length pL of the *lastCosts* array. It contains last pL accepted solution values $C(S_i)$, for $i \in \{0, \ldots, pL\}$. When we find a random feasible solution, which is not worse than $C(S_c)$, where $c = n \mod pL$, we make it a current solution and insert it into *lastCosts* at position $n \mod pL$. In our implementation we have chosen a time-stopping criteria of five-minutes and after specified number of iterations without improving solution we restart the algorithm by increasing all values in the array (parameters used in testing are specified in the result section).

Algorithm 3 Late Acceptance Hill-Climbing algorithm

```
1:  procedure algorithmLAHC
2:      S ← feasible solution
3:      C(S) ← cost (S)
4:      lastCosts[] ← C(S)
5:      n ← 0
6:      while Stopping criteria not met do
7:          S′ ← FeasibleNeighbourOf (S)
8:          C(S′) ← cost (S′)
9:          if C(S′) ≤ C(S) or C(S′) ≤ lastCosts[n mod pL] then
10:             S ← S′
11:             C(S) ← C(S′)
12:         lastCosts[n mod pL] ← C(S)
13:         n ← n + 1
```

LAHC can also accept worsening solutions, so it does not get stuck in local minima so often and when it does, the combination of the accepting criteria and restarts lets us most of the time escape the local minima and explore other regions of the search space.

The performance depends on pL of $lastCosts$. If $pL = 1$, LAHC degrades to HC. With bigger pL LAHC explores more of the search space, takes more time, has a better chance finding a better solution (bigger chance not to get stuck in local minima, but converges slower).

2.6.6 Step Counting Hill-Climbing algorithm

Meta-heuristic presented in 2013 (Bykov [11]), which is simpler than LAHC. Our implemented variant outperformed LAHC and it seems to be very reliable on different data-sets. Since year 2013, when it was presented, it has been applied only to few problems and we show, how good results in reasonable time it can give for CVRP. Google scholar mentions this algorithm only 18 times in total and never in relation with any variant of VRP. The algorithm has a parameter pL, which signifies how many iterations we allow a solution to be accepted according to an old solution cost, which is stored in B. In other words, a new neighbouring solution is accepted, if its cost is in a closed interval $[0, B]$, where B changes every n accepted iterations. This means, that we can accept a solution, that is worse than the actual solution, but it will never be worse than a solution, which was best before n iterations.

On lines 9–12, Algorithm 4 accepts random neighbouring solution, which is better or equal to the current solution, or it is better than the upper-bound B. If we reach the maximum of iterations allowed (line 13), we update B with the actual cost. So we allow the algorithm to search neighbouring solutions

Algorithm 4 Step Counting Hill-Climbing algorithm

```
1:  procedure ALGORITHMSCHC
2:      S ← feasible solution
3:      C(S) ← cost (S)
4:      B ← C(S)
5:      n ← 0
6:      while Stopping criteria not met do
7:          S′ ← FeasibleNeighbourOf (S)
8:          C(S′) ← cost (S′)
9:          if C(S′) ≤ C(S) or C(S′) < B then
10:             S ← S′
11:             C(S) ← C(S′)
12:             n ← n + 1
13:         if pL ≤ n then
14:             B ← C(S)
15:             n ← 0
```

in an interval, which is every pL steps more and more bounded by B, slowly converging to global minimum with a chance of escaping local minima.

Furthermore, we can improve the speed of the algorithm by dynamically updating the parameter pL. If $pL = 1$, SCHC becomes a HC algorithm as in the case of LAHC. We want this feature, when we have many possible neighbourhood solutions, which can quickly improve the cost function. However, when we get closer to global minimum, we have less feasible neighbourhood solutions, that improve the cost function and we could get into local minima, so we accept solutions, which are worse, than current one. This can speed-up the process, but it is difficult to decide, how to update pL.

We have also implemented restarts. It means, that if we get stuck at the same cost for a specified number of iterations, we add a value to B (number of restarts times a random number, which is modulated by an actual cost). See result section for more information on testing.

2.6.7 Simulated Annealing

Simulated annealing (SA) (Harmanani, Azar, Helal, and Keirouz [23]) is in optimization world a well-known method used to generate approximate solutions to large combinatorial problems and was first introduced by Kirkpatrick, Gelatt, and Vecchi [25].

SA begins with a basic solution and using the local search accepts the neighbourhood solutions if they are better than the current solution, otherwise it is accepted with a probability $p = e^{\frac{-\Delta C}{T}}$, where ΔC is a difference between actual and the neighbouring solution cost and T is a temperature. T is being

decreased by fT with each iteration and so does the p. Eventually T stops by reaching the temperature eT.

The algorithm is based on the idea, that if we set T high enough, we allow more room for searching the neighbouring solutions and as we converge to global minimum, the algorithm accepts more and more only the improving moves. If T is very low, the algorithm behaves as HC.

It is a very effective algorithm, if we set-up all three arguments (T, fT, eT) well. However, it is very difficult and not very efficient to always manually set-up and prune the parameters in solving real problems. We also implement a random inner-iteration loop, which tries to improve the solution for a specified number of iterations at the same temperature level (see the implementation for more details.

Algorithm 5 Simulated Annealing algorithm

```
1:  procedure algorithmSimulatedAnnealing
2:      S ← feasible solution
3:      C(S) ← cost (S)
4:      while eT ≤ T do
5:          S' ← FeasibleNeighbourOf (S)
6:          C(S') ← cost (S')
7:          if C(S') ≤ C(S) OR random (0,1) ≤ exp ((C(S)−C(S'))/T) then
8:              S ← S'
9:              C(S) ← C(S')
10:         T ← T · fT
```

2.6.8 Tabu Search

Tabu Search (TS) (Gendreau, Hertz, and Laporte [19]) is a well-known local search algorithm implementing a tabu list *tabuList* of length n. *TabuList* records last n neighbourhood operations, that we can not use for a number of iterations T. We use three neighbourhood operations (swap, delete and insert, double-delete and double-insert). Let an operation converting $C(S)$ to $C(S')$ be X. We implement the tabu list by having two tables $tabuList_{i,j}$, $tabuListR_{i,j}$, for $i, j \in \{0, \ldots, N\}$. $TabuList_{i,j}$ signifies a forbidden operation of manipulating with node i and route j. $TabuListR_{i,j}$ signifies a forbidden operation of manipulating with both node i and j. We keep a global time *TabuTime*, which we increase every iteration. If we update a time in the tabu lists (Algorithm 6, line 8), we insert a sum of *TabuTime* and T. With this improvement we do not have to update all values every iteration, but we check if a move is tabu or not (line 5), we ask $tabuList[X] \leq TabuTime$. Please see the source code for a detailed implementation of this improvement.

Algorithm 6 Tabu Search algorithm

```
1: procedure ALGORITHMTABUSEARCH
2:     while Stopping criteria not met do
3:         S' ← BestFeasibleNeighbourOf(S)
4:         C(S') ← cost(S')
5:         if C(S') < C(S) AND tabuList[X] ≤ 0 then
6:             S ← S'
7:             C(S) ← C(S')
8:             tabuList[X] ← T + 1
9:         decrease all values of tabuList
```

2.7 Computational results

2.7.1 Computational model

We run the tests on Intel(R) Core(TM)2 Quad CPU Q6600 2.40GHz with 6GB RAM and Windows 7 Ultimate 64-bit operating system. Every instance is run 5 times for 5 minutes and for one benchmark set, we always run 4 separate programs in parallel. We run them in parallel, because otherwise it would take us around 850 hours of computational time. The results obtained by our algorithms can only be worse. Each of the programs is solving a quarter of all the instances on separate core. All algorithms were implemented in C++ language and compiled as a C++ project in Dev-C++ 4.9.9.2 by g++.

We measure the performance of algorithms by calculating the average gap A (Definition 1) from best-known solutions found in [2] and calculating the average of these A for the whole benchmark set B (Average performance $= \sum_{i=0}^{n} A_i$, where n is the number of instances in B).

Definition 1 *Let S be our average solution from 5 runs, S^* the best-known solution and $C(S), C(S^*)$ the cost of these solutions respectively. Then the average gap A is calculated as follows:*

$$A = \frac{C(S) - C(S^*)}{C(S^*)} \cdot 100$$

2.7.2 Comparison of implemented algorithms

We show the performance of our algorithms on the three previously described benchmark sets. Algorithms SA, LAHC, SCHC and TS have one or more parameters, which have to be set manually. The algorithms show best performance, if we experiment with the parameters. However, this is not possible for real-world applications and so for the purpose of testing, we use the same parameters for each benchmark set. We experimented with the parameters and you can find the ones used in testing in the testing file "*tests.cpp*" on the accompanying CD (functions *FinalChrist*, *FinalLi*, *FinalUchoa*).

We have allowed a maximum time of 5 minutes for one instance, but in many cases the algorithms finish in a matter of seconds. Generally with more customers, the running time is bigger. We have also excluded results for HC, because these are 10 or more times worse, than results of the other algorithms. With algorithms SA, LAHC, SCHC we also test 4 different versions. The versions differ by the percentage chance of using closest search strategy per each iteration (0%, 50%, 75%, 99%) instead of a random selection of customers in a neighbourhood.

In instances with many customers ($N > 500$), the results are not as close to the best known value as in the smaller instances. This is caused by stopping the algorithm due to the time-limit, before it finishes. We decided for the time-limit, because totally we have tested the algorithms on 136 instances and if we run each one 5 times for five minutes by each algorithm (we have totally 15 versions of our algorithms), it takes us at most 12 750 minutes (by running four programs in parallel).

By comparing the default version of algorithms and the versions of using closest search strategy, we show that the ones with closest search strategy perform better and that by using the strategy in 99% we get the best results. The results are shown in tables in accompanied $results v2.xlsx$ file. Due to this fact, in the following graphs we only show results for the version of algorithms with the 99% chance of using closest search strategy.

We have also implemented only basic variation of TS, which gives us a comparison of how efficient are our algorithms.

We now summarize the performance of algorithms on the three benchmarks.

Christofides, Mingozzi and Toth (CMT) (1979) [13]

All of the implemented algorithms get good results on this set with an average gap less than 1%, but SA perform more than 0.1% better than the others due to low number of customers in this set (Figure 2.8). Both algorithms LAHC and SCHC converge quickly and does not allow much time for exploring the state space. This can be adjusted by different setting of parameters. By experimenting with the parameters SCHC performed the best in finding most best-known solutions (SCHC found 7/14, SA found 4/14).

We show, that due to the similar positioning of customers in the instances, A gets bigger with bigger N (Figure 2.9). We show this only for SA, but for the other algorithms the result is similar (see the $xlsx$ file).

We conclude, that CMT set is not very good for measuring the overall performance of CVRP algorithms, because it includes only two kinds of customer positioning and these instances differ only by the number of customers. We use CMT only because it is widely used in literature.

Figure 2.8: Comparison of our algorithms on CMT instances

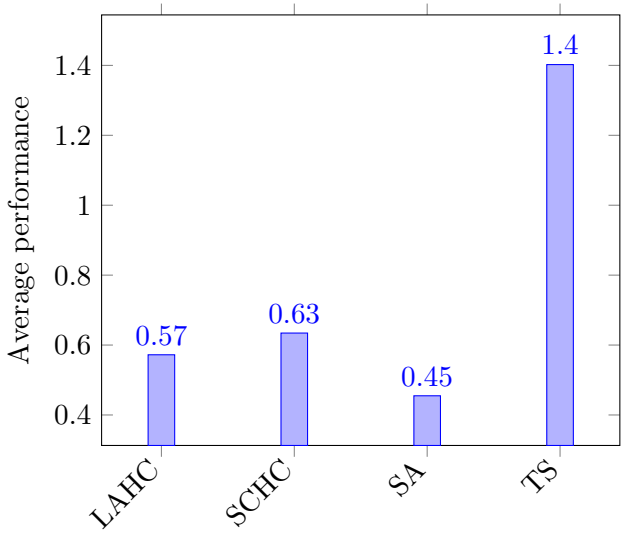
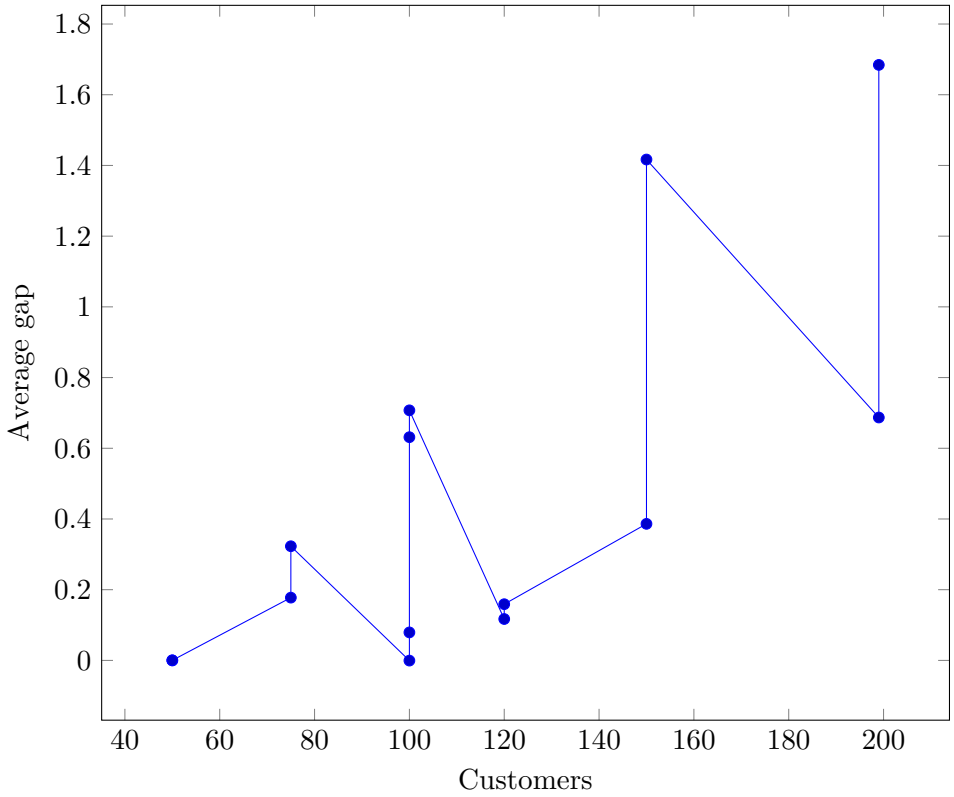


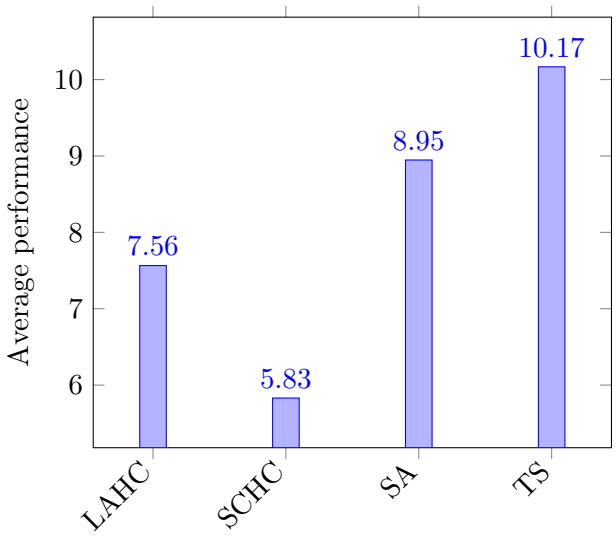
Figure 2.9: Dependency of SA on the number of customers on CMT instances



Li et al. (2005) [27]

Artificial large scale set of instances. All of our algorithms had a big gap (Figure 2.10), which is due to the artificial positioning. We see that in the even larger set of Uchoa et al, where is not such a big gap. Best results computed SCHC with the *Averageperformance* of 5.83.

Figure 2.10: Comparison of our algorithms on Li et al instances



In Figure 2.11 we see similar behaviour as in Figure 2.9. This is due to the artificiality of this benchmark set.

Uchoa et al. (2014) [36]

We believe that this set will be mostly used in future research, because it simulates real-life and random positioning of customers.

SCHC has obtained the best results (Figure 2.12) and proved to be very competitive to the commonly used SA.

We also show, that with a random positioning of customers, there is no more the dependency of A on N (Figure 2.13).

2.7.3 Comparison with other solvers

It is very complicated to compare with other solvers, because they do not have any benchmark tests or they show only best results. In all cases we were not able to set-up the solver's parameters in a way that would give us these results. For the purpose of comparison we run the solvers with default setting on CMT and Uchoa et al instances. We use CMT, so that reader can easily compare to other research papers, but our main focus is on the new set.

Figure 2.11: Dependency of SCHC on the number of customers on Li et al instances

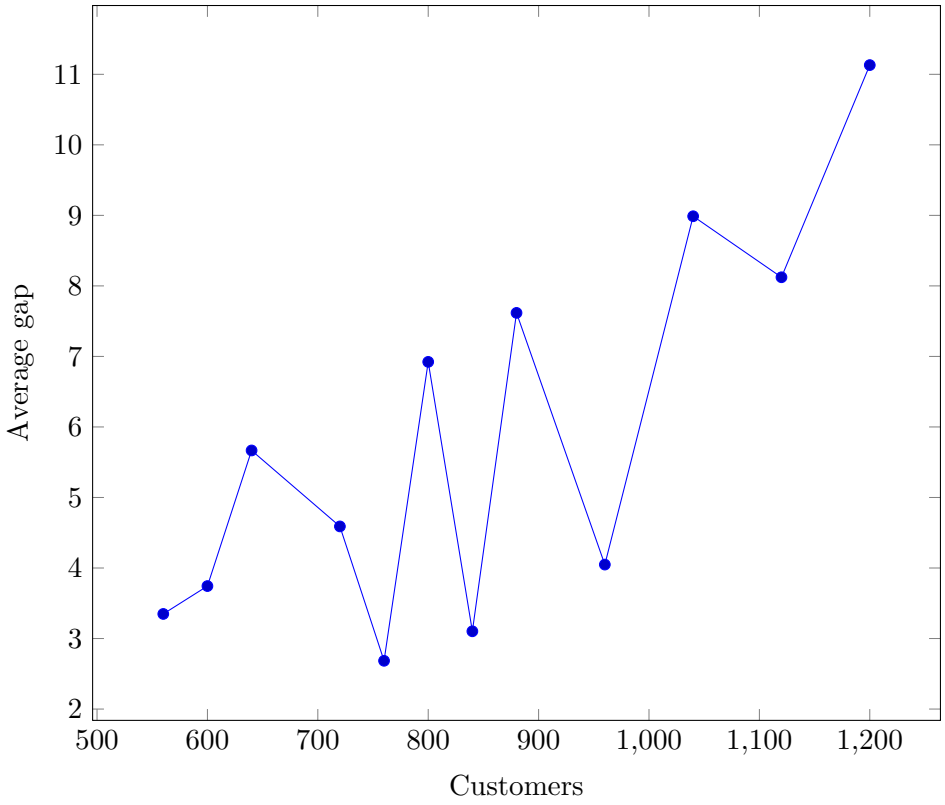


Figure 2.12: Comparison of our algorithms on Uchoa et al instances

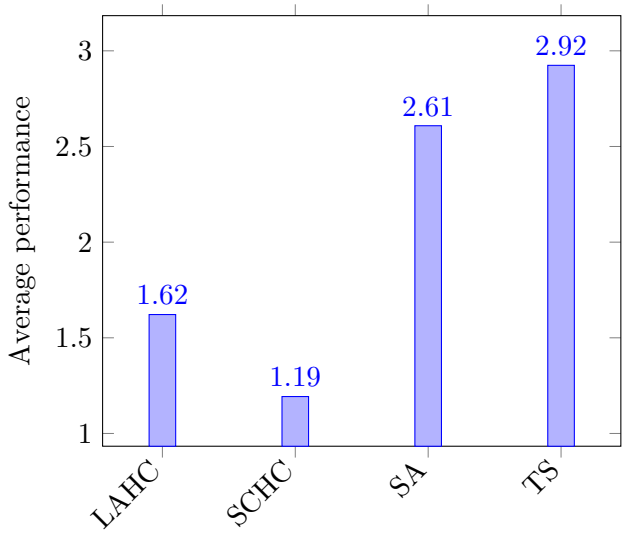
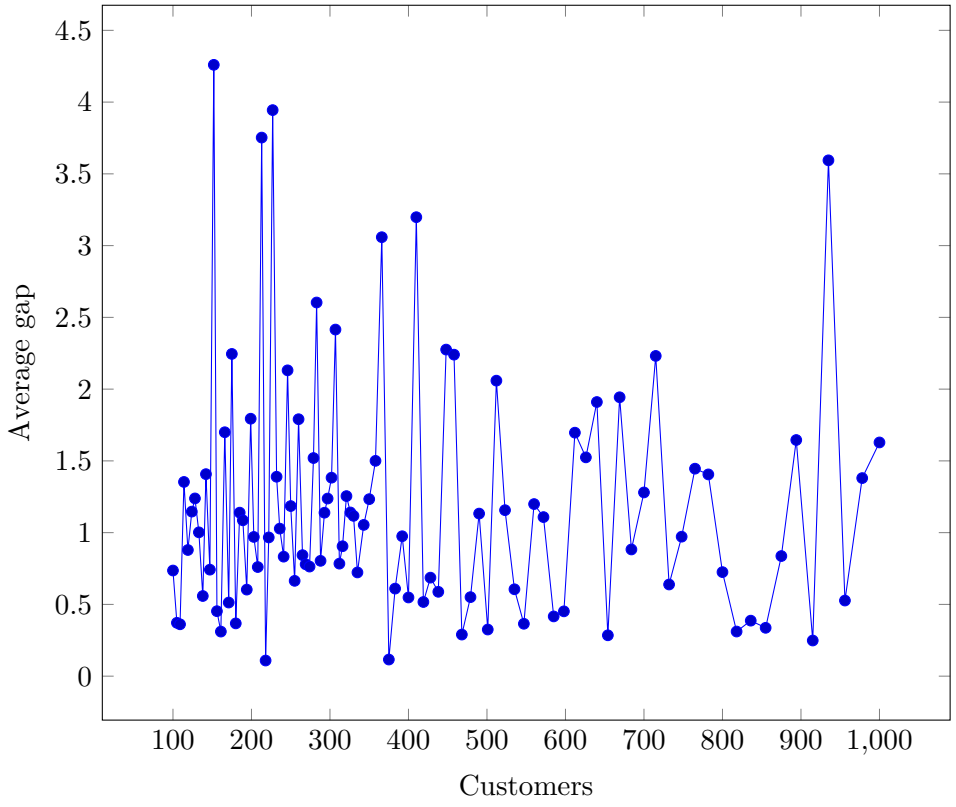


Figure 2.13: Dependency of SCHC on the number of customers on Uchoa et al instances



In Figure 2.14, we show the best results we were able to obtain from some of these solvers and compare them to our best results of SCHC for CMT bechmark. 2.14

In Figure 2.15, we show that our implementation for CVRP problem gives better results in average case, than best obtained result from the other solvers.

In Figure 2.16, we also compare the average results to the results obtained from the best scoring solver VRPH on Uchoa et al benchmark instances and show that even in this case of real-life problem our algorithm performs better. $LAHC - best$ and $SCHC - best$ show an $Averageperformance$ computed from gaps between a best-known solution and the best found solution in 5 runs.

We also compare SCHC with state-of-the-art algorithms presented in Uchoa, Pecin, Pessoa, Poggi, Subramanian, and Vidal [36]. These algorithms (ILS-SP, UHGS) are extremely complicated and their average running time per instance is 71.71 and 98.79 minutes respectively. Our average results differ less than 1% 2.17 from these two algorithms and we are able to compute each instance in less than 5 minutes.

Figure 2.14: Dependency of solvers on the number of customers on CMT instances

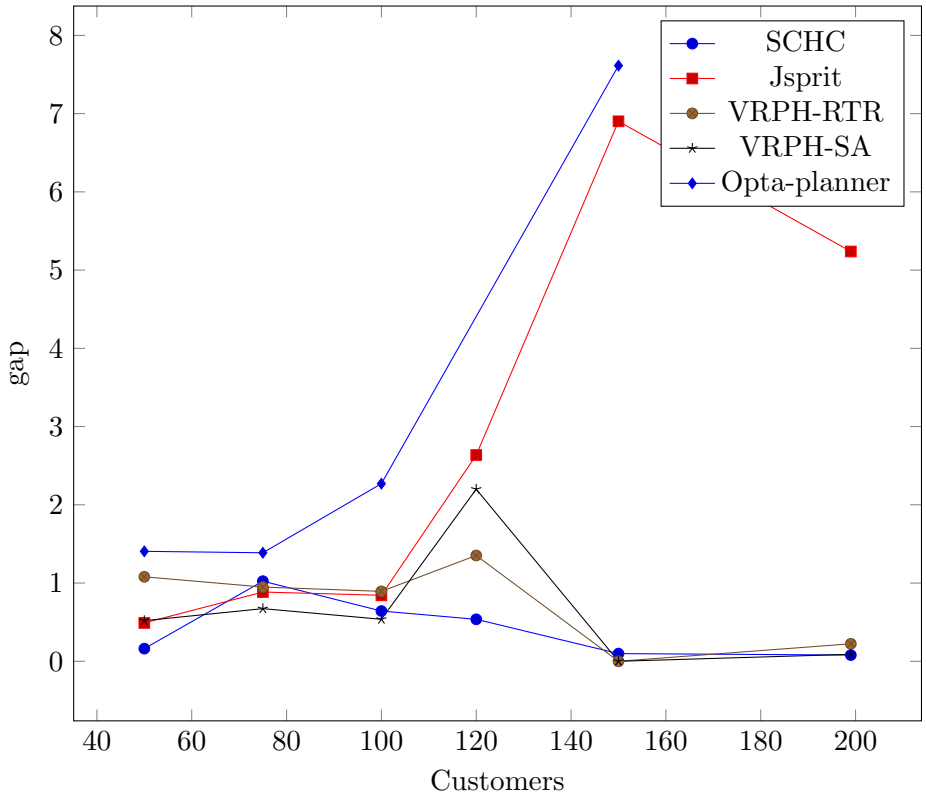


Figure 2.15: Comparison with other solvers on CMT instances

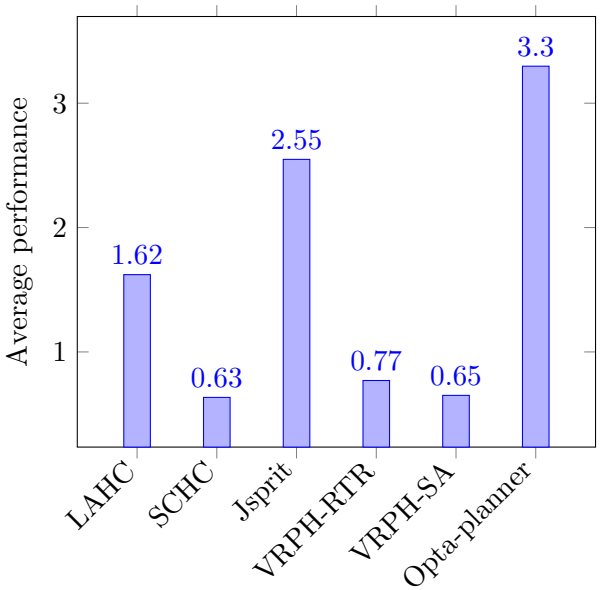


Figure 2.16: Comparison with other solvers on Uchoa et al instances

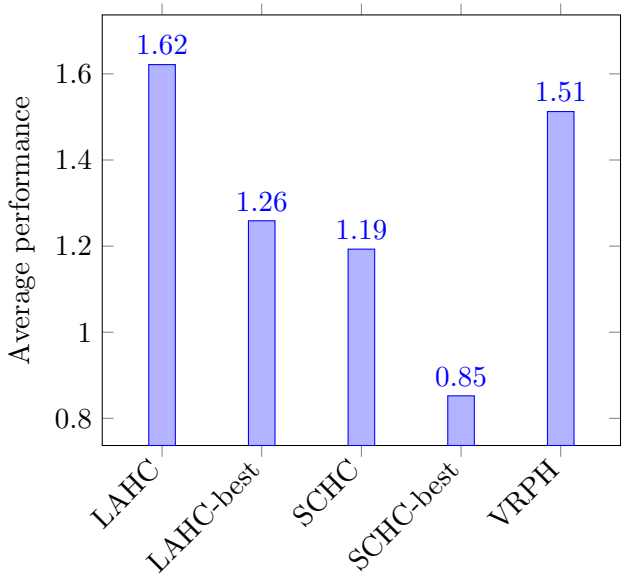
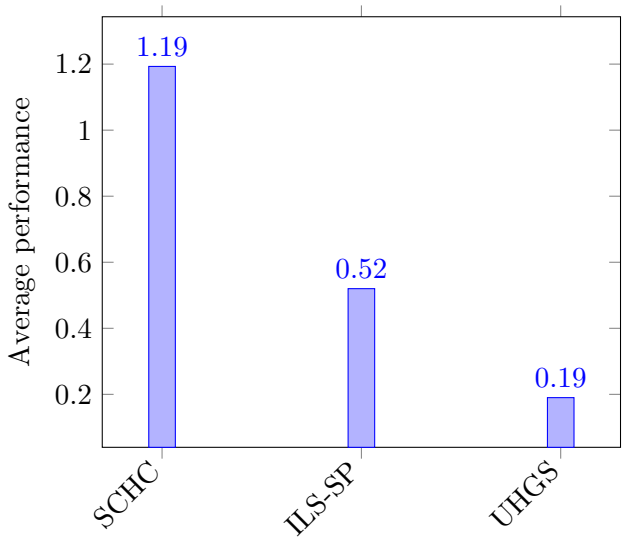


Figure 2.17: Comparison with state-of-the-art algorithms



2.8 Conclusion

We have shown, that both LAHC and SCHC perform well compared to other algorithms. We provide results for the first application of these algorithms to CVRP and compare these with other meta-heuristic algorithms, solvers and with state-of-the-art algorithms. We have shown the best performance between the solvers and a small average gap from the state-of-the-art algorithms.

Chapter 3

Other versions and further directions

During the process of researching and implementing algorithms for CVRP, we came across other very interesting variants of VRP. One of these is Cumulative Capacitated Vehicle Routing Problem (CCVRP). We provide a review of this problem and present few ideas for further research.

3.1 Cumulative Capacitated Vehicle Routing Problem

3.1.1 Introduction

The Cumulative Capacitated Vehicle Routing Problem (CCVRP) is a recently formulated problem, which is defined as CVRP with two differences. The first difference is, that in CCVRP the number of vehicles m is contained in the input and is set to a fixed value. The second difference involves the objective function. In CCVRP we want to determine a set of routes with the minimum of the sum of arrival times at customers. Let t_i^k be the arrival time of vehicle k at customer i. Then the objective function of CCVRP can be formulated as follows:

$$\min \sum_{k=1}^{m} \sum_{i=1}^{n} t_i^k$$

If the number of customers $N \leq m$, then the optimal solution is to have $N - 1$ routes, each starting at depot 0 and visiting exactly one customer.

The CCVRP generalizes the Traveling Repairman Problem (TRP) studied in Afrati, Cosmadakis, Papadimitriou, Papageorgiou, and Papakonstantinou [7], by adding capacity constraints and a homogeneous vehicle fleet. The most important application in real-life is a disaster scenario, where we want to get

to the customers in smallest total time possible, while having limited number of vehicles.

3.1.2 Literature

There has been published only few research papers on this problem. The problem was first formulated in Ngueveu, Prins, and Calvo [28] and the authors provide us with a memetic algorithm (genetic algorithm). They also illustrate necessary pre-computations details, when applying neighbourhood operations. An iterated local search algorithm was developed by Chen, Dong, and Niu [12], which obtains some new best-known solutions. A two-phase meta-heuristic is proposed in Ke and Feng [24] with a summary results of different algorithms.

3.2 Further directions

- Apply the newly presented algorithms (LAHC, SCHC) to other VRP variants and especially to CCVRP, which has not been extensively studied.
- Implementation of function, that will automatically adjust the parameters of LAHC and SCHC based on previous computations and their results.
- Measure, whether an implementation of tabu list would help these two algorithms in their performance.

Bibliography

[1] Symphony. URL https://projects.coin-or.org/SYMPHONY.

[2] Cvrp library. URL http://vrp.atd-lab.inf.puc-rio.br/index.php/en/.

[3] Odl studio. URL http://www.opendoorlogistics.com/.

[4] Tsplib. URL http://neo.lcc.uma.es/vrp/wp-content/data/Doc.ps.

[5] Cvrplib. URL http://vrp.atd-lab.inf.puc-rio.br/index.php/en/.

[6] Open-vrp. URL https://github.com/mck-/Open-VRP.

[7] F. Afrati, S. Cosmadakis, C. H. Papadimitriou, G. Papageorgiou, and N. Papakonstantinou. The complexity of the traveling repairman problem. *Journal of Information Technology Theory and Application*, 20(1): 79–87, 1986.

[8] L. Bodin, B. Golden, A. Assad, and M. Ball. Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations Research*, 10(2):96–98, 1983.

[9] E. K. Burke and Y. Bykov. A late acceptance strategy in hill-climbing for exam timetabling problems. In *7th International Conference on the Practice and Theory of Automated Timetabling(PATAT2008)*, 2008.

[10] Yuri Bykov. Late acceptance hill-climbing algorithm (lahc), . URL http://www.cs.nott.ac.uk/~yxb/LAHC/.

[11] Yuri Bykov. Step counting hill-climbing algorithm (schc), . URL http://www.cs.nott.ac.uk/~yxb/SCHC/.

[12] Ping Chen, Xingye Dong, and Yanchao Niu. An iterated local search algorithm for the cumulative capacitated vehicle routing problem. *AISC*, 136:575–581, 2012.

[13] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. *Combinatorial Optimization*, 1:315–338, 1979.

[14] G. Clarke and J. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations research*, 12(4):568–581, 1964.

[15] Albert Corominas, Alberto Garcıa-Villoria, and Rafael Pastor. Improving parametric clarke and wright algorithms by means of iterative empirically adjusted greedy heuristics. *Statistics and Operations Research Transactions*, 38(1):3–12, 2014.

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

[17] Güneş Erdoğan. Vrp spreadsheet solver, 2013. URL `http://verolog.deis.unibo.it/vrp-spreadsheet-solver`.

[18] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106(3):491–511, 2005.

[19] Michel Gendreau, Alain Hertz, and Gilbert Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 40(10):276–1290, 1994.

[20] D. Goldberg. Genetic algorithms in search, optimization, and machine learning. 1989.

[21] L. Bruce Golden, S. Raghavan, and A. Edward Wasil, editors. *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008.

[22] Chris Groer. Vrph. URL `http://www.coin-or.org/projects/VRPH.xml`.

[23] H. Harmanani, D. Azar, N. Helal, and W. Keirouz. A simulated annealing algorithm for the capacitated vehicle routing problem. Proceedings of the ISCA 26th International Conference on Computers and Their Applications, 2011.

[24] Liangjun Ke and Zuren Feng. A two-phase metaheuristic for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, 40:633–638, 2013.

[25] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–680, 1983.

[26] J. K. Lenstra and A. H. G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

[27] F. Li, B. Golden, and E. Wasil. ery large-scale vehicle routing: new test problems, algorithms, and results. *Computers & Operations Research*, 32 (5):1165–1179, 2005.

[28] Sandra Ulrich Ngueveu, Christian Prins, and Roberto Wolfler Calvo. An effective memetic algorithm for the cumulative capacitated vehicle routing problem. *Computers & Operations Research*, 37:1877–1885, 2010.

[29] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa. Improved branch-cut-and-price for capacitated vehicle routing. In *Integer Programming and Combinatorial Optimization*, pages 393–403. Springer, 2014.

[30] Victor Pillac. Vroom. URL `http://victorpillac.com/vroom/`.

[31] Stefan Schröder. jsprit. URL `http://jsprit.github.io/`.

[32] Geoffrey De Smet. Optaplanner. URL `http://www.optaplanner.org/`.

[33] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[34] M. M. Solomon and J. Desrosiers. Time window constrained routing and scheduling problems. *Transportation Science*, 22(2):1–13, 1988.

[35] Paolo Toth and Daniele Vigo, editors. *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. 2014.

[36] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Anand Subramanian, and Thibaut Vidal. New benchmark instances for the capacitated vehicle routing problem. 2014.

Appendix A

Testing manual

Instructions to repeat the testing process with our algorithms:

- The Dev-C++ project is contained in BP directory.
- For graphical output please change the directory of gnuplot on line 7 of file plot.cpp.
- Compile and run the project with changes in main.cpp:

  TEST(number of runs, from instance, to instance)
- Complete test of each benchmark:

  FinalChrist(5, 0, 14); for all instances of CMT

  FinalLi(5, 0, 12); for all instances of Li et al

  FinalUchoa(5, 0, 100); for all instances of Uchoa et al
- During the testing of CMT we compiled 4 programs and run them in parallel:

  FinalChrist(5, 0, 3);

  FinalChrist(5, 3, 6);

  FinalChrist(5, 6, 10);

  FinalChrist(5, 10, 14);
- During the testing of Li et al we compiled 4 programs and run them in parallel:

  FinalLi(5, 0, 3);

  FinalLi(5, 3, 6);

  FinalLi(5, 6, 9);

  FinalLi(5, 9, 12);

- During the testing of Uchoa et al we compiled 4 programs and run them in parallel:

  FinalUchoa(5, 0, 25);

  FinalUchoa(5, 25, 50);

  FinalUchoa(5, 50, 75);

  FinalUchoa(5, 75, 100);

- The output is summarized in a table, where at the top is the name of algorithm and each instance has 3 rows (best, average, worst result) and a column for each algorithm (including those with 0%, 50%, 75% and 99% of using closest search).

Appendix B

Acronyms

VRP Vehicle routing problem

CVRP Capacitated vehicle routing problem

VRPTW Vehicle routing problem with time windows

CCVRP Cummulative capacitated vehicle routing problem

LAHC Late Acceptance Hill-Climbing algorithm

SCHC Step Counting Hill-Climbing algorithm

SA Simulated Annealing

TS Tabu Search

Appendix C

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of source codes
- `BP` ..... the Dev-C++ project
- `text` ..... the thesis text directory
  - `thesis.pdf` ..... the thesis text in PDF format
  - `thesis.tex` ..... the thesis text in TEX format
- `results` ..... the results directory
  - `resultsv2.xlsx` ..... the excel result sheet
- `Benchmarks` ..... the benchmarks directory
  - `Uchoa` ..... the Uchoa et al instances
  - `Li` ..... the Li et al instances
  - `ChristofidesEilon` ..... the CMT instances