Technical University of Denmark

Adaptive Large Neighbourhood Search with exact methods for VRPTW

Author:
Jonas Christensen
s093074

Main supervisor:
Stefan Røpke
Department of Management Engineering

August 1, 2014

PREFACE

This M.Sc. thesis has been prepared by Jonas Mark Christensen during the period from the 1st of March, 2014 to the 1st of August, 2014, and constitute 30 ECTS point.

I have been supervised by Professor MSO Stefan Røpke, DTU Management Engineering. This thesis is the final requirement to obtain the degree: Master of Science in Engineering.

Readers of this thesis are assumed to have some knowledge of Operations Research (OR).

ABSTRACT

This thesis considers the Vehicle Routing Problem with Time Windows, which consist of determining a set of feasible vehicle routes to deliver goods to a set of customers, at a minimum cost.
The solution method is an Adaptive Large Neighbourhood Search algorithm in which many matheuristic procedures are tested. The ones that give the best results are a set partitioning method and an exact Branch Cut & Price method on a reduced graph.
The final algorithm shows very good results, and solves the 100 customer Solomon instances with an average best gap of 0.01% using 110 seconds on average. The algorithm are also tested on the 200, 400 and 600 customer Gehring and Homberger instances, and shows promising results for the 200 customer instances, but the matheuristic approaches fails to find improvements within the timelimit for most of the 400 and 600 customer instances.

Keywords: Vehicle Routing Problem, Large Neighbourhood Search, Matheuristic, Set Partitioning Problem, Operations Research

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

1 INTRODUCTION

Transportation of goods is an important part of today's society. Large amount of money are being spend on transporting goods, and decreasing the cost a few percentage can lead to big savings in absolute terms. It is therefore obvious to cut the costs associated with transportation.

Several approaches can be taken to cut costs. One could either improve the infrastructure or change the equipment. Both of these approaches require huge investments and could take years before they become profitable. Instead one could look at Operations Research techniques, and try to optimise the transportation with respect to the resources available.

This thesis considers a so-called vehicle routing problem and tries to find a near optimal transportation plan for inland transportation of goods, where the total number of kilometres driven is minimised. A good transportation plan both decreases the cost, as well as being more economical friendly as transportation is responsible for a great part of the total CO_2 pollution.

The class of Vehicle Routing Problems (VRP) are well-known and complex combinatorial problems, which has received considerable attention the last 50 years. This is both due to the importance of the problems in the field of distribution and logistics, as well as them being hard, both in theory and in practice.

A standard VRP can be stated as follows. Given a set of n customers and a fleet of k identical vehicles belonging to a given depot. The problem is to design k minimum cost routes, such that each customer is served exactly once and the truck capacities are not exceeded. Furthermore each truck must start and end its route at the assigned depot. Figure 1 is an illustration of the optimal solution to a VRP with time windows. The black square is the depot, and the rest of the nodes are customers. Each unique color define a truck's route. The solution looks messy, but this is due to the time windows, that might not make it attainable that two customers next to each other are visited by the same truck. Some of the routes crosses over itself, which cannot happen in an optimal solution for a standard VRP, if the triangle inequality is satisfied.

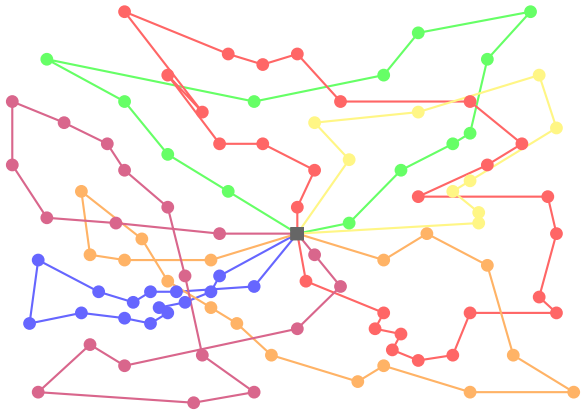


Figure 1.: The optimal solution to a VRP with time window

Many extensions of this standard problem exists, for example; Periodic VRP, Split Delivery VRP, Rich VRP, VRP with Time windows, VRP with Pickup and Delivery, VRP with Back-

haul, Multi-depot VRP, for a full overview of these as well as several other VRP variants see [15, 34].

This thesis considers the Vehicle Routing Problem with Time Windows (VRPTW), where each customer must be serviced within a given time-period. This problem has been addressed using many different techniques both including (meta)heuristics and exact methods. Solomon [31] proposed, in 1987, 56 instances with 100 customers that since then have been used to benchmark new algorithms. Up until recently, there were still some unsolved instance, but Ropke [28] solved the last one of them. However, even after 25 years of research the exact methods are still intractable in practice, as the computation time is too long for the methods to be used in real world applications.

Many heuristics have been proposed for the VRPTW. As discussed in the recent VRPTW survey by Desaulniers et al. [9] many new heuristics are Evolutionary Algorithms, but the Adaptive Large Neighbourhood Search algorithm presented by Pisinger and Ropke in [23] still remains competitive.

This thesis describes and uses an Adaptive Large Neighbourhood search algorithm to solve the VRPTW. The algorithms uses 5 different destroy methods, 2 insertion methods, a Simulated Annealing acceptance criteria and a resetting mechanism. Furthermore a local search method is embedded into the ALNS framework to further improve promising solutions.

The standard ALNS algorithm is extended by implementing and testing different exact submethods within the ALNS framework. The resulting algorithm is thus a matheuristic.

Matheuristics (or *model based metaheuritics*) is a relative new buzzword in the field of Operations Research (OR), however the ideas have existed for a long time. In OR there basically exists two streams of research, exact methods and heuristic methods. Exact methods guarantee, in theory, to find the optimal solution, however the run time often increases dramatically with the instance size. Heuristic algorithms most often do not provide any guarantee on the solution quality, but usually finds good solutions in limited time. By combining the strengths of these two approaches allows one to overcome the difficulties that occur when the individual concepts are applied solely.

The literature on matheuristics for the Vehicle Routing Problems in general is still quite small and primarily revolves around the same main idea. This thesis describes and tests several new matheuristics approaches that can be used together with a highly advanced Adaptive Large Neighbourhood Search metaheuristic.

Many heuristics approaches for the VRPTW tries to minimise the number of vehicles used, and then searches for the minimum cost distribution plan using this number of vehicles. The approach described here only focuses on minimising the total travel distance, thus making it hard to do a fair comparison with most state-of-the-art metaheuristics, but making it easier to compare the quality of the solutions with optimal solutions.

The final algorithm shows very promising results for the Solomon test instances, solving all the instances with an average gap of 0.11%, using on average 110 seconds. Out of the 56 instances the presented algorithm finds an optimal solution for 49 (equivalent to 88%) of these instances.

The remainder of this thesis is organised as follows. Chapter 2 gives a formal mathematical description of the VRPTW problem. Chapter 3 describes the recent advances in matheuristics,

mainly for the Vehicle Routing Problems. Chapter 4 gives a thorough description of the standard ALNS algorithm. Chapter 5 describes the different matheuristic approaches developed. Chapter 6 tests different aspects of the algorithm and Chapter 7 show the final results. Lastly chapter 8 contains the final conclusion.

The basis for the work presented in chapter 4 were created in a former project at DTU in collaboration with K. Arentoft with Stefan Røpke as supervisor. However, several new aspects have been studied and tested, but the main algorithm more or less remains the same.

2 MATHEMATICAL MODEL

Throughout this thesis the Vehicle Routing Problem with Time Windows (VRPTW) is considered. In the problem you consider a depot, and a set of customers with a demand. The objective is to serve these demands at a minimum cost, to further complicate the problem each customer only accepts deliveries within a certain time interval.
In order to model the problem, define the following sets

- $V = \{0, 1, ..., n, n+1\}$
  The set of nodes, where node 0 and $n+1$ are the depot and $1, ..., n$ are the customers.
- $A = \{(i, j) | i, j \in V\}$
  The set of arcs
- $K = \{1, ..., k\}$
  The set of available vehicles.

Furthermore, define the following parameters

- c_{ij}: Cost of using arc (i, j)
- t_{ij}: Time for travelling along arc (i, j), includes service time of node i.
- a_i: The time from when node i accepts deliveries.
- b_i: The time when node i no longer accepts deliveries.
- Q: The capacity of the trucks.
- d_i: The demand for node i.

Lastly, define two sets of variables

$$x_{ij}^k = \begin{cases} 1 & \text{If arc } (i,j) \text{ is traversed by vehicle } k \\ 0 & \text{Otherwise.} \end{cases}$$

and a time variable, w_i^k. If node i is visited by vehicle k then w_i^k indicates when service starts at node i, otherwise w_i^k is undefined.

With this the problem can be modelled as seen in model (1).

$$\texttt{Min Z} = \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \tag{1a}$$

`Subject to:`

$$\sum_{k \in K} \sum_{j \in V} x_{ij}^k = 1 \qquad \forall i \in V \setminus \{0, n+1\} \tag{1b}$$

$$\sum_{j \in V} x_{0,j}^k = 1 \qquad \forall k \in K \tag{1c}$$

$$\sum_{i \in V} x_{i,n+1}^k = 1 \qquad \forall k \in K \tag{1d}$$

$$\sum_{j \in V} x_{ji}^k = \sum_{j \in V} x_{ij}^k \qquad \forall k \in K, i \in V \setminus \{0, n+1\} \tag{1e}$$

$$w_i^k + t_{ij} \leq w_j^k + (1 - x_{ij}^k) \cdot M \qquad \forall k \in K, (i,j) \in A \tag{1f}$$

$$a_i \leq w_i^k \leq b_i \qquad \forall k \in K, i \in V \tag{1g}$$

$$\sum_{i \in V} d_i \sum_{j \in V} x_{ij}^k \leq Q \qquad \forall k \in K \tag{1h}$$

$$x_{ij}^k \in \{0, 1\} \qquad \forall k \in K, (i,j) \in A \tag{1i}$$

$$w_i^k \geq 0 \qquad \forall k \in K, i \in V \tag{1j}$$

The objective function, (1a) minimises the cost of travelling. The first constraint, (1b), ensures that each customer is served exactly once. Constraint (1c) and (1d) makes sure that the trucks leaves and returns to the depot. (1e) makes sure that if a truck visits a customer, then it also leaves this customer. Constraint (1f) updates the w_i^k variable, and (1g) makes sure that the service takes place while the customer accepts deliveries, lastly (1h) makes sure that each truck only serves customers where the total demand do not exceed the truck capacity. (1i) and (1j) defines the variables and sets the proper bounds.

The classic Capacitated Vehicle Routing Problem (CVRP) is provably $\mathcal{NP}$-hard. The CVRP is a relaxation of VRPTW, thus VRPTW is $\mathcal{NP}$-hard. That and combined with the intractability of the exact methods justifies the need for a heuristic method.

3 LITERATURE REVIEW

This literature review mainly focuses on matheuristic, primarily with focus on Vehicle Routing Problems and VRPTW especially. For a recent survey of state-of-the-art heuristic and exact solution methods for VRPTW see [9], or [15, 34] for an overview of several other VRP variants. The articles considered in this review were selected by starting from a selected few articles. Then I looked at some of the relevant articles they referenced, and some articles that had cited one of these articles. This process continued until I had a good overview of the method applied and new articles did not contribute with anything new that had not already been covered by one of the articles I had already read.
The following section, 3.1, reviews some of the surveys within the field, and classifies the different approaches. Section 3.2 reviews the most common approach for VRP matheuristics, namely set partitioning approaches. Section 3.3 describes some other approaches, and lastly section 3.4 provides an overview of the described algorithms.

3.1 SURVEYS AND CLASSIFICATION

This section review three surveys. The first one is a general one for combinatorial optimisation problems, whereas the two last ones are specific for Vehicle Routing Problems.

Puchinger and Raidl surveys in [25] state-of-the-art approaches of combining exact algorithms and metaheuristics to solve combinatorial optimisation problems. Some of these hybrids aim at providing exact solutions in shorter time, whereas others focus on getting better heuristic solutions. They categorise these combinations according to the following two main categories:

- *Collaborative*
  Collaborative combinations are algorithms with methods that exchange information. However each method are not part of the other, and they can thus be executed individually.
- *Integrative*
  Puchinger and Raidl defines integrative combinations as algorithms where one technique is embedded into another technique, and thus there is a distinguished master algorithm.

Doerner and Schmid surveys in [10] recent matheuristics trends for the Rich Vehicle Routing Problems. They outline three main directions of hybridisation between exact algorithms and metaheuristics; set partitioning approaches, local branching and decomposition.

In the set partitioning (SP) approaches, a metaheuristic is used as a *generator* of routes. And a SP problem is solved in order to consider many good routes at the same time. This makes it possible to consider many local optimal solutions at the same time and combine them in an optimal manner.

The local branching approach is a general-purpose MIP heuristic in which cuts of the following type are used in the beginning,

$$\Delta(x, \bar{x}) \leq k$$

where $\bar{x}$ is the incumbent, x a proposed solution, $\Delta(\bullet, \bullet)$ the 'distance' between the two solutions (Hamming distance for binary variables), and k is a positive integer. Local branching favours early updating of the incumbent solution, thus producing high-quality solutions in the beginning of the computation. This general idea was first proposed by Fischetti and Lodi in [12].

Decomposition approaches splits the full problem into smaller sub problems that easily can be solved to optimality, or atleast near-optimality, with the hope that the solutions to the sub problems can be combined to a full feasible solution.

Archetti and Speranza [2] focuses on decomposition approaches, Improvement heuristics and column generation based approaches in their survey on matheuristics for routing problems, thus excluding local branching schemes in their survey. The reason being that only few algorithms for routing problems fall into this class. The column generation based and decomposition approaches coincides with the two other classes considered by Doerner and Schmid. Archetti and Speranza also considers a class of matheuristic they describe as improvement heuristics, these heuristics uses mathematical programming models to improve a solution found by another heuristic.

3.2 SET PARTITIONING BASED

The most used matheuristic approach used in Vehicle Routing Problems are set partitioning (SP) approaches. This section review some of the articles using this method, and tries to outline some of the differences between the different methods.

Rochat and Taillard [27] proposed in 1995 a general VRP heuristic that follows the SP approach and uses a tabu search algorithm as generator. The heuristic only applies the SP model as a postoptimisation step, the main focus is thus on the tabu search algorithm. The algorithm is tested on Capacitated Vehicle Routing Problems (CVRP) and VRPTW instances, and for the VRPTW it succeeds in finding 27 new best solutions for the 56 Solomon instances. However these solutions have since been improved.

Kelly and Xu [17] implements a simpler two-phase tabu search algorithm, where the first phase generates a set of routes. In the first phase they allow for infeasible full solution, as long as the routes themselves are feasible, that is they allow solutions where a customer is *over-covered* or not *covered* at all. In the second phase they set up a set partitioning problem that considers all of the generated routes. This SP model is solved by means of a tabu search algorithm with a simple swap neighbourhood. During the second phase they also allow infeasibilities, but ensure a feasible solutions by utilising two greedy *patching* procedures, and through this generates new routes. Therefore the final solution could be better than the one achieved by using a general-purpose MIP-solver, but on the other hand, it could also be worse as the tabu search do not ensure optimality of the SP problem. They consider the CVRP and distance constrained CVRP problem, and test the algorithm on the 14 instances by Christofides, Mingozzi and Toth presented in [20]. Out of these 14 instances they find the best known solution in 10 cases, and solves the rest with an average gap of 1.76%, compared to the then best known solutions.

Alvarenga et al. [1] tackles the VRPTW by means of a two-phase genetic algorithm. In the first phase many independent populations are made, where there are no influence or genetic material interchanged between each population. The routes from the best individual from each population is added to a set R. Before phase II a SP model is solved to optimum by considering

the set of routes in R. In the second phase several new reduced partial solutions, based on the solution to the SP model is generated. For these reduced problems the genetic algorithm is executed once and the routes from the best individual in each population are added to the set R. Thus the full problem is decomposed with the hope of finding a better solution to the reduced problems that improves the overall solution. In the end of phase II the full set of routes, R, is added to the global R_{Global} and all elements from R are deleted, such that R is empty. If the time limit is not reached this cycle starts over again. When the time limit is exceeded the routes in the set R_{Global} is used to solve a SP model. The algorithm is tested on the 56 Solomon test set, and a time limit of 60 min was used. Comparing with the 33 instances where the optimal solutions were known at the time, they find the optimal solution in 24 cases, and solves the 33 instances with an average gap of 0.29%. In the end they compare the performance of the described approached with and without using the SP model over the set R (between phase I and II), and concludes that using the SP model over the set R improves the solution by as much as 5%.

Pirkwieser and Raidl [22] describes two matheuristic for the periodic VRPTW. The first one, called VNS-ILP, is a Variable Neighbourhood Search algorithm that acts as the sole provider of columns for a set covering problem. The set covering model is called several times during the execution of VNS-ILP and feeds the variable neighbourhood search algorithm with a new best solution to continue from. The second algorithm, called CG-EA, is a hybrid between a Column Generation algorithm and an Evolutionary Algorithm. The pricing problem is an elementary shortest path problem with resource constraints (ESPPRC) which is solved heuristically by means of a tabu search algorithm and a greedy label correcting dynamic programming algorithm. The solution to the restricted master problem (RMP) is feeded to the Evolutionary algorithm (EA), such that the initial population in the EA are the columns where the corresponding variable in the RMP is non-zero. The selection criteria in the EA is then based on the corresponding LP-values, such that routes with a higher LP-value have a higher chance of being selected. This procedure initialises half of the initial population, and a variant of the VNS algorithm provides the last part of the full population. VNS-ILP is compared with a pure VNS algorithm, and CG-EA is compared with a column generation algorithm with the two heuristic approaches to the pricing problem as described above, and a pure Evolutionary Algorithm. Both matheuritistic yields significantly better results than their pure metaheuristic counterpart. However the VNS-ILP is significantly better than CG-EA, and even VNS outperforms CG-EA. The algorithms are tested on newly derived instances, and thus cannot be compared with any other heuristics than the ones presented in the paper.

Subramanian et al. proposes in [33] an algorithm for a class of Vehicle Routing Problems. Their algorithms do however not consider time windows. The algorithm uses the Iterated Local Search algorithm abbreviated ILS-RVND as described in [21, 32]. Furthermore a SP model is used to optimise over the set of generated routes. However, to reduce the set of considered routes in the SP model, they define a criteria determining when a route should be added to the pool of routes used in the SP model. When the SP model finds an incumbent solution better than the current best, then the ILS-RVND algorithm is called. If the ILS-RVND heuristic succeeds in finding a better solution the *cutoff*-value is updated for the SP-model. The algorithm is tested on the A, B, E, M, P series of tests set for the CVRP and CMT sets from [20]. For the A, B, E, M and P series all known optimal solutions were obtained, and for the three open M-problems the algorithm improved the best known for two out of 3 problems, and obtained the best known for the last problem. For the CMT-sets the average gap between the average solutions and the best known solutions were 0.08%, the same as the the algorithm by Rochat and Taillard from [27].

Yildirim and Çatay [36] implements a parallel ant colony optimisation algorithm where the solution to a SP model formulation is used to update the pheromones. In the ACO phase the Time-based Ant System (TbAS) as presented in [35] is used. The algorithm is tested on the R and RC instances of the Solomon instances. Comparing with the best known solution they report they improve 3 out of 39 best known solutions, and the average gap is 0.22%[1].

3.3 DECOMPOSITION AND COLUMN GENERATION BASED

This section considers some of the decomposition and column generation based approaches found in the literature.

Archetti et al. [3] describes an optimisation-based heuristic for the Split Delivery Vehicle Routing Problem (SDVRP). A tabu search heuristic is used to identify parts of the solution space with high quality solutions. This neighbourhood is then explored by means of a suitable integer programming model. The model handles the demand and capacity constraints of the problem, and uses routes found by the tabu search as the decision variables. Moreover they use information from the tabu search algorithm to generate the set C', which is the set of customers that are very unlikely to be served more than once in an optimal solution. This is used in the model to reduce the complexity and thus ensuring a faster execution time. In the end this matheuristic is compared with a pure tabu search heuristic, where both heuristics are given the same time limit. The results show that out of 42 instances the matheuristic gives the best results for 33 of them. They conclude "*Optimization techniques in conjunction with heuristic search can indeed lead to better solutions than focusing entirely on heuristic search*".

Franceschi et al. considers in [13] the Distance-Constrained CVRP. They described the so-called *Selection, Extraction, Recombination, and Reallocation* (SERR) method. In the *Selection* phase nodes are selected, which are deleted from the solution in the *Extraction* step. The *Recombination* part applies heuristic procedures to generate a set of new possible routes, and in the *Reallocation* step the best of these routes are selected by means of a set partitioning model with side constraints. The *Recombination* step is a two-phase method, where the first phase generates simple routes. The second phases is a pricing loop that uses the dual information that is available after having solved the LP relaxation of the SP model. The dual information is used to calculate the reduced cost of a route, which is used to determine whether or not the route should be added to the SP model. This limits the number of routes in the SP model, which makes it faster to solve. This method shares great similarities with the delete and reinsert idea that is known from Large Neighbourhood Search (LNS) algorithms. The 39 instances from the A, B, E an P datasets are considered for the CVRP. These 39 instances are solved with an average gap on 0.36%[2].

Prescott-Gagnon et al. [24] uses a two phase algorithm for the VRPTW, with a slightly changed objective function. First of all they try to minimise the number of vehicles used, in the second phase they then search for the minimum cost distribution plan using the minimum number of trucks. The second phase is a Large Neighbourhood Search algorithm where the neighbourhood is explored using a branch-and-price heuristic. The proposed column generation heuristic is an adaptation of the exact method used by Desaulniers et al. [8], where only the

1 Even though it is not explicitly stated I believe they use full precision, and thus cannot compare with the known optimal solutions as they are computed using truncated precision.

2 Two similar algorithms are presented, where the only difference is the initial solution (FJ and SWEEP), the gap is calculated on the basis of the best solution achieved by these two algorithms.

tabu search method is used to provide columns to the master problem. In order to ensure the integrality constrains a heuristic branching method is used, where decision are imposed on the route variables from the master problem. If a fractional solution is found after the linear relaxation is solved, the variable with the largest fractional value is simply fixed to 1. Furthermore, no backtracking is allowed in the branch-and-bound tree, and thus branching decisions cannot be undone to go up in the search tree. Doing so, the solution might be worse than the current solution, and even no feasible solution might exist. If the solution is worse than the current, the deteriorated solution is accepted to contribute to the diversification of the search. If no feasible solution is found the previous solution remains the current solution. The algorithm is tested on the Solomon instances and Gehring and Homberger's instances with 200, 400, 600, 800 and 1000 customers [14]. For the Solomon instances they compare their algorithm with the ALNS algorithm by Pisinger and Ropke presented in [23]. Considering all the 56 instances both algorithms uses the same number of vehicles, but the algorithm by Prescott-Gagnon et al. finds a distribution plan that is 0.16% better on average, however the ALNS algorithm is 12 times faster[3]. For Gehring and Homberger's (GH) instances with 200, 400, 600, 800 and 1000 customer they compare with the leading heuristics and are generally better than these. Furthermore the branch-and-price algorithm manage to find 106 new best solutions for the 300 GH instances.

3.4 OVERVIEW

Table 1 summarises the different presented matheuristics. The column 'Meta' describes which general metaheuristic framework is used, the column 'Method' describes which matheuristic ideas the algorithm is based on. SP means (generalised) set partitioning/set cover models, D is for decomposition and CG column generation. The next two columns contains information on which problem the algorithm focus on and which datasets it is tested on. The three last columns shortly summarises the performance of the algorithms. The column 'New best' describes how many new best solutions were found, comparing with the best known solution available the time the article was published. 'Avg. Gap%' is the average gap compared to the best known solutions reported in the paper. 'Current Gap %' is only applicable for the algorithms tested on the Solomon VRPTW instances, and is the average gap when comparing with the optimal solutions.

Table 1 cannot be used to compare the algorithms with each other, as they consider different datasets. Additionally the best known solutions are not necessarily the same, which makes the column 'Avg. Gap%' useless for comparison. It can however be used to give an overview of the different techniques applied and the individual performance of the algorithms.

Author	Year	Meta	Method	Problem	Instances	New Best	Avg. Gap%	Current Gap%
Rochat & Taillard	1995	TS	SP	VRPTW	Solomon	27/56	-1.19	5.63
Kelly & Xu	1999	TS	SP	(D)CVRP	CMT	0/14	0.51	-
Alvarenga et al.	2007	GA	SP/D	VRPTW	Solomon	0/56	-	1.57
Pirkwieser & Raidl (VNS-ILP)	2010	VNS	SP	PVRPTW	-	-	-	-
Pirkwieser & Raidl (CG-EA)	2010	EA	CG	PVRPTW	-	-	-	-
Subramanian et al.	2012	ILS	SP	(D)CVRP	CMT/M	0/14 / 2/5	0.08 / -	- / -
Yildirim & Çatay	2014	ACO	SP	VRPTW	R(C) Solomon	3/39	0.22	-
Archetti et al.	2008	TS	D	SDVRP	-	-	-	-
Franceschi et al.	2006	LNS	D/SP	(D)CVRP	A, B, E, P	5/39	0.36	-
Prescott-Gagnon et al.	2009	LNS	D/CG	VRPTW[4]	GH	106/300	-	-

Table 1.: Comparison of the different matheuristics.

3 Prescott-Gagnon et al. uses an OPT 2.3 GHz CPU whereas Pisinger and Ropke uses a P4 3 GHz CPU.
4 First of all minimising number of vehicles, and then the distance.

4 ADAPTIVE LARGE NEIGHBOURHOOD SEARCH

Adaptive Large Neighbourhood Search (ALNS) belongs to the class of heuristics known as Very Large Scale Neighbourhood search (VLSN) algorithms. These algorithms are based on the fact that searching a large neighbourhood increases the number of high quality solutions in the neighbourhood, and therefore VLSN algorithms may return better solutions. Searching a large neighbourhood exhaustively is time consuming, and therefore various techniques are used in order to limit the search.

Ropke and Pisinger [29] first proposed ALNS and it improved the best known solution for over 50% of the 350 benchmark instances for the Pickup and Delivery Problem with Time Windows. This algorithm was modified to be used for the VRPTW and Pisinger and Ropke describes in [23] an ALNS algorithms that solves all the 56 Solomon instances with an average gap of only 0.36%[1], with 47s as the average time spent. Furthermore, for 27 out of the 56 instances the algorithm found an optimal solution.

ALNS is an extension of Large Neighbourhood Search (LNS). In LNS you start with a feasible solution and part of this solution is destroyed in each iteration, the solution is then repaired by some repair heuristic, hence we are only sampling the neighbourhood and thus limiting the search. LNS algorithms often only uses one destroy and repair method. ALNS allows for multiple destroy and repair methods and adaptively changes the probability of choosing one of these methods such that the one that performs best is more likely to be chosen.
This adaptive layer is handled by assigning weights to the different heuristics and use a roulette wheel selection principle. Consider k heuristics, and let $w_i, i \in \{1, 2, \ldots, k\}$ denote the weight of heuristic i. Then heuristic i is selected with probability,

$$P_i = \frac{w_i}{\sum_{j=1}^{k} w_j}$$

The basic idea is to adjust these weights accordingly to the performance of the heuristic. The search is divided into a number of segments, and the weights are adjusted in the beginning of each segment. A segment is defined as a certain number of iterations, η.
Define π_i as the score of heuristic i in the previous segment, and θ_i as the number of times heuristic i have been used in the previous segment. The score of a heuristic, π_i, is updated according to the scores, σ, described in table 2.
σ_1 makes sure a heuristic providing a new global best solution is rewarded, such that the probability is increased in the next segment. Similar a heuristics score is increased by σ_2 if a solution is accepted by a specific acceptance criteria as it has improved the current solution and thus moved to a new part of the solution space. The parameter σ_3 awards heuristics that can diversify the search. Determining whether it was the destroy or the repair heuristic that provided the improvement is hard, therefore both heuristics' scores are updated.

1 When comparing with the then best known solutions. Comparing with the optimal solutions the average gap is 0.44%

Parameter	Description
σ_1	The remove-insert operation resulted in a new global best solution.
σ_2	The remove-insert operation resulted in a solution better than the current solution, and it has not been accepted before.
σ_3	The remove-insert operation resulted in a solution worse than the cost of the current solution, and it has not been accepted before.

Table 2.: The adaptivity parameters of ALNS.

After each segment j, the weights to be used in the next segment are calculated as

$$w_{i,j+1} = w_{ij}(1-r) + r\frac{\pi_i}{\theta_i}$$

Where π_i and θ_i are as previously described, and r is the reaction factor which controls how quickly the weights adjusts to the changes in the effectiveness.

The acceptance criteria can either be a simple 'Only accept improving solutions' or a simulated annealing acceptance criteria and the stop criteria could be based on the number of iterations or time.
Section 4.1 describes the construction heuristic, and 4.2 proposes two local search neighbourhoods. Section 4.3 describes the acceptance criteria used. Section 4.4 and 4.5 describes the destroy and repair methods. Section 4.6 describes the resetting mechanism, and lastly section 4.7 is a description of the overall algorithm.

4.1 CONSTRUCTION HEURISTIC

M. Solomon describes in [31] three sequential insertion heuristics, I1, I2 and I3. Out of these I1 is the most successful, and therefore this is used as the construction heuristic.

In Solomons I1 sequential insertion heuristics a route is first initialised with one customer, hereafter customers are inserted into this route as long as there are more that can be inserted. When there are no more customers to be inserted a new route is added and the process restarts. This continue as long as there are unrouted customers.

The customer to select in the beginning can either be the one geographically farthest from the depot, or the one with the lowest allowed service time.

The heuristic uses two different criteria, $c_1(i,u,j)$ and $c_2(i,u,j)$, in order to determine which customer to insert after initialising a route. These criteria are defined as

$$\begin{aligned} c_1(i,u,j) &= \alpha_1(c_{iu} + c_{uj} - \mu c_{ij}) + \alpha_2(b_{ju} - b_j) \\ \alpha_1 + \alpha_2 &= 1, \quad \alpha_1 \geq 0, \quad \alpha_2 \geq 0, \quad \mu \geq 0 \\ c_2(i,u,j) &= \lambda c_{0u} - c_1(i,u,j), \qquad \lambda \geq 0 \end{aligned}$$

Here c_{ij} denotes the distance between node i and j, and 0 is the depot. b_{ju} denotes the new time for service to begin at customer j, given u is inserted on the route, and b_j is the beginning

of service prior to the insertion.
The parameters α_1 and α_2 are weights for a distance and a time term, respectively. μ controls the importance of savings in distance, whereas λ is used to define how much the best insertion place for a customer depends on its distance to the depot.
Let $(i_0, i_1, i_2, \ldots, i_{m-1}, i_m)$ be the current route, where i_0 and i_m are the depot. After seeding the first customer the best feasible insertion cost is computed for all unrouted customers, u.

$$c_1\left(i(u), u, j(u)\right) = \min_{\rho=1,\ldots,m} c_1\left(i_{\rho-1}, u, i_\rho\right)$$

hereafter the best customer, u^* to be inserted in the route is the one for which,

$$c_2\left(i(u^*), u^*, j(u^*)\right) = \max_u \left\{c_2\left(i(u), u, j(u)\right) | u \text{ is unrouted and the route is feasible}\right\}$$

Customer u^* is then inserted into the route between $i(u^*)$ and $j(u^*)$. The overall algorithm can be seen in algorithm 1.

Algorithm 1 `Construction Heuristic`$(\alpha_1, \alpha_2, \mu, \lambda)$

```
D ← All Customers
i = 0
while |D| > 0 do
    r_i ← argmax_{i∈D} (c_{0i})
    while There exists at least 1 customer with a feasible insertion place in the route r_i do
        Find u*, i(u*) and j(u*)
        r_i ← r_i ∪ u* inserted between i(u*) and j(u*)
        D ← D \ {u*}
    end while
    x ← x ∪ r_i
    i ← i + 1
end while
return x
```

4.2 LOCAL SEARCH

In order to improve the solutions found by the repair methods, two local search neighbourhoods have been implemented. These neighbourhoods can be combined in different ways, which result in 6 different local search methods. Section 4.2.1 describes a simple neighbourhood based on relocating a single customer to another part of the solution, section 4.2.2 described a more complex neighbourhood that best can be described as a inter route 3Opt neighbourhood, lastly section 4.2.3 describes the different ways these two neighbourhoods can be combined.

4.2.1 Relocate Neighbourhood

The relocate neighbourhood is a simple neighbourhood that tries to move customers from the current position to a more affordable place in the solution.

For every customer it is examined if it is possible to move the customer to a cheaper place in the solution, and then moves the one that improves the current solution the most. Only moves that are better than the current best move are being fully explored (with respect to

feasibility), as this speed up the runtime. The size of the neighbourhood is $\mathcal{O}(n^2)$, as for every customer it is investigated if it can be moved to a position just before any other customer.
Instead of picking the overall best moves, all possible improving moves can be enumerated. Then using a roulette wheel selection principle to select the customer to be inserted. This stochastic method would help diversify the search, and might provide better results than the deterministic method, just picking the best move. Both methods have been implemented and section 6.1.2 test whether or not is a good idea to use this stochastic method instead of the deterministic one.

4.2.2 3Opt* Neighbourhood

The 3Opt* neighbourhood selects three arcs, from three different routes, in the current solution, and swap these arcs to examine if one of the swaps can improve the solution. This is similar to the 2Opt* neighbourhood that is commonly used for the VRPTW in the literature. The 3Opt* neighbourhood also considers these 2Opt* neighbourhood moves, and the 2Opt* is thus a part of the 3Opt* neighbourhood. The standard 3Opt neighbourhood only considers arcs from a single route, and have shown good results as an improvement heuristic for the Travelling Salesman Problem (TSP).

Consider three customers i, j and k, from three different routes, and let the arcs leaving these customers be (i, i^+), (j, j^+), (k, k^+). The 3Opt* neighbourhood examines all unique combinations of three customers and for every combination makes 5 different swaps of the three arcs, where no arcs are reversed. These 5 swaps are shown in figure 2, where the one in the upper left corner is the original solution.

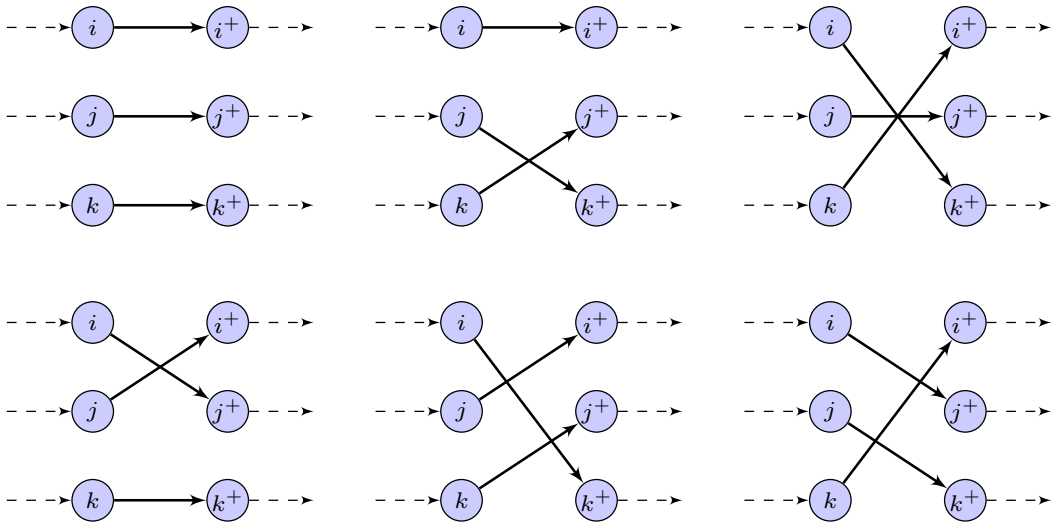


Figure 2.: 3Opt* neighbourhood for three customers, i, j and k. The solution in the upper left corner is the original solution.

Even though the three considered customers must be from three different routes, the size of the neighbourhood is still in the order of $\mathcal{O}(n^3)$. This means that this neighbourhood takes longer to explore than the Relocate neighbourhood, however it is also a larger neighbourhood, which means it could contain better solutions.

4.2.3 Local search methods

The local search method will be applied after the repair heuristic have been completed, if a certain criteria (to be explained in section 4.7) is satisfied. Combining the two previously described neighbourhoods can result in different local search methods.

The following 6 local search methods have been tested, and compared.

1. **Only Relocate (Reloc)**
   This method only uses the Relocate neighbourhood, and continues until the solution is local optimal with respect to the Relocate neighbourhood.

2. **Only 3Opt* (3Opt*)**
   This method only uses the 3Opt* neighbourhood, and continues until the solution is local optimal with respect to the 3Opt* neighbourhood.

3. **Adaptively choose between Relocate and 3Opt* (Adapt)**
   As with the destroy and insertion methods, this method assigns a probability to each of the two neighbourhood. The neighbourhood that is selected continues to improve the solution until the solution is local optimal with respect to the chosen neighbourhood. The weights are adaptively changed throughout the algorithm, similar to what has earlier been described.

4. **Relocate and 3Opt* (Re&3O)**
   This method searches both the Relocate an 3Opt* neighbourhood, and performs the move that is the overall best. This continues until the solution is local optimal with respect to both neighbourhoods.

5. **Random move, single optimal (Ra-1o)**
   This method considers both the Relocate and the 3Opt* neighbourhood. In each iteration of the local search method one of the neighbourhoods are selected, with equal probability, and the best move from the selected neighbourhood is performed. This continues until the solution is local optimal with respect to the chosen neighbourhood.

6. **Random move, twice optimal (Ra-2o)**
   Just like the Ra-1o method, but when the solution is local optimal for the selected neighbourhood, the other neighbourhood is selected. This continues until the solution is local optimal with respect to both neighbourhoods.

All of these local search method will be tested in section 6.1.2, to see which of them provide the best results.

4.3 ACCEPTANCE CRITERIA

The presented algorithm uses a simulated annealing acceptance criteria.
Simulated annealing is based on the observation, that accepting a worse solution can help diversify the search. However, it is discouraged to accept solutions that are a lot worse than the current, and thus a probability of accepting an inferior solution is based on the fitness of the

solution compared to the current solution. The probability of accepting a worse solution, x, is given by

$$e^{\frac{f(x^c) - f(x)}{T}}$$

Here x^c is the current solution and x is the proposed solution. T is the temperature parameter, $T > 0$. The temperature is decreased in each iteration by a cooling factor ζ in order to make it less likely to choose an inferior solution in the end of the algorithm, $0 < \zeta < 1$. If the attempted solution is better than the current the attempted solution is accepted with probability 1. The simulated annealing acceptance criteria is shown in algorithm 2, where x^b denotes the best solution

Algorithm 2 `Simulated Annealing`(x, x^c, x^b, T, ζ)

```
if f(x) < f(x^c) then
    x^c ← x
    if f(x) < f(x^b) then
        x^b ← x
    end if
else
    Sample a random number 0 ≤ y < 1
    if y ≤ e^((f(x^c) − f(x))/T) then
        x^c ← x
    end if
end if
T ← T · ζ
return (x^c, x^b, T)
```

The starting temperature T_{start} is chosen based on the solution before entering the main loop in the ALNS algorithm, as proposed by [29]. It is calculated such that a solution w times worse than the starting solution is accepted with a probability of 50%, in the first iteration.

$$T_{start} = -\frac{w \cdot f(x)}{\ln(0.5)}$$

The parameter w needs to be tuned as well as the cooling factor ζ.

4.4 DESTROY METHODS

Five different destroy methods have been implemented. For 4 of the methods ϵ percentage of the solution is destroyed, where ϵ is a random number between

$$\epsilon \in \{0.05, \pi\}$$

Here π is a parameter that describes how big a part of the solution is allowed to be destroyed. For the fifth method, at least ϵ percentage of the solution is destroyed.

4.4.1 Random Removal

The most simple destroy methods is just to remove a number of random customers.
The algorithm is as shown in algorithm 3 where ϵ is the percentage of a solution to be destroyed.

Along with providing surprisingly good results it also helps to diversify the search, such that the solution space can be searched more extensively.

Algorithm 3 `Random Removal`(x, ϵ)

1: $D \leftarrow \emptyset$
2: **while** $|D| < \epsilon \cdot |N|$ **do**
3: $\quad i \leftarrow$ A random customer $i \in x$
4: $\quad x \leftarrow x \setminus \{i\}$
5: $\quad D \leftarrow D \cup \{i\}$
6: **end while**
7: **return** (x, D)

4.4.2 Worst Removal

The potential for improving the solution is greater if you remove an 'expensive' part of the solution. This can be achieved by removing the customers that adds the most to the current solution.
For every customer it is calculated how much the solution will decrease if the customer is removed from the solution. Instead of just removing the worst customer, a randomness factor is introduced such that the algorithm is not entirely deterministic. This is done by sorting the customers after decreasing removal price, sampling a random number, y, and removing the element $\lfloor y^p \cdot |L| \rfloor$ where $|L|$ is number of rows in the list. p is a parameter determining how likely an element is to be chosen. If $p = 1$ every element has equal probability of being chosen, the higher p gets the less likely the bottom elements become and the more likely the top elements become. The algorithm is shown in algorithm 4 where ϵ is how big a percentage of the solution that should be destroyed. The `RemovalCost` in line 4 is how much the solution is improved by removing customer c from the solution, x.

Algorithm 4 `Worst Removal`(x, ϵ, p_{worst})

1: $x^* \leftarrow x$
2: $D \leftarrow \emptyset$
3: **while** $|D| < \epsilon \cdot |N|$ **do**
4: $\quad$ Array $L =$ All customers $c \in x$, sorted by decreasing `RemovalCost`(c, x)
5: $\quad$ Pick a random number $0 \leq y < 1$
6: $\quad$ Choose customer $i = L\,\lfloor y^{p_{worst}} \cdot |L| \rfloor$
7: $\quad x \leftarrow x \setminus \{i\}$
8: $\quad D \leftarrow D \cup \{i\}$
9: **end while**
10: **return** (x, D)

There is two ways to calculate the costs. All of the cost can be based on the solution before any removal occurred, that is all of the cost can be based on x^*. Or it can be based on the changed solution, x, such that the removal cost might change in each iteration. Milthers tests in [19] both of these methods and concludes that the second one is by far the best. Therefore only the second one has been implemted, as shown in algorithm 4. p_{worst} is a parameter that needs to be tuned.

4.4.3 Shaw Removal

If the customers removed are not related to each other there might only exists a few feasible insertions with the same number of routes. Hence you might end up with more routes that, most commonly, results in a worse solution. P. Shaw proposed in [30] that related customers are removed.
Consider two customers i and j, a common relatedness criteria used in VRPTW, is based on the distance between i and j, the difference between the time i and j are visited in the current solution, the difference in demand and whether or not they are on the same route. Formally the relatedness $R(i, j)$ is calculated as

$$R(i,j) = \varphi c_{ij} + \chi|t_i - t_j| + \psi|d_i - d_j| + \omega T_{ij}$$

Here t_i is the time customer i is visited, and T_{ij} is whether or not two customers i and j are on the same route in the current solution. The first term is the distance between i and j weighted by φ, the next term is the difference between arrival times, and the third term is the difference between the demands. A low value of $R(i, j)$ means a higher relatedness. φ, χ, ψ and ω are parameter that needs to be tuned.
In the Shaw Removal algorithm a random customer is selected in the beginning and the relatedness to this customer is calculated for the rest of the customers. The customers are listed according to increasing relatedness, $R(i, j)$. Just as in the `Worst Removal` procedure a randomness factor is introduced. The overall algorithm is as shown in algorithm 5.

Algorithm 5 `Shaw Removal`(x, ϵ, p_{shaw})

1: $c \leftarrow$ `Randomcustomer`
2: $D \leftarrow c$
3: **while** $|D| < \epsilon \cdot |N|$ **do**
4: Array L = All customers $r \in x \setminus D$, sorted by increasing relatedness, $R(r, c)$
5: Pick a random number $0 \leq y < 1$
6: Choose customer $i = L\lfloor y^{p_{shaw}} \cdot |L| \rfloor$
7: $D \leftarrow D \cup \{i\}$
8: **end while**
9: $x \leftarrow x \setminus D$
10: **return** (x, D)

Here ϵ is how big a percentage of the solution should be destroyed and p_{shaw} is a parameter describing how likely a given element is to be chosen, both similar to the parameters for `Worst Removal`.
The random customer in `Randomcustomer` can be chosen in different ways, either it can be completely random, or it can use the same idea used in `Worst Removal` - making the worse customers more likely to be chosen. Both methods have been implemented, and the second one will be denotes as $\texttt{Shaw}_{\texttt{Worst}}$ `Removal`.

4.4.4 Random Route Removal

The cost of a solution is most commonly cheaper the fewer routes that can be used.
To try to decrease the number of routes in the current solution, a method that removes random routes have been implemented. A pseudocode of the algorithm can be seen in algorithm 6. This algorithm destroy routes until at least ϵ percentage have been removed. Hence this could, in theory, destroy the full solution.

Algorithm 6 `Random Route Removal`(x, ϵ)

```
D ← ∅
while |D| < ε · |N| do
    r ← A random route r ∈ x
    for all Customers, c ∈ r do
        D ← D ∪ {c}
    end for
    x ← x \ {r}
end while
return (x, D)
```

4.5 REPAIR METHODS

This section describes two repair methods, `Greedy Insertion` and the more advanced `Regret Insertion`. Both of these are purely deterministic as none of them depends on a random variable.

4.5.1 Greedy Insertion

Greedy insertion is a simple insertion heuristic, inserting the customer that adds the least to the overall cost.
Given a partial solution x, and the customers to be inserted, D, the algorithm runs as shown in algorithm 7, where $f_i(c)$ is the i'th best insertion place for customer c.

Algorithm 7 `Greedy Insertion`(x, D)

```
while D ≠ ∅ do
    for all i ∈ D do
        r(i) ← Best insertion route for customer i
        p(i) ← Best insertion place in route r(i) for customer i
        f1(i) ← Price of best insertion for customer i
    end for
    c ← argmin_{i∈D}(f1(i))
    x ←Insert c in route r(c) at position p(c)
    D ← D \ {c}
end while
return x
```

It is always feasible to add a customer to a new route, therefore $f_1(i)$ is, in the beginning of every iteration, initialised to the cost of adding customer i into a new route. The feasibility of a insert position is only checked if it improves the current best insert position for the given customer.

If a customer, c_1, is inserted in a route, r_1, in iteration i, and another customer, c_2, can be inserted cheapest into another route, r_2, then there is no need to search through all of the neighbourhood for customer c_2 in iteration $i+1$. For every route $r \neq r_1$, c_2 can still be inserted cheapest into the route r_2. r_1 is the only route changed therefore it is sufficient to search through this route for a cheaper insertion place. On the other hand if a route, r is changed we will have to go through the full neighbourhood for every customer that had this route, r, as the best insertion. We also need to go through the whole neighbourhood in the first iteration. These considerations are not explicitly shown in algorithm 7, but it speeds up the execution considerably.

This is one of the most simple insertion heuristic, not taking anything else into consideration than what seems to be best at the moment.

4.5.2 Regret Insertion

There might be a customer where the difference between the best insertion place and the second best insertion place is high, hence if the customer is not inserted now it might be forced to be inserted at the second best place. This idea is formalised in the regret heuristic outlined in algorithm 8.

Algorithm 8 `Regret Insertion`(x, D)

while $D \neq \emptyset$ **do**
  for all $i \in D$ **do**
    $r(i) \leftarrow$ Best insertion route for customer i
    $r_2(i) \leftarrow$ Second best insertion route for customer i
    $p(i) \leftarrow$ Best insertion place in route $r(i)$ for customer i
    $f_1(i) \leftarrow$ Price of best insertion for customer i
    $f_2(i) \leftarrow$ Price of second best insertion for customer i
  end for
  $c \leftarrow \text{argmax}_{i \in D} \left(f_2(i) - f_1(i)\right)$
  $x \leftarrow$Insert c in route $r(c)$ at position $p(c)$
  $D \leftarrow D \setminus \{c\}$
end while
return x

As it is always possible to add a new a route, $f_1(i)$ and $f_2(i)$ are both initialised to be the cost of adding a new route, $c_{i,0} + c_{0,i}$. Thus a new route is only added when there does not exist an insertion place for any customer, $c \in D$, lower than the cost of adding a new route.
Similar to the greedy insertion procedure it is not necessary to search through the full neighbourhood in every iteration, this can be done in the same way as for the greedy insertion. But here you have to search the full neighbourhood if one of the two best routes are changed.
The algorithm shown in algorithm 8 is a so-called `2-Regret Insertion`, where you compare the best insertion place with the second best. More generally `k-Regret Insertion` compares the best insertion cost with the k'th best insertions.

$$\underset{i \in D}{\text{argmax}} \left(\left(\sum_{j=2}^{k} f_j(i) \right) - f_1(i) \right)$$

Only the `2-Regret Insertion` shown in algorithm 8 have been implemented.

4.6 RESETTING MECHANISM

In the beginning of the algorithm the temperature is high, and hence the probability of accepting an inferior solution is high. Hence the algorithm might find a solution in the beginning, in a neighbourhood with high quality solutions, and then slowly move away from this neighbourhood, and not return. The resetting mechanism allows the algorithm to return to this high-quality area.
If the solution have not improved in γ consecutive iterations the current solution is set to the best solution ($x^c \leftarrow x^b$), such that the algorithm will continue from the high quality area. After

resetting, at least γ iterations must performed before resetting again.
This resetting mechanism is embedded into the acceptance criteria.

4.7 THE OVERALL ALNS ALGORITHM

In the main algorithm there are three more parameters, κ and η and ψ. κ determines when to use `Local Search`(x) in the ALNS algorithm, and η is the segmentsize used in the ALNS algorithm. If $\kappa = 0.03$ local search is performed on solutions that are at most 3% worse than the best solution. ψ is a noise parameter used in the local search and insertion heuristics. When $\psi = 2\%$ it means that every time the cost of an insertion is calculated it is multiplied with $1 + y$ where y is a random number $-0.02 \leq y \leq 0.02$. This adds some stochasticity to the otherwise deterministic methods, which can help to diversify the search, the parameter will be tested in chapter 6 in order to see if it helps provide better results. A flowchart of the overall algorithm can be seen in fig. 3. The stopping criteria of the algorithm is 25000 iterations.

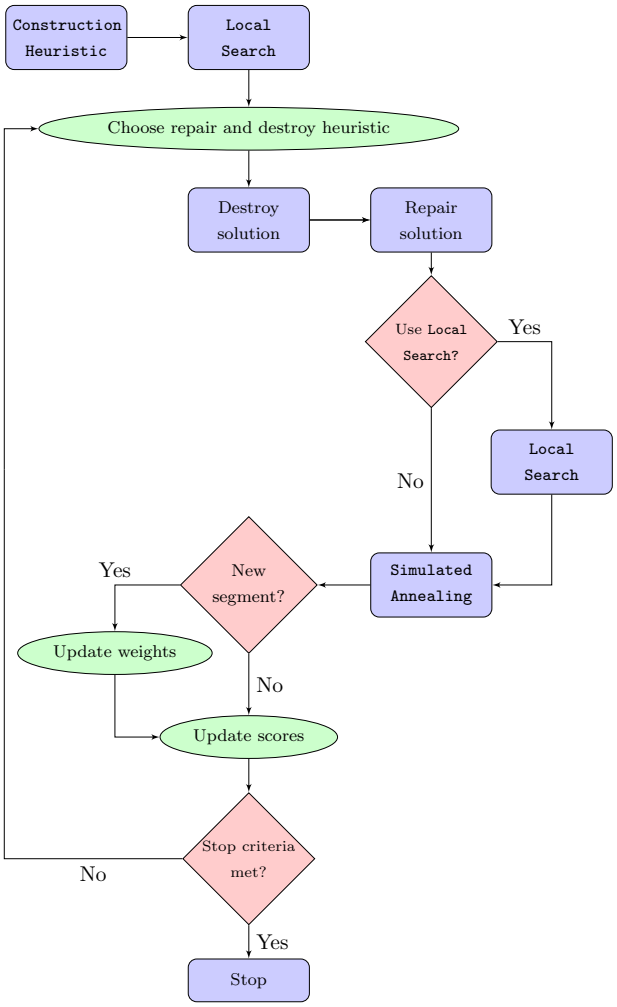


Figure 3.: Flow chart of the overall algorithm.

5 EXACT SUB METHODS

In this chapter some exact submethods that can be used in the ALNS algorithm will be described. These submethods are smaller sub problems that can be solved to optimality relatively fast and might be able to improve the current best solution.

5.1 SET PARTITIONING PROBLEM

Throughout the ALNS algorithm many different routes are generated. All of these routes can be collected and considered by solving a set partitioning problem. This way all the routes generated throughout the algorithm can be considered, and combined in an optimal way. Though only a subset of all the routes are generated, this solution is not guaranteed to be the overall optimal solution.
Define the set Ω_{gen} as the set of generated routes, this is of course a subset of all the routes $\Omega_{gen} \subset \Omega$, the cost of a route is denoted as c_p for a route $p \in \Omega_{gen}$. Furthermore a_{ip} will contain information of which customer a route $p \in \Omega_{gen}$ visits, that is

$$a_{ip} = \begin{cases} 1 & \text{If route } p \in \Omega_{gen} \text{ visits customer } i \\ 0 & \text{Otherwise.} \end{cases}$$

The decision that needs to be made is whether or not a route is used in the solution, this can be modelled by a binary variable.

$$y_p = \begin{cases} 1 & \text{If route } p \in \Omega_{gen} \text{ is used in the solution} \\ 0 & \text{Otherwise.} \end{cases}$$

With this the set partitioning problem can be modelled as seen in model (2).

$$\texttt{Min Z} = \sum_{p \in \Omega_{gen}} c_p y_p \tag{2a}$$

`Subject to:`

$$\sum_{p \in \Omega_{gen}} a_{ip} y_p = 1 \qquad \forall i \in V \setminus \{0, n+1\} \tag{2b}$$

$$y_p \in \{0, 1\} \qquad \forall\, p \in \Omega_{gen} \tag{2c}$$

(2a) is the objective function, minimising the cost of using the routes. Constraint (2b) make sure that every customer is visited exactly once (here V is defined as in chapter 2), and lastly (2c) defines the variables as binary.
The constraint (2b) could be changed to a set cover constraint, stating that each customer must be visited at least once,

$$\sum_{p \in \Omega_{gen}} a_{ip} y_p \geq 1 \qquad \forall i \in V \setminus \{0, n+1\} \tag{3}$$

Solutions where a customer is visited more than once will be considered infeasible by the ALNS algorithm. But if the solution visits a customer twice this customer can be removed from one of the routes, which would lead to a better solution. Instead of removing customers from a route, I have ensured, through constrain (2b), that this does not happen, even though it could provide better results.
As all the routes $p \in \Omega_{gen}$ are feasible it is not necessary to include the time window constraints and demand constraint, as these are satisfied in each of the single routes.

Many of the routes generated throughout the algorithm will be the same. By maintaining a hashtable of all the previously generated routes it is possible to check, in constant time, for duplicates. This speeds up the process as the set of generated routes will be smaller, which means it takes up less memory, and more importantly, can be solved faster. Also, the best solution is given as input to the model, such that CPLEX can start from this solution. The timelimit is set to 100 s, but as the best solution is given as input the method will output a solution that is no worse than the best solution.

But when should you use this? You could solve the set partitioning problem several times throughout the algorithm, or just in the end. Solving the set partitioning problem several times throughout the algorithm might help to explore new areas of the solution space. Solving the SP problem after termination of the ALNS algorithm allows the consideration of all generated routes. The parameter ρ denotes the number of iterations between each calls to the set partitioning model. This parameter will be tuned in chapter 6.

If the set partitioning problem provides a new best solution, the current solution is updated as well, so this new area can be further explored. If it returns the current best solution, there are three possibilities: 1) Do nothing, 2) set the current solution to the best solution, 3) increase the temperature and set the current solution to the best solution. Doing nothing lets the algorithm further explore the current region of the solution space. Setting the current solution to the best solution is a lot like the resetting mechanism described in chapter 4. If the set partitioning problem returns the same solution two consecutive times, it might be because the set partitioning problem is solved too often, or the temperature is too low. ρ describes how often the set partitioning is solved, and will be tested and assigned the value that seems to give the best performance. Therefore the temperature could be increased in order to allow the algorithm to search a larger solution space.
This implementation uses a combination of option 1) and 3) when the set partitioning model returns the same solution as the last time the set partitioning model was solved. If less than 50% of the total number of iterations has been executed it does nothing, and if at more than 50% of the total number of iterations has been executed it sets the current solution to the best solution and reheats the temperature to 5% more than it was when the best solution was achieved. The reason behind this is to allow a large part of the solution space to be explored in the first half of the algorithm, and in the second half, focus more on the region that has shown to produce good solutions, but also increase the temperature to allow for the algorithm to explore solutions close to this neighbourhood.

5.2 ROUTE OPTIMISATION (TSPTW)

The routes generated throughout the algorithm are not necessarily optimal with respect to the visited customers, there might exist a route visiting the same set of customers, that is shorter. The problem of finding the shortest single route between a set of customers can be formulated as a Mixed Integer Program (MIP). Consider the set of customer that is visited in an already generated route. As the generated route is feasible you know that the capacity constraint is satisfied, and this constraint can thus be omitted. The solution should still be a route, i.e visit every considered customer exactly once, and return to the depot. Furthermore the time-windows should also be satisfied. This problem is know as the Travelling Salesman Problem with Time Windows (TSPTW).

Define n as the number of customers, and $V = \{0, 1, \ldots, n+1\}$ as nodes, where 0 and $n+1$ are the depot. Moreover define A as the full set of arcs (i, j). Let $V^+(i)$ be the nodes that can be reached from the node i, and $V^-(i)$ the sets of nodes that can reach node i. The decision needed is at which time the customer should be visited, and which arcs that should be used. This can be modelled by a continuous time variable, w_i, and a binary variable, x_{ij} defined as

$$x_{ij} = \begin{cases} 1 & \text{If arc } (i,j) \text{ is traversed} \\ 0 & \text{Otherwise.} \end{cases}$$

In model (4) presented below c_{ij} denotes the cost of traversing arc (i, j) and t_{ij} is the time for travelling along arc (i, j), which includes service time at node i.

$$\texttt{Min Z} = \sum_{(i,j) \in A} c_{ij} x_{ij} \tag{4a}$$

`Subject to:`

$$\sum_{j \in V^+(i)} x_{ij} = 1 \qquad \forall i \in V \setminus \{n+1\} \tag{4b}$$

$$\sum_{k \in V^-(i)} x_{ki} = 1 \qquad \forall i \in V \setminus \{0\} \tag{4c}$$

$$w_i + t_{ij} \leq w_j + (1 - x_{ij}) \cdot M \qquad \forall (i,j) \in A \tag{4d}$$

$$a_i \leq w_i \leq b_i \qquad \forall i \in V \tag{4e}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A \tag{4f}$$

$$w_i \geq 0 \qquad \forall i \in V \tag{4g}$$

In model (4), the objective function (4a) minimises the overall cost of the solution. Constraint (4b) ensures that every node is left except the end node, and (4c) ensures that every node except the start node is entered. Constraint (4d) and (4e) ensures the time windows are satisfied. This is done similarly to the time windows constraint (1f) and (1g) in the Vehicle Routing Problem with Time Windows formulation in chapter 2.

The Travelling Salesman Problem with Time Windows, is known to be $\mathcal{NP}$-hard, and is also hard in practice. Therefore it is not an good idea to try to optimise each and every route, as some of the generated routes will be bad even when optimised. Instead we wish to determine

which routes are good and try to optimise these. A way this can be done is by considering the set of generated routes, and solve a set partitioning problem to LP-optimality. The routes which are in the optimal basis must possess some desirable property, and therefore it is examined if these routes can be optimised. Afterwards the original route is marked as 'Optimised' such that we do not try to optimise that route again. Furthermore, if there exists an optimised version that route is marked 'Optimised' and is added to the set of generated routes.
Doing so the Route Optimisation cannot be tested without the inclusion of the set partitioning model, as the set partitioning model is the only way to include the optimised route in the solution.

During testing it was revealed that not all problems could be solved to optimality within reasonable time, and therefore a timelimit was set to 10 s, if this timelimit is reached the route is marked as 'Optimised', even though it has not been optimised. This is to prevent solving the TSPTW again for this set of customers.

5.3 EXACT INSERTION METHOD

Even though the Regret and Greedy insertion heuristics provide good results, it would be interesting to see whether or not an exact insertion method could improve the overall result. As time is an important factor it is crucial that the exact method is relative fast, therefore the devised model should be as compact as possible, and therefore not everything will be accounted for in the model.

Consider a partial solution, where a set of customers, I, need to be inserted. In this model a customer can be inserted between two already inserted customers.

Define $P(r)$ as the sets of positions where a customer can be inserted into in route $r \in R$. Let $p(r)_-$ denote the customer before position $p(r)$ and $p(r)_+$ as the customer after position $p(r)$, see fig. 4 for a visual illustration. Figure 4 considers a part of a route r. The green diamonds in the figure symbolises the insertion positions, and the associated number denotes the position, $p(r)$. The positions are used to reference the customers, which is shown by the labels associated with the blue nodes.

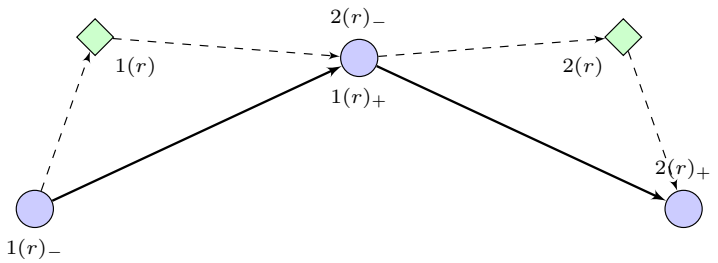


Figure 4.: Illustration of the positions, $p(r)_-$ and $p(r)_+$.

The parameters used in the model are listed and explained below:

- Q: Capacity of the vehicles
- δ_r: Current demand on route r. That is $Q - \delta_r$ is the remaining capacity.
- d_i: Demand for customer $i \in I$

- $c_{irp(r)}$: Cost of inserting customer i at position $p(r)$ in route r
- c_i: Cost of inserting customer i into an empty route.
- $\bar{t}_{i,j}$: The time it takes to go from one customer i to customer j, including service time at customer i.
- a_i: The time from when node $i \in V$ accepts deliveries.
- b_i: The time when node $i \in V$ no longer accepts deliveries.

Where the set V is the full set of nodes, as defined in chapter 2.

In the model (5), there are 4 sets of variables, 2 binary sets of variables, and two continuous sets of variables. The variables $x_{irp(r)}$ and σ_i are related to where the customers are inserted in the solution. $x_{irp(r)}$ is a binary variable denoting, whether or not customer i is inserted in position $p(r)$ in route r.

$$x_{irp(r)} = \begin{cases} 1 & \text{If customer } i \text{ is inserted in position } p(r) \text{ in route } r \\ 0 & \text{Otherwise.} \end{cases}$$

And σ_i denotes whether or not a customer is inserted into an otherwise empty route. This also ensures that the model cannot be infeasible, providing that the instance is feasible.

$$\sigma_i = \begin{cases} 1 & \text{If customer } i \text{ is inserted into an empty route} \\ 0 & \text{Otherwise.} \end{cases}$$

The variables w_i and $t_{rp(r)}$ are related to the time a customer is visited. w_i denotes the time an already inserted customer is visited. $t_{rp(r)}$ denotes at which time position $p(r)$ in route r is visited. Figure 5 explains the relations between these variables. The blue nodes (labelled 0 through 5) make out the original route. The green nodes are the positions where customers can be inserted. The solid line is the route after insertion, and a customer is inserted between the nodes labelled 3 and 4.

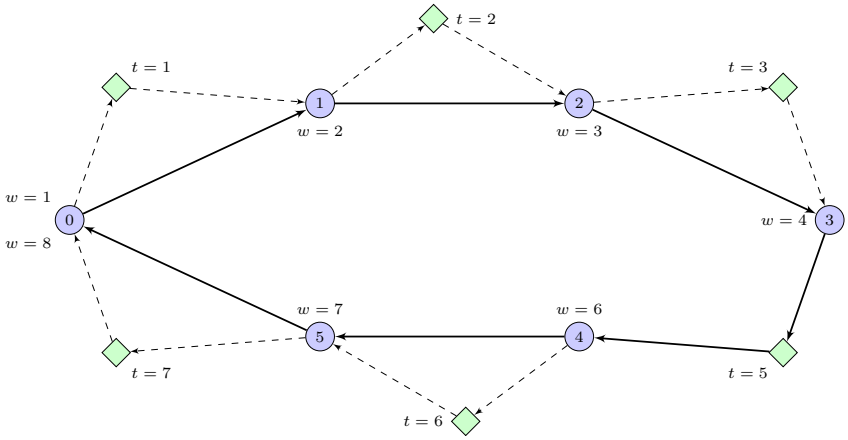


Figure 5.: Explanation of the connection between the t and w variables.

Figure 5 shows a feasible assignment of the t and w variables when all travelling times are $\bar{t}_{ij} = 1$ for all (i, j). If no customer is inserted in position $p(r)$ in route r, then the corresponding t

variable should be greater than or equal to the time the previous customer is visited, $t_{rp(r)} \geq w_{p(r)_-}$, which is also shown in the figure.
With all this the model can be stated as seen below in model (5).

$$\texttt{Min Z} = \sum_{i \in I} \sum_{r \in R} \sum_{p \in P(r)} c_{irp} x_{irp} + \sum_{i \in I} c_i \sigma_i \tag{5a}$$

Subject to:

$$\sum_{r \in R} \sum_{p \in P(r)} x_{irp} + \sigma_i = 1 \qquad \forall i \in I \tag{5b}$$

$$\sum_{i \in I} \sum_{p \in P(r)} d_i x_{irp} \leq Q - \delta_r \qquad \forall r \in R \tag{5c}$$

$$\sum_{i \in I} x_{irp} \leq 1 \qquad \forall r \in R, p \in P(r) \tag{5d}$$

$$t_{rp} \geq w_{p(r)_-} + \sum_{i \in I} \bar{t}_{p(r)_-,i} x_{irp} \qquad \forall r \in R, p \in P(r) \tag{5e}$$

$$w_{p(r)_+} \geq t_{rp} + \sum_{i \in I} \bar{t}_{i,p(r)_+} x_{irp} + \left(1 - \sum_{i \in I} x_{irp}\right) \bar{t}_{p(r)_-,p(r)_+} \qquad \forall r \in R, p \in P(r) \tag{5f}$$

$$\sum_{i \in I} a_i x_{irp} \leq t_{rp} \leq \sum_{i \in I} b_i x_{irp} + \left(1 - \sum_{i \in I} x_{irp}\right) b_{p(r)_-} \qquad \forall r \in R, p \in P(r) \tag{5g}$$

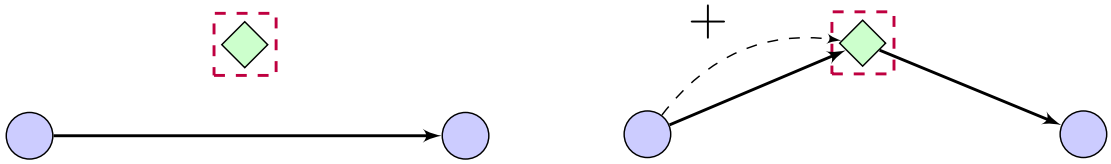
$$a_{0(r)_-} \leq w_{0(r)_-} \leq b_{0(r)_-} \qquad \forall r \in R \tag{5h}$$

$$a_{p(r)_+} \leq w_{p(r)_+} \leq b_{p(r)_+} \qquad \forall r \in R, p \in P(r) \tag{5i}$$

$$x_{irp} \in \{0, 1\} \qquad \forall i \in I, r \in R, p \in P(r) \tag{5j}$$

$$\sigma_i \in \{0, 1\} \qquad \forall i \in I \tag{5k}$$

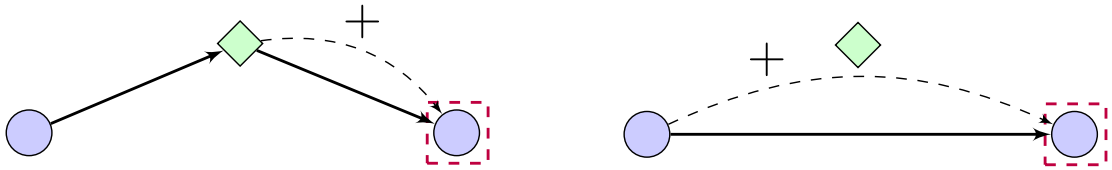
The objective function, (5a), minimises the overall cost of inserting the removed customers. Constraint (5b) ensures that either a customer is inserted into one of the existing routes or it is inserted into its own route. Constraint (5c) ensures that the capacity is not exceeded, and constraint (5d) ensures that at most a single customer can be inserted into a position.
Constraint (5e)-(5i) ensures that the time windows are satisfied. Equation (5e) sets the t variable for every position. The sum on the right hand side makes sure that the correct travel time is added, if any customer is inserted in the corresponding position. Figure 6 illustrates how t_{rp} is updated. The encircled node denote which node the time is updated for. The solid arcs denote the route and the dashed arcs symbolises which travel times are added to the $w_{p(r)_-}$ variable.
If no customer is inserted in the position, then the constraint (6) becomes $t_{rp} \geq w_{p(r)_-}$ as shown in fig. 6a. If a customer, $\hat{i}$, is inserted, then $t_{rp} \geq w_{p(r)_-} + \bar{t}_{p(r)_-,\hat{i}}$ must be satisfied, as shown in fig. 6b.



(a) $t_{rp} \geq w_{p(r)_-}$ **(b)** $t_{rp} \geq w_{p(r)_-} + \bar{t}_{p(r)_-,\hat{i}}$

Figure 6.: Illustration of how t_{rp} is updated in eq. (5e).

Constraint (5f) sets the w variable for the already inserted customer. Here there are two cases to consider; no customer is inserted into the previous position, or a customer is inserted into the position, both cases are shown in fig. 7. The encircled node denote which node the time is updated for. The solid arcs denote the route and the dashed arcs symbolises which travel times are added to the t_{rp} parameter.



(a) $w_{p(r)_+} \geq t_{rp} + \bar{t}_{\hat{i},p(r)_+}$ **(b)** $w_{p(r)_+} \geq t_{rp} + \bar{t}_{p(r)_-,p(r)_+}$

Figure 7.: Illustration of how w_i is updated in eq. (5f).

Consider an arbitrary position, $p'(r)$ and assume there exists an $\hat{i} \in I$ such that $x_{\hat{i}rp'} = 1$, then constraint (5f) becomes

$$w_{p'(r)_+} \geq t_{rp'} + \bar{t}_{\hat{i},p'(r)_+}$$

here the travel time between the inserted customer and the considered customer is accounted for. This case is shown fig. 7a. In the second case, fig. 7b, no customer is inserted in the position, eq. (5f) then becomes

$$w_{p'(r)_+} \geq t_{rp'} + \bar{t}_{p'(r)_-,p'(r)_+}$$

Where the travel time from the previous customer instead is taken into account.
Constraint (5g) defines the bounds on the t variables. Again, lets consider an arbitrary position $p'(r)$, if a customer $\hat{i} \in I$ is inserted into this position the bounds on the corresponding t variable becomes

$$a_{\hat{i}} \leq t_{rp'} \leq b_{\hat{i}}$$

If no customer is inserted the bounds enforced by constraint (5g) becomes

$$0 \leq t_{rp'} \leq b_{p'(r)_-}$$

But due to constraint (5e) the bounds on the t variable becomes

$$w_{p'(r)_-} \leq t_{rp'} \leq b_{p'(r)_-}$$

Where $b_{p'(r)_-}$ can be considered as a 'big-M'.
Constraints (5h) and (5i) defines the bounds on the w variables, and (5j) and (5k) respectively defines the x- and σ-variable as binary.

This model has some limitations. For example it cannot insert multiple customers between two already inserted customers, and if a new route is added it is not be able to add more customers to this route. If two customers next to each other are removed by the destroy heuristic, the model (5) is not able to recreate the original solution, but the greedy and the regret insertion heuristic could recreate the solution. Therefore this insertion method is coupled with a new destroy heuristic that do not remove two consecutive customers, such that it will not return a

solution worse than the original solution. However, due to the limitations, the regret and greedy insertion method could give a solution better than the one found by model (5).
Tests on a diverse set of instances, all with 100 customers, where a random number between 5 and 35 customers were deleted and approximately 150.000 executions shows that this model on average finds the optimal solution in 0.05 seconds. Even though this is quite fast, the method is still slower than the two heuristic methods. The exact method should however provide good results, but as the execution time for the exact insertion method is substantially larger than the two heuristics (regret and greedy), it is not a good idea to include this insertion method in the adaptivity of the algorithm. This is because this method is expected to outperform the two heuristic, and this would imply that this method is used most often, which is not desirable due to the extra time this would lead to. Instead a parameter, θ_1, is introduced. θ_1 is the probability of using this exact insertion method in a given iteration.

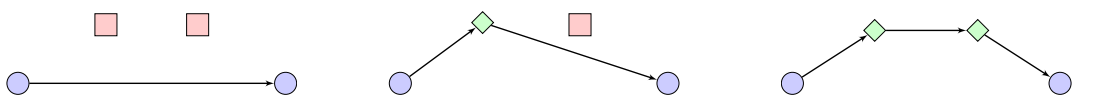
5.4 EXACT DESTROY/INSERTION

Model (5), as presented in section 5.3, assumes that a set of customers have already been removed from the solution and then finds the overall best place to re-insert all of these customers. This is somehow only a suboptimal approach as the solution heavily depends on which customers already are removed from the solution. Instead you can combine the destroy and insertion procedure into a single, more complex, model.

The variable y_i will denote whether or not a customer, i is removed from the solution, that is;

$$y_i = \begin{cases} 1 & \text{If customer } i \text{ is removed from the solution} \\ 0 & \text{Otherwise.} \end{cases}$$

The parameter γ_i will denote the profit of removing customer i. The rest of the variables, and parameters are as described in section 5.3.
In the presented model two customers next to each other cannot both be removed. Furthermore if a customer is removed there cannot be inserted a customer in either of the two positions next to the removed customer, and as in model (5) two customers cannot be inserted into a single position. These three situations are illustrated in fig. 8, where the blue nodes are nodes in the original solution, and the new solution. The red rectangles are removed customers, and the green nodes are inserted customers. Making these configurations infeasible ensures that it is easy to calculate the cost of inserting a customer, as we know the customer before and after. Similar it also makes it easier to compute the profit of removing a customer.



(a) Two consecutive customers removed.

(b) Customer inserted next to a removed customer.

(c) Two consecutive customers inserted.

Figure 8.: Disallowed solutions in model (6).

The overall model is presented below in model (6).

$$\texttt{Min Z} = \sum_{i\in I}\sum_{r\in R}\sum_{p\in P(r)} c_{irp}x_{irp} + \sum_{i\in I} c_i\sigma_i - \sum_{i\in I}\gamma_i y_i \tag{6a}$$

Subject to:

$$\sum_{r\in R}\sum_{p\in P(r)} x_{irp} + \sigma_i = y_i \qquad \forall i \in I \tag{6b}$$

$$\sum_{i\in I}\sum_{p\in P(r)} d_i x_{irp} \le Q - \delta_r + \sum_{i\in r} d_i y_i \qquad \forall r \in R \tag{6c}$$

$$\sum_{i\in I} x_{irp} \le 1 - \left(y_{p(r)_-} + y_{p(r)_+}\right) \qquad \forall r \in R, p \in P(r) \tag{6d}$$

$$y_{p(r)_-} + y_{p(r)_+} \le 1 \qquad \forall r \in R, p \in P(r) \tag{6e}$$

$$y_0 = 0 \tag{6f}$$

$$t_{rp} \ge w_{p(r)_-} + \sum_{i\in I} \bar{t}_{p(r)_-,i} x_{irp} \qquad \forall r \in R, p \in P(r) \tag{6g}$$

$$w_{p(r)_+} \ge t_{rp} + \sum_{i\in I} \bar{t}_{i,p(r)_+} x_{irp} + \bar{t}_{p(r)_-2,p(r)_+} y_{p(r)_-} \left(1 - \sum_{i\in I} x_{irp} - \left(y_{p(r)_-} + y_{p(r)_+}\right)\right) \bar{t}_{p(r)_-,p(r)_+} \qquad \forall r \in R, p \in P(r) \tag{6h}$$

$$\sum_{i\in I} a_i x_{irp} \le t_{rp} \le \sum_{i\in I} b_i x_{irp} + \left(1 - \sum_{i\in I} x_{irp}\right) b_{p(r)_-} \qquad \forall r \in R, p \in P(r) \tag{6i}$$

$$a_{0(r)_-} \le w_{0(r)_-} \le b_{0(r)_-} \qquad \forall r \in R \tag{6j}$$

$$\left(1 - y_{p(r)_+}\right) a_{p(r)_+} \le w_{p(r)_+} \le \left(1 - y_{p(r)_+}\right) b_{p(r)_+} + y_{p(r)_+} b_{p(r)_-} \qquad \forall r \in R, p \in P(r) \tag{6k}$$

$$x_{irp} \in \{0,1\} \qquad \forall i \in I, r \in R, p \in P(r) \tag{6l}$$

$$\sigma_i \in \{0,1\} \qquad \forall i \in I \tag{6m}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \cup \{0\} \tag{6n}$$

In model (6), the objective function, (6a) calculates the overall profit of removing and reinserting the customers. Constraint (6b) makes sure that if a customer is removed, then it must also be reinserted into the solution. Furthermore constraint (6b) also ensures that if a customer is not removed, then it cannot be inserted. Constraint (6c) is the capacity constraint, making sure that the capacity constraint on the trucks are not exceeded. The constraint (6d) ensures that a customer only can be inserted into a position if none of the two customers next to the position are removed, and if so, only a single customer can be placed at the position, thus making the two situations shown in fig. 8b and fig. 8c infeasible. Equation (6e) ensures that two consecutive customers cannot be removed, corresponding to the situation shown in fig. 8a, this constraint is not strictly necessary as it is also enforced by constraint (6d). Constraint (6f) makes sure that the depot is not removed.

Constraints (6g)-(6k) are the related to the time windows, and are somewhat quite similar to constraint (5e)-(5i) of model (5), only with the addition of the extra variable, y. Constraint (6g) sets the t-variable, and is the same as constraint (5e) explained in section 5.3. Constraint (6h) sets the w-variable by considering the time the previous position is visited, and adding the correct travel time depending on if a customer is visited or not. The term,

$$\left(1 - \sum_{i\in I} x_{irp} - \left(y_{p(r)_-} + y_{p(r)_+}\right)\right) \cdot \bar{t}_{p(r)_-,p(r)_+}$$

ensures that the travelling time from the previous customer to the considered customer only is added if there is not inserted a customer in the previous position and none of the two customers next to the position are removed. If the previous customer is removed then the travelling time for the customer before the previous customer is added to the t_{rp} variable instead. Figure 9 illustrates which $\bar{t}_{i,j}$ are considered in the 4 different cases in eq. (6h). In figure fig. 9, the green diamonds are the positions, the red squares denotes a deleted customer and the blue nodes are customers in the original and update route. The encircled node denotes which w_i is updated. The solid arcs are the route, and the dashed arcs denotes which $\bar{t}_{i,j}$ is added to the t_{rp} variable. When no customer is inserted and no customer is removed then constraint (6h) becomes

$$w_{p(r)_+} \geq t_{rp} + \bar{t}_{p(r)_-,p(r)_+}$$

This situation is shown in fig. 9a. On the other hand, if a customer, $\hat{i}$ is inserted in the position pr then $w_{p(r)_+}$ should satisfy

$$w_{p(r)_+} \geq t_{rp} + \bar{t}_{\hat{i},p(r)_+}$$

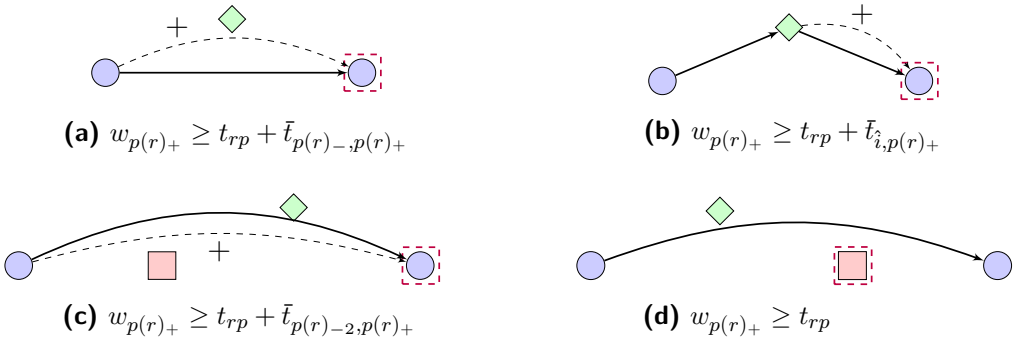
As shown in fig. 9b.
If the customer $p(r)_-$ is removed ($y_{p(r)_-} = 1$), then the time from the customer $p(r)_{-2}$ should instead be considered, and the constraint (6h) becomes

$$w_{p(r)_+} \geq t_{rp} + \bar{t}_{p(r)_{-2},p(r)_+}$$

As shown in fig. 9c. If the customer the time is updated for is removed ($y_{p(r)_+} = 1$), then we do not have to add any time and the constraint becomes

$$w_{p(r)_+} \geq t_{rp}$$

This situation is shown in fig. 9d.



(a) $w_{p(r)_+} \geq t_{rp} + \bar{t}_{p(r)_-,p(r)_+}$ **(b)** $w_{p(r)_+} \geq t_{rp} + \bar{t}_{\hat{i},p(r)_+}$

(c) $w_{p(r)_+} \geq t_{rp} + \bar{t}_{p(r)_{-2},p(r)_+}$ **(d)** $w_{p(r)_+} \geq t_{rp}$

Figure 9.: Illustration of how w_i is updated in eq. (6h).

Constraint (6i) sets the lower- and upper bounds for the t-variables, this constraint is the same as (5g) explained in section 5.3. Constraint (6j) sets the proper time window for the depot in the beginning of the route, and constraint (6k) defines the lower- and upper bound for the rest of the w-variables. If the customer i', is not removed ($y_{i'} = 0$) the constraint becomes

$$a_{i'} \leq w_{i'} \leq b_{i'}$$

However, if the customer is removed constraint (6h) defines a lower bound, and in this case the constraint (6k) should not be binding. If a customer j' is removed the constraint becomes

$$0 \leq w_{j'} \leq b_{k'}$$

where k' is the customer before j'. If j' is removed then there is not inserted a customer just before j'. Therefore $w_{k'} = w_{j'}$ will be feasible as defined by the previous constraints. Therefore the upper bound for the $w_{j'}$ variable is $b_{k'}$ in order to tighten the LP-relaxation.

It is always feasible to keep the solution as it is, and not removing a single customer. Therefore the model cannot be infeasible, and $z = 0$ is an upper bound. The solution the model returns is the optimal solution with respect to the solution that was given as input. Hence if the model outputs an improved solution this new solution could be given as input to the model once more to potentially get an even better solution. Therefore the model can, in theory, be solved several consecutive times and improve the solution.

But when should this model be used? As the model guarantees that the solution is no worse than the input solution it can be used when a new best solution is found. However, in the beginning of the algorithm the quality of the new best solutions are poor therefore it is a good idea to wait until the algorithm have been through at least half of the total number of iterations, as this will fasten the algorithm. Furthermore a parameter θ_2 is assigned. θ_2 describe the probability of using this method in a given iteration, similar to the parameter θ_1 for the exact insertion method. The parameter θ_2 will be tuned in chapter 6.

5.5 SET PARTITIONING INSERTION METHOD

The exact insertion method described in section 5.3 has some limitations. To overcome these limitations in the form of another Mixed Integer Programming model is not straightforward, and will also lead to the model being more computationally hard.
Another idea would be be to generate many diverse routes and choose the best combination of these using a set partitioning model, as presented in 5.1. This is exactly what is done in the Set Partitioning Insertion method (SPInsert).

Consider a partial feasible solution where a set of customers are removed, and needs to be inserted. Furthermore, let R be the set of generated routes. The SPInsert method first adds the routes from the original solution, before the customers were deleted, to the set R. This ensures that the algorithm does not find a worse solution than the current. Hereafter the Regret and Greedy heuristic are used and the routes from these two solutions are added to the set R. After this *NumRounds* new randomised greedy solution are generated, and the routes from these solutions are added to the set R.

Two methods to generate new solutions have been implemented. A random greedy method and a Random k'th best.

- **Random greedy:**
  The Random greedy method selects a customer at random and inserts the customer at the place where it is inserted cheapest possible. After a customer have been added to a route, this new route is added to the set R. This continues until all customers have been inserted. Two consecutive calls of the Random Greedy method will most likely return different solutions because the customers are selected in different order.

- **Random k'th best**
  Just as the Random Greedy method, the Random k'th best method selects a random

customer. The method takes a random parameter $k \in \{1, 2, \ldots, \hat{k}\}$, as input, where $\hat{k}$ is the maximum allowed k. k describes how many insertions places to consider when determining where the customer should be inserted. If $k = 2$ the two best insertion places are found, and the customer is inserted into one of these places at random. If $k = 3$ the three best insertion places are found, and the customer is into one of these places, each chosen with equal probability.
After the customer have been inserted the new route is added to the set R. This continues until all customers have been inserted.

For the Random k'th best method there are three random variables, in each iteration. 1) Which customer to insert. 2) How many insertion places to consider, k. 3) Where to insert the customer. Due to these random variables the probability of inserting a customer at the best place is higher than the probability of the second best place. The following example calculates these probabilities for $\hat{k} = 3$.
The probability of inserting a customer at the t'th best place can be calculated as

$$P(t) = \sum_{i=0}^{\hat{k}-t} \frac{1}{\hat{k}-i} \cdot \frac{1}{\hat{k}}$$

In the case where $\hat{k} = 3$ the probabilities are

$$\begin{aligned} P(1) &= \frac{1}{3} \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{3} + \frac{1}{1} \cdot \frac{1}{3} &&= \frac{11}{18} \\ P(2) &= \frac{1}{3} \cdot \frac{1}{3} + \frac{1}{2} \cdot \frac{1}{3} &&= \frac{5}{18} \\ P(3) &= \frac{1}{3} \cdot \frac{1}{3} &&= \frac{2}{18} \end{aligned}$$

That means, that with $\hat{k} = 3$ a given customer still have approximately 60% probability of being inserted into the most affordable place. So it should still give relative good routes for $\hat{k} = 3$, but also helps generate routes that would not be generated by the Random greedy method.

Every time a new solution is made one of these methods are chosen at random, and in the end all the routes are considered by solving a set partitioning problem.

The SPInsert method takes as input a solution, x, a set of customers to be inserted D, and $\hat{k}$ and returns a new solution x. When updating the set R hashing method are used in order to ensure there are no duplicate routes in R.
As with the two previously described insertion method, θ_3 will denote the probability of using this insertion method in a given iteration.

5.6 SET PARTITIONING DESTROY/INSERTION METHOD

A natural extension of the SPInsert method described in section 5.5, is to take a full feasible solution as input and destroy this solution *NumDestroys* number of times. Then for every restricted solution use the SPInsert method to generate feasible routes with respect to the destroyed solution, and in the end consider all of the routes generated throughout this procedure using a set partitioning model. Doing so, many different routes will be generated which improves the chance

of finding an improved solution. This method can, as with the Exact Destroy/Insertion method, be used when a new best solution is found late in the algorithm. The parameter *NumDestroys* will denote how many times the original solution is destroyed. Furthermore, a parameter, θ_4 is assigned that describes the probability of using the method in a given iteration.

5.7 HEURISTIC BRANCH CUT & PRICE

Throughout the ALNS algorithm a lot of information can easily be gathered and used to further optimise the solutions. Such information includes; good routes and good edges. The set partitioning model exploits the information regarding the routes. In order to exploit the good edges a more sophisticated method must be used.

Branch & Price are used in most, if not all, state-of-the-art exact VRPTW algorithms. Branch & Price is similar to Branch & Bound, the main difference is that the linear relaxations are solved by column generation.

Column generation is a general approach which can be used to solve LP-problems by gradually considering more variables. Column generation considers two different problems, a master problem and a sub problem (or pricing problem). In the master problem only few variables are considered. The sub problem takes as input the dual variables from the master problem, and searches for variables, which can improve the LP-solution, i.e variables with negative reduced costs. These variables are hereafter added to the master problem which finds new dual variables to be used in the next iteration of the sub problem. When the sub problem fails to find a new variable with negative reduced costs the solution from the master problem has been proven optimal. Thus the solution is determined in the master problem, and is proved optimal in the sub problem. However, the solution is not necessarily integer feasible, and therefore you need to branch.

When branching in Branch & Price it is not encouraged to branch on the master problem variables. Doing so the sub problem needs to be extended to forbid the generation of disallowed variables. Rather the branching decisions are imposed on the variables in the sub problem.

The master problem for VRPTW is similar to model (2), as described in the section 5.1, the main difference is that is an LP-relaxation.

$$\texttt{Min Z} = \sum_{p \in \Omega} c_p y_p \tag{7a}$$

`Subject to:`

$$\sum_{p \in \Omega} a_{ip} y_p = 1 \qquad \forall i \in V \setminus \{0, n+1\} \tag{7b}$$

$$0 \leq y_p \leq 1 \qquad \forall p \in \Omega \tag{7c}$$

The objective function (7a) minimises the overall cost, and constraint (7b) makes sure that every customer is visited exactly once. Constraint (7c) defines the variables as linear. In model (7), Ω is the full set of feasible routes. Instead of considering the full set of routes only a subset is considered, and the problem solved is referred to as the *Restricted master problem*.
The set Ω consists of feasible routes, and thus the pricing problem needs to find feasible routes.

The problem is thus to find the shortest path that starts and ends at the depot, satisfy the capacity and time window constraints and contains no cycles. This problem is known as an *Elementary shortest path problem with resource constraints* (ESPPRC).

Let $\alpha_i,\ i \in V \setminus \{0, n+1\}$ be the dual variables associated with constraint (7b) of the master problem, and let $\alpha_0 = 0$. All other parameters are as described in the mathematical model for VRPTW in chapter 2. The variables are similar to the variables described in the same chapter, but without the k index. The sub problem can be stated as seen in (8).

$$\texttt{Min Z} = \sum_{(i,j)\in A} (c_{ij} - \alpha_i) x_{ij} \tag{8a}$$

`Subject to:`

$$\sum_{j \in V} x_{ij} \leq 1 \qquad \forall i \in V \setminus \{0, n+1\} \tag{8b}$$

$$\sum_{j \in V} x_{0,j} = 1 \tag{8c}$$

$$\sum_{i \in V} x_{i,n+1} = 1 \tag{8d}$$

$$\sum_{j \in V} x_{ji} = \sum_{j \in V} x_{ij} \qquad \forall i \in V \setminus \{0, n+1\} \tag{8e}$$

$$w_i + t_{ij} \leq w_j + (1 - x_{ij}) \cdot M \qquad \forall (i,j) \in A \tag{8f}$$

$$a_i \leq w_i \leq b_i \qquad i \in V \tag{8g}$$

$$\sum_{i \in V} d_i \sum_{j \in V} x_{ij} \leq Q \tag{8h}$$

$$x_{ij} \in \{0, 1\} \qquad \forall (i,j) \in A \tag{8i}$$

$$w_i \geq 0 \qquad \forall i \in V \tag{8j}$$

The constraints (8c)-(8j) are similar to the constraints (1c)-(1j) in the standard VRPTW formulation, but without the k index as only a single route is needed. The constraint (8b) imposes the path elementarity constraint, but is not strictly necessary due to the uniqueness of the time variable, and positive travel times. The objective function minimises the reduced cost, and will thus find the route that seems to be able to improve the solution the most at the moment.

Dror [11] proved that the ESPPRC problem is $\mathcal{NP}$ hard, and therefore the problem is most often solved using relaxations or heuristics.

One of the current best exact VRPTW solvers are by Baldacci et al. [5], and is a Branch Cut & Price algorithm where cutting planes are generated to strengthen the linear relaxation. Branch Cut & Price (BCP) is an exact framework, and thus, in theory, ensures optimality given enough time. But the problem here is the time, Baldacci et al. [5] reports an average time of almost 8 hours for the R2 Solomon instances. This is far too long to be used within an heuristic framework. In order to speed up the execution one can consider a reduced graph, and solve the problem to optimality on that graph. However, this does not ensure overall optimality, but can be used as a postoptimisation step in the end of the ALNS algorithm.

The problem then becomes how to reduce the graph, and which edges to consider. This thesis considers the edges used in z best unique obtained solutions. However, the best obtained

solutions are expected to be much alike each other, and thus many of the edges used will be the same. To add more edges, information is extracted from the solution to the LP-relaxation of the set partitioning problem. By solving the LP-relaxation, and extending the reduced graph to include those edges used in the LP-solution will add more edges used in routes that posses some desirable properties with respect to the LP-relaxation. To limit the number of edges added, an upper bound, $\tau \cdot n$, has been set on the number of routes possibly used from the LP-solution. Here τ is a parameter and n is the number of customers in the instance. If there are more routes used in the LP-solution, then the routes are sorted with respect to the y-value in the LP-solution, such that the 'best' routes are added.

S. Ropke's Branch Cut & Price algorithm presented in [28] are used to solve the VRPTW problem on the reduced graphs. A time limit on 600 seconds have been used.

6 TESTING

This section will describe how the parameters of the algorithms have been tuned. Section 6.1 focuses on tuning the parameters of the standard ALNS method. Section 6.2 considers the described exact method, and focuses on whether or not the addition of these actually improves the performance of the algorithm.

The parameters are tuned by testing the algorithm on the following five Solomon instances: `C204`, `R112`, `R208`, `RC103` and `RC206`. These five instances represent a diverse subset of the full set of Solomon instance. One of the clustered instance, two random instances and two random clustered instances are considered, furthermore instances with wide and tight time windows are used. Generally it is not good to tune on too many instances as this will lead to 'over tuning', ensuring good performance on these instances with no knowledge for other instances.
In order to calculate the performance for a parameter value the solutions are compared to the best known solutions by computing the gap. For the Solomon instances the optimal solutions can be found in [26] and [28], in order to compare the results, the same rounding as in the exact methods are made for the costs.
Denote the obtained solution by Z and the best known solution by Z^* the gap is calculated as

$$\frac{Z - Z^*}{Z^*} \cdot 100\%$$

In order to find the best parameter value, the gap for a given setting is calculated for all of the instances. To compare different parameter settings the mean gap, $\bar{x}$, is calculated along with the standard deviation, σ.
The tests are run on DTU's HPC system, with a 2.6 GHz Processor and where the MIP models uses 8 threads. Furthermore, CPLEX version 12.1 is used to solve the MIP models.

6.1 ALNS

The described standard ALNS algorithm has in total 21 parameters - see appendix A. To narrow the parameters testing phase, 14 values have been reused from the literature since they provided good results for similar problems. Many of these values are based on the values used by Ropke and Pisinger in [29], which considers a pickup and delivery problem with time windows. As this problem is in the class of Vehicle Routing Problems, it is safe to assume that the values for those parameters also works well for a VRPTW. Hence only 7 values have been tuned.
Section 6.1.1 shows the result of the tuning phase, and section 6.1.2 test other aspects of the algorithm, e.g. the stochastic local search method mentioned in section 4.2.

6.1.1 Parameter Tuning

The 7 parameters that has been tuned is: The cooling factor, ζ, the start temperature parameter, w, the maximum removal percentage, π, the segmentsize, η, the local search parameter, κ,

resetting parameter, γ, and lastly the noise parameter ψ. For a given parameter value the algorithm is run 20 times with 25000 iterations in order to calculate the mean performance.
To limit the testing, only a single parameter is tuned at a time, for the rest of the parameters, initial guesses on good parameter values have been made by looking in the literature. This is of course not optimal, but this was done in order to minimise the time spent tuning the algorithm, and be able to test more values for the individual parameters.

Cooling Factor - ζ

The cooling factor defines, together with the start temperature, the temperature range of the algorithm. Ropke and Pisinger [29] use $\zeta = 0.99975$. Inspired by this the following values have been tested,

$$\zeta = \{\, 0.9995,\ 0.99975,\ 0.9999,\ 0.999925,\ 0.999950,\ 0.999975 \}$$

The results can be seen in table 3.

	Overall		C204		R112		R208		RC103		RC206	
ζ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0.9995	1.20%	1.30	0.25%	1.08	1.63%	0.62	0.49%	0.55	2.55%	1.63	1.08%	0.70
0.99975	0.95%	1.06	0.50%	1.48	1.08%	0.57	0.56%	0.61	**1.53%**	1.23	1.11%	0.69
0.9999	**0.72%**	**0.86**	**0.00%**	**0.00**	**0.78%**	0.48	0.24%	0.29	1.70%	1.14	0.90%	0.61
0.999925	0.96%	1.36	0.53%	2.30	1.06%	0.55	0.26%	0.29	2.22%	1.01	**0.73%**	0.48
0.99995	1.18%	1.25	**0.00%**	**0.00**	1.40%	0.60	0.34%	0.44	3.21%	**0.85**	0.96%	**0.41**
0.999975	1.60%	1.78	**0.00%**	**0.00**	2.23%	**0.42**	**0.20%**	**0.07**	4.61%	1.00	0.96%	0.48

Table 3.: Performance of the algorithm for different values of the parameter ζ. $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 3 shows that $\zeta = 0.9999$ gives the overall best performance, and also gives good results on each of the five test instances.

Start Temperature Parameter - w

Ropke and Pisinger [29] defines the start temperature the same way as in this thesis, and the parameter tuning shows that $w = 0.05$ performs best for the considered pickup and delivery problem. Inspired by this the algorithm is tested with the following values.

$$w = \{\, 0.005,\ 0.010,\ 0.015,\ 0.25,\ 0.05,\ 0.075,\ 0.1 \}$$

The result can be seen in table 4.
Table 4 shows that $w = 0.015$ works best, it gives the best overall performance, and standard deviation for the considered test instances.

Maximum Removal Percentage - π

For the maximum removal percentage parameter π, the following values have been tested

$$\pi = \{\, 20\%,\ 25\%,\ 30\%,\ 35\%,\ 40\%\ \}$$

The results can be seen in table 5

	Overall		C204		R112		R208		RC103		RC206	
w	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0.005	0.96%	1.28	0.99%	1.98	1.18%	0.63	0.38%	0.39	1.70%	1.57	**0.53%**	0.45
0.01	0.75%	1.05	0.25%	1.08	0.85%	0.49	0.41%	0.51	1.51%	1.52	0.75%	0.75
0.015	**0.71%**	**0.73**	**0.00%**	**0.00**	0.98%	**0.35**	0.54%	0.63	**1.37%**	**0.92**	0.66%	0.50
0.025	0.76%	0.93	0.25%	1.08	**0.83%**	0.69	0.25%	0.28	1.51%	1.07	0.97%	0.50
0.05	1.06%	1.50	**0.00%**	**0.00**	1.36%	0.66	**0.19%**	**0.08**	2.89%	1.30	0.85%	1.54
0.075	1.29%	1.60	**0.00%**	**0.00**	1.56%	0.68	0.22%	0.07	3.72%	1.22	0.94%	1.58
0.1	1.54%	1.70	**0.00%**	**0.00**	2.01%	0.40	0.23%	0.04	4.36%	1.34	1.12%	**0.36**

Table 4.: Performance of the algorithm for different values of the parameter w. $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

	Overall			C204		R112		R208		RC103		RC206	
π	$\bar{t}$	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
20%	**28.85**	1.68%	1.66	2.07%	2.78	1.46%	0.53	0.77%	0.69	2.61%	1.68	1.51%	0.68
25%	34.39	1.53%	1.57	2.72%	2.46	1.21%	0.68	0.33%	0.31	2.27%	1.32	1.12%	0.54
30%	41.24	0.96%	1.17	0.74%	1.77	1.07%	0.57	0.30%	0.38	1.80%	1.28	0.91%	0.64
35%	47.37	**0.72%**	**0.77**	**0.00%**	**0.00**	**0.86%**	0.54	0.23%	**0.28**	**1.70%**	**0.64**	**0.81%**	0.66
40%	53.52	0.72%	0.85	**0.00%**	**0.00**	0.88%	0.56	**0.20%**	0.29	1.81%	0.95	**0.73%**	**0.54**

Table 5.: Performance of the algorithm for different values of π. $\bar{t}$ denotes the average time in seconds, $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 5 shows that low values for π is not as good as higher values. Moreover the algorithm is faster the less customers that are removed, which intuitively makes sense. The average gap for $\pi = 35\%$ and $\pi = 40\%$ is almost the same, but $\pi = 35\%$ is faster. Therefore $\pi = 35\%$ will be used in the algorithm.

Segmentsize - η

Ropke and Pisinger uses in [29] a segmentsize on $\eta = 100$ but there is not given any explanation of why this is chosen.
The following values have been tested.

$$\eta = \{25, 50, 75, 100, 125, 150\}$$

The results can be seen in table 6.

	Overall		C204		R112		R208		RC103		RC206	
η	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
25	0.67%	0.79	**0.00%**	**0.00**	0.90%	0.56	**0.15%**	**0.09**	1.37%	0.96	0.93%	0.73
50	0.70%	0.89	**0.00%**	**0.00**	**0.68%**	0.61	0.21%	0.29	1.70%	1.15	0.93%	0.61
75	0.66%	0.77	**0.00%**	**0.00**	0.89%	0.55	0.28%	0.39	1.46%	1.03	0.68%	0.46
100	0.67%	0.72	**0.00%**	**0.00**	0.82%	0.52	0.45%	0.56	1.32%	0.86	0.77%	**0.59**
125	**0.59%**	**0.67**	**0.00%**	**0.00**	0.83%	0.52	0.33%	0.39	**1.18%**	**0.81**	**0.63%**	0.61
150	0.62%	0.75	**0.00%**	**0.00**	0.80%	**0.46**	0.32%	0.40	1.25%	1.08	0.75%	0.58

Table 6.: Performance of the algorithm for different values of η. $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 6 shows that $\eta = 125$ produces the best result for these chosen five test instances. $\eta = 125$

will therefore be used as the segmentsize. However, the differences are quite small, and might as well be causes by the random nature of the algorithm.

Local Search Parameter - κ

κ defines how good a solution should be before trying to further optimise it using local search. κ have been tested by only using the relocate neighbourhood as local search method, and the following values have been tested

$$\kappa = \{-\infty, 0\%, 1\%, 2\%, 3\%, 4, 5\%, 6\%, 7\%, 8\%, 9\%, 10\%, \infty\}$$

$-\infty$ means that the local search procedure is not used at all, and ∞ means that it is used in every iteration.
It is expected that a bigger value for κ results in better solutions, but comes at a cost, namely the time spent, therefore we cannot only focus on the quality of the solutions, but will also need to look at the extra time it takes to perform the local search. The performance table can be seen in table 7, which also contains information on the overall time spent.

	Overall			C204		R112		R208		RC103		RC206	
κ	$\bar{t}$	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
$-\infty$	**36.61**	0.94%	1.22	**0.00%**	**0.00**	1.02%	0.60	0.28%	0.44	2.62%	1.50	0.78%	0.70
0%	36.68	1.23%	1.46	0.25%	1.08	0.79%	0.50	0.32%	0.50	3.38%	1.31	1.40%	0.83
1%	37.58	1.03%	1.19	**0.00%**	**0.00**	1.21%	0.66	0.36%	0.59	2.62%	1.29	0.95%	0.71
2%	39.04	0.88%	1.01	**0.00%**	**0.00**	0.98%	0.58	0.27%	0.45	1.97%	1.24	1.20%	0.76
3%	40.41	0.75%	0.79	**0.00%**	**0.00**	0.96%	0.59	0.26%	0.36	1.50%	0.87	1.02%	0.79
4%	41.75	0.66%	0.79	**0.00%**	**0.00**	**0.73%**	**0.37**	0.28%	0.39	1.37%	1.04	0.90%	0.79
5%	42.80	0.67%	0.79	**0.00%**	**0.00**	**0.73%**	0.42	0.36%	0.45	1.48%	0.92	0.77%	0.79
6%	45.87	0.72%	0.81	0.25%	1.08	0.97%	0.50	0.30%	0.38	1.17%	0.71	0.93%	0.70
7%	48.57	0.78%	0.88	**0.00%**	**0.00**	0.91%	0.62	0.27%	0.39	1.65%	1.15	1.06%	0.56
8%	49.37	0.66%	0.86	**0.00%**	**0.00**	0.82%	0.61	0.15%	0.08	1.57%	1.12	0.75%	0.68
9%	50.89	0.67%	0.78	**0.00%**	**0.00**	0.86%	0.50	**0.14%**	0.09	1.50%	1.05	0.84%	**0.46**
10%	53.37	0.73%	0.72	**0.00%**	**0.00**	1.06%	0.59	0.17%	0.17	1.55%	0.99	0.87%	0.61
∞	86.48	**0.55%**	**0.61**	**0.00%**	**0.00**	1.20%	0.69	0.16%	**0.08**	**0.77%**	**0.50**	**0.60%**	0.48

Table 7.: Performance of the algorithm for different values of κ. $\bar{t}$ denotes the average time in seconds, $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 7 shows that $\kappa = \infty$ seems to be the best, both looking at the average gap and the standard deviation, and $\kappa = 4\%$ is the second best. However $\kappa = 4\%$ is considerably faster, and $\kappa = 4\%$ more than halves the computation time compared with $\kappa = \infty$. The small performance increase do not justify the increase in the time spent, and therefore $\kappa = 4\%$ will be used.
The expectation that a bigger κ value gives better solutions seems to be incorrect. This could be explained by the relative few number of tests on each instance, which means the randomness of the algorithm have a higher impact. It could also be explained by the algorithm getting stuck in a good local optimal solution far into the execution and is then unable to get away from it again.

Resetting Parameter - γ

For the Resetting parameter, γ the following values have been tested.

$$\gamma = \{2500, 5000, 7500, 10000, 12500, 15000, 17500, 20000, \infty\}$$

$\gamma = \infty$ have been tested to see if the addition of the resetting mechanism actually improves the solutions. The results is seen in table 8.

	Overall		C204		R112		R208		RC103		RC206	
γ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
2500	0.62%	0.74	**0.00%**	**0.00**	0.76%	0.51	0.45%	0.65	1.21%	1.01	**0.69%**	**0.51**
5000	**0.57%**	**0.61**	**0.00%**	**0.00**	**0.61%**	**0.40**	0.29%	0.33	**0.98%**	**0.72**	0.98%	0.59
7500	0.77%	0.94	**0.00%**	**0.00**	0.78%	0.60	0.38%	0.53	1.85%	1.21	0.83%	0.63
10000	0.73%	0.79	**0.00%**	**0.00**	0.87%	0.50	0.29%	0.46	1.50%	0.88	0.97%	0.69
12500	0.81%	0.97	**0.00%**	**0.00**	0.73%	0.40	**0.21%**	**0.29**	2.13%	1.06	0.96%	0.71
15000	0.82%	0.89	**0.00%**	**0.00**	0.71%	0.54	0.52%	0.55	1.80%	1.10	1.08%	0.88
17500	0.72%	0.85	**0.00%**	**0.00**	1.08%	0.59	0.38%	0.45	1.37%	1.17	0.76%	0.70
20000	0.81%	0.98	0.01%	0.02	0.76%	0.48	0.37%	0.51	1.98%	1.25	0.94%	0.70
∞	0.78%	0.83	**0.00%**	**0.00**	0.81%	0.62	0.30%	0.35	1.61%	0.94	1.17%	0.64

Table 8.: Performance of the algorithm for different values of γ. $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 8 shows that $\gamma = 5000$ is best as it gives the best average gap, and lowest standard deviation. Comparing the results for $\gamma = \infty$ with $\gamma = 5000$ we see that the addition of the resetting mechanism is worthwhile as it improves the average gap from 0.78% to 0.57%.
The stopping criteria is 25000 iterations, this means that 20% of the algorithm is allowed to run without any improvements before resetting.

Noise Parameter - ψ

For the noise parameter, ψ the following values have been tested.

$$\psi = \{0\%, 1\%, 2\%, 3\%, 4\%, 5\%\}$$

Intuitively small values for ψ should give the best result as this gives most focus to the best insertion, but still helps diversify the search. The results is shown in table 9.

	Overall		C204		R112		R208		RC103		RC206	
ψ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0%	0,98%	1,15	0,27%	0,82	0,69%	0,55	0,42%	**0,00**	2,58%	1,13	0,93%	0,80
1%	**0,93%**	**0,97**	0,07%	0,05	0,98%	0,61	0,67%	**0,00**	**2,00%**	1,19	0,92%	0,76
2%	1,00%	1,05	0,06%	0,05	1,10%	**0,51**	0,36%	**0,00**	2,18%	**1,16**	1,32%	0,99
3%	0,96%	1,12	0,06%	0,05	**0,86%**	0,55	**0,33%**	**0,00**	2,53%	1,20	1,03%	0,81
4%	0,95%	1,13	**0,00%**	**0,00**	1,03%	0,59	0,55%	**0,00**	2,32%	1,30	**0,87%**	0,92
5%	1,01%	1,13	**0,00%**	**0,00**	0,90%	0,58	0,50%	**0,00**	2,48%	1,36	1,19%	**0,67**

Table 9.: Performance of the algorithm for different values of ψ. $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 9 shows that $\psi = 1\%$ gives the best result in this test. However, the differences are rather small, and could as well be explained by the random nature of the algorithm. Also looking at the performance for $\psi = 0\%$ we see that the performance for C204 is quite bad compared to the rest parameter values, this is most likely just an unlucky execution. As the potential gain is small, $\psi = 0\%$ is selected in the algorithm as this also help keep the algorithm simple.

6.1.2 Algorithm Tests

Stochastic Relocate Local Search Test

As described in section 4.2, two versions of the relocate neighbourhood have been implemented, a deterministic and a stochastic. The stochastic should help diversify the search, and find other solutions than the deterministic relocate local search.

Tests shows that the deterministic method produces better results than by using the stochastic method. In order to try to explain why this is, a test were set up. In the test the standard criteria for using the local search method were used, $\kappa = 4\%$, and the deterministic local search method were the only local search method used to improve the solution. The stochastic method were used on the same solution as the deterministic one, but only to measure the improvement, and not change the actual solution, i.e to compare the two methods. This was done for every non local optimal solution. The test were executed once for all of the 56 Solomon instances, which result in an approximate of a quarter million executions of the local search method.
Doing this the improvement of the stochastic method can be compared with the improvement for the deterministic method. Formally, let x be a solution that is not local optimal with respect to the relocate neighbourhood. Furthermore, let GI_D be the improvement in gap using the deterministic relocate local search method on x, and GI_S defined similarly, but instead for the stochastic local search method. Figure 10 is a histogram of the values $v = GI_D - GI_S$, that is negative values means that the stochastic method is better than the deterministic. The histogram is grouped such that observation where $-0.25 < v \leq 0$, belongs to the group 0. The group -1.5 however contains all the observations less than -1.5, and 2.75 contains all the observations greater than 2.75.

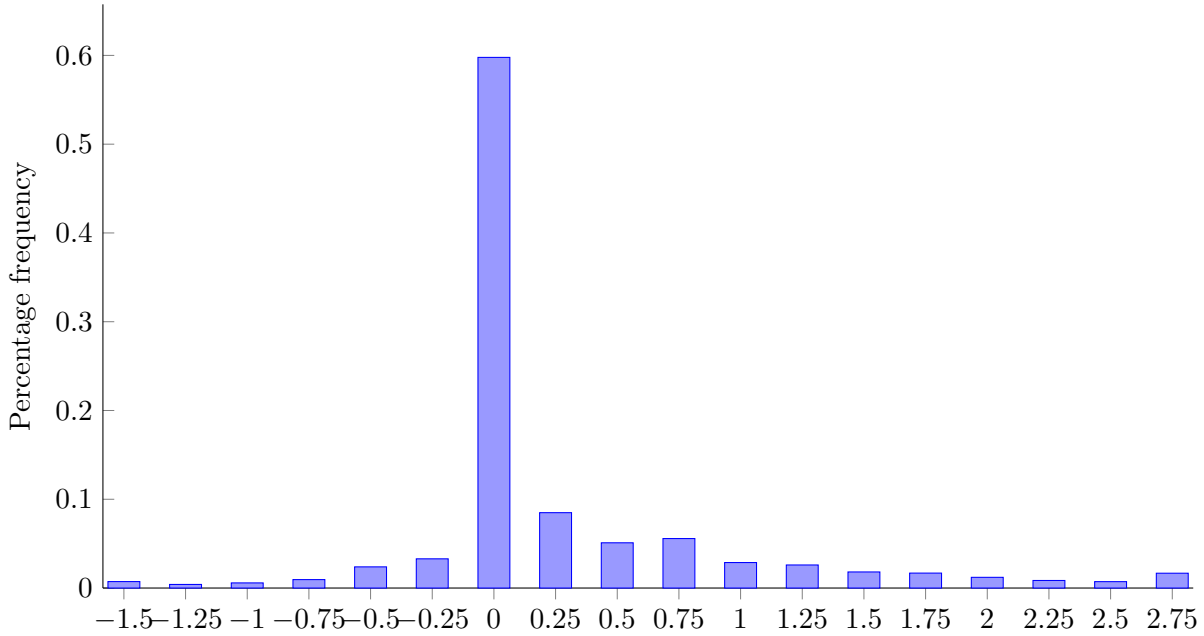


Figure 10.: Comparison between stochastic local search and deterministic local search. The placement of the bars represent the difference in gap improved by the local search methods, a negative value means that the stochastic local search performs better. The height of bars represent percentage of the frequency.

Figure 10 clearly suggest that the deterministic method is better than the stochastic. The figure shows that $-0.25 < v \leq 0$ in almost 60% of the observations - further data analysis in fact shows that v in 55% of the cases is numerical 0. The figure also shows that there are far more observations where the difference is positive, which means that the deterministic performs better. In only 12% of the observations are the stochastic method better than the deterministic method, whereas the deterministic method is strictly better than the stochastic method in 33% of the observations.

The minimum v is -9.86, and the maximum is 4.25, the average however is 0.20 which also suggest that the deterministic method gives the best results.

That the deterministic method gives the best result was expected. The only aspect not covered in fig. 10 is in to what extent the stochastic method diversifies the search. Or almost equivalently, how many times the deterministic local search method returns the same solution. This was checked in the same test, by using hashing methods to determine the number of unique solutions the deterministic local search method returns, and comparing with the total number of local search calls. The data shows that 14% of the calls to the deterministic local search method returns a solution that has already been found by the local search method.

Local Search Tests

The 6 different local search methods described in section 4.2.3 have been tested by using each of the methods as the single local search method, and then running the 5 test instance 20 times. Table 10 shows the results, where 'No-Local' means that no local search method have been used. Table 10 looks at the time, as well as the quality of the solutions.

	Overall			C204		R112		R208		RC103		RC206	
ψ	$\bar{t}$	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
No-Local	**66.26**	0.94%	1.05	0.09%	0.04	0.83%	0.59	0.54%	0.60	2.15%	1.27	1.09%	0.96
Reloc	69.24	0.72%	0.93	0.09%	0.04	0.60%	**0.52**	0.36%	0.55	1.80%	1.32	0.75%	0.60
3Opt*	235.41	0.92%	1.14	**0.00%**	**0.00**	1.25%	0.64	0.48%	0.59	2.28%	1.53	0.58%	0.63
Adapt	152.41	0.72%	0.89	**0.00%**	**0.00**	0.87%	0.53	0.24%	0.42	1.64%	1.21	0.86%	0.67
Re&3O	366.45	0.57%	0.79	**0.00%**	**0.00**	0.76%	0.65	0.32%	0.55	1.27%	1.11	0.48%	0.56
Ra-1o	177.50	0.58%	0.86	**0.00%**	**0.00**	0.78%	0.58	0.21%	0.32	1.49%	1.31	**0.43%**	**0.49**
Ra-2o	283.20	**0.46%**	**0.62**	**0.00%**	**0.00**	**0.59%**	0.57	**0.11%**	**0.09**	**1.05%**	**0.72**	0.54%	0.61

Table 10.: Performance of the algorithm for using different local search methods. $\bar{x}$ denotes the average gap, $\bar{t}$ is the time spent in seconds and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 10 clearly shows that the Ra-2o method is the best; it gives the best average solution, and smallest standard deviation. However the time spent is quite large, and almost 4 times as great as the time when only using the relocate neighbourhood, which also provides good results.
The Ra-1o method gives quite good results in a time that is 2.5 greater than the time compared to only using the relocate neighbourhood.
This suggest the following two approaches.

1. **ALNS relocate**
   Use the Relocate neighbourhood as the only local search method. The test shows that this should provide good results in reasonable time.

2. **ALNS random optimal**
   Use Ra-1o method when the solution is within the threshold, κ, of the best achieved solution, and then use the Ra-2o method when a new global solution is found, in order to ensure it is local optimal for both of the neighbourhoods. Doing so will result in a local search method that is stochastic. This is quite contradictory comparing with the comparison between the deterministic and stochastic relocate methods. The difference here is that the random variable only determines which neighbourhood to consider, and not which customer to insert as was the case with the stochastic relocate method. This local search method chooses deterministic between two stochastic methods. This should provide good results, but the time is expected to be approximately 3 times larger than the ALNS relocate method.

Both of these methods will be tested further in chapter 7, to see the performance for the Solomon instances. Only the relocate local search methods have been used in the following tests.

6.2 EXACT METHODS

This section focuses on testing the exact methods described in chapter 5.
Each of the methods have been tested by solving the previously introduced 5 test 20 times. The following parameters have been tested; The set partitioning parameter, ρ, the use of the route optimiser and the probabilities of using the 4 insertion methods, θ_1, θ_2, θ_3 and θ_4. The parameters $\hat{k}$, *Numrounds* and *NumDestroys* for the SPInsert/Destroy method as well as z and τ for the Branch Cut and Price heuristic have been set by ad hoc testing, due to missing time.

$$\hat{k} = 3, \qquad Numrounds = 50, \qquad NumDestroys = 10, \qquad z = 10, \qquad \tau = 0.50$$

6.2.1 Set Partitioning - ρ

For the set partitioning parameter, ρ the following values have been tested.

$$\rho = \{2500,\ 5000,\ 7500,\ 10000,\ 12500,\ 15000,\ 25000,\ \infty\}$$

In all the cases is the set partitioning problem solved in the end after the 25000 iterations, except $\rho = \infty$ where the set partitioning problem is not solved at all.
Table 11 shows the results from the test.

Table 11 shows that it actually is a good idea to solve the set partitioning problem throughout the algorithm, and the performance is best when $\rho = 2500$, achieving an average solution just 0.21% from the optimal, comparing with 0.93% when the set partitioning is not used. However the average execution time is almost doubled, but as the performance is that much better it seems worth the extra time. Intuitively smaller values for ρ should also be better as the set partitioning problem is solved more often, however solving more often than every 2500'th iteration seems too often, and therefore $\rho = 2500$ is chosen in the algorithm.

6.2.2 TSPTW

As described in chapter 5 the route optimiser cannot be tested without the addition of the set partitioning model, as this is the only way to include the optimised routes in the solution.

	Overall			C204		R112		R208		RC103		RC206	
ρ	$\bar{t}$	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
2500	134.17	**0.21%**	**0.37**	**0.00%**	**0.00**	**0.19%**	**0.15**	**0.19%**	**0.30**	0.04%	0.16	0.62%	0.58
5000	127.45	0.25%	0.41	**0.00%**	**0.00**	0.39%	0.32	0.19%	0.33	**0.00%**	**0.01**	0.69%	0.57
7500	138.34	0.33%	0.66	**0.00%**	**0.00**	0.33%	0.27	0.21%	0.43	0.22%	0.96	0.90%	0.76
10000	92.18	0.32%	0.59	**0.00%**	**0.00**	0.44%	0.32	0.29%	0.54	0.06%	0.22	0.82%	0.97
12500	98.18	0.26%	0.44	**0.00%**	**0.00**	0.31%	0.27	0.21%	0.43	0.08%	0.25	0.71%	0.59
15000	113.60	0.30%	0.64	**0.00%**	**0.00**	0.35%	0.31	0.24%	0.52	0.22%	0.95	0.69%	0.76
25000	88.43	0.39%	0.60	**0.00%**	**0.00**	0.61%	0.42	0.39%	0.62	0.39%	0.90	**0.54%**	**0.51**
∞	**69.69**	0.93%	1.45	0.53%	2.39	0.62%	0.48	0.36%	0.56	2.02%	1.45	1.12%	0.86

Table 11.: Performance of the algorithm for different values of ρ. $\bar{t}$ denotes the average time in seconds, $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 12 shows the comparison between only using the set partitioning Problem (with $\rho = 2500$) and then also using the TSPTW model to optimise the routes. As previously described a time-limit on 10 s has been set.
Preliminary tests showed little to no improvement, therefore it was decided to do a full scale tests to see if the TSPTW model had an impact, and thus all 56 Solomon instances are solved 10 times.

		Best Solution		Average Solution		Optimals	
	$\bar{t}$	$\bar{x}$	$\max x$	$\bar{x}$	$\max x$	Total	Instances
SPP	101.15	0.03%	0.38%	0.18%	1.09%	331	42
TSPTW	161.75	0.04%	0.54%	0.17%	1.29%	331	42

Table 12.: Performance of the algorithm with and without the Route Optimiser. $\bar{t}$ is the average time used for the algorithm, $\bar{x}$ denotes the average gap and $\max x$ is the worst achieved solution both when considering the best solution, or the average solution. The Optimal columns denotes how many optimal solutions were found, and for how many instances the algorithm found an optimal solution.

Table 12 shows that the addition of the route optimiser does not have an effect. The difference in the gap can be explained by the random nature of the algorithm, and the time is 60% higher than without.
Looking into the data shows that out of all the routes in the best found solution only 3.8% of these were found using the route optimiser. Furthermore only 2.1% of the calls to the TSPTW model actually resulted in a better route. In 2.5% of the calls to the model, the model stopped without finding an optimal solution, and was stopped due to the timelimit. These 2.5% of the calls, corresponds to 53% of the time used, and excluding these the model found an optimal solution using on average 0.23 seconds.

Baldacci et al. describes in [6], an exact method for the TSPTW problem that outperforms all other exact methods. However, it is not worthwhile to implement faster algorithm in this case as the slow MIP-procedure does not improve the quality of the solutions.

6.2.3 Exact Insertion Method - θ_1

The parameter θ_1 describes the probability of using the Exact Insertion method in a given iteration.
No time limit have been set, hence the model returns the optimal solution. The following values have been tested.

$$\theta_1 = \{0\%, 1\%, 2\%, 3\%, 4\%, 5\%\}$$

Furthermore, the Exact Insertion method have also been included in the adaptiveness of the algorithm. For this test the number of iterations were lowered to 5000 and the cooling factor, ζ, were adjusted such that the algorithm went through the same temperature range. Table 13 shows the performance for these tests. The table reports the performance and the average time for the test.

	Overall			C204		R112		R208		RC103		RC206	
θ_1	$\bar{t}$	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0%	**65.66**	0.65%	0.82	**0.00%**	**0.00**	0.77%	0.61	**0.15%**	**0.31**	1.28%	1.12	1.05%	0.70
1%	73.69	**0.60%**	**0.67**	0.06%	0.05	0.74%	0.63	0.20%	0.36	**1.14%**	**0.62**	0.84%	0.75
2%	105.27	0.70%	0.74	0.06%	0.05	0.85%	0.59	0.30%	0.51	1.39%	0.67	0.91%	0.80
3%	119.57	0.73%	0.88	0.09%	0.04	0.69%	**0.40**	0.42%	0.62	1.68%	1.30	**0.75%**	**0.54**
4%	130.03	0.72%	0.80	0.07%	0.05	**0.69%**	0.51	0.27%	0.45	1.21%	0.92	1.36%	0.83
5%	150.35	0.68%	0.70	0.21%	0.65	0.76%	0.51	0.31%	0.49	1.21%	0.66	0.90%	0.69
Adapt	125.16	1.75%	1.73	**0.00%**	**0.00**	1.84%	0.73	1.20%	0.88	4.26%	1.73	1.43%	0.96

Table 13.: Performance of the algorithm for different values of θ_1. $\bar{t}$ denotes the average time in seconds, $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 13 shows that $\theta_1 = 1\%$ performs the best. However the difference is quite small, and could be explained by the relative small number of times an instance is solved. The algorithm for $\theta_1 = 0\%$ is the same as the algorithm used when the set partitioning parameter, ρ, was ∞ (results shown in table 11). Even though the algorithm is the same, the quality of the solutions differs a lot. This is due to the random nature of the algorithm and too few number of times an instance is solved. As there is not a clear improvement, and to keep the algorithm simple $\theta_1 = 0\%$ is chosen.
When looking at the performance of adaptive test it is clear that this is quite bad. Whether or not the bad performance is due to the few number of iterations or the that insertion method is bad is unclear. The underlying data, however, shows that the exact insertion method was only chosen 19% of the times, and it is thus being outperformed by at least one of the heuristic method. This point in the direction that the insertion method actually is bad, even when time is not considered.
As previously described the algorithm finds an optimal solution using on average 0.05 seconds.

6.2.4 Exact Destroy/Insertion Method - θ_2

The parameter θ_2 describes the probability of using the Exact Destroy/Insertion method in a given iteration.
Similar to the Exact Insertion method, no time limit was set, and the model always returns

the optimal solution. As the model is more complex, and expected to be slower than the Exact Insertion method smaller values for θ_2 have been tested

$$\theta_2 = \{0\%, 0.2\%, 0.4\%, 0.6\%, 0.8\%, 1\%\}$$

Furthermore, the method have also been included in the adaptivity of the algorithm. For this test the number of iterations were changed to 5000, and the cooling effect changed such that the algorithm went through the same temperature range. The method have also been applied solely as an improvement heuristic; trying to improve the new best solutions found after 20000 iterations, this test is denoted by 'Best' in table 14, that shows the result.

	Overall			C204		R112		R208		RC103		RC206	
θ_2	$\bar{t}$	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0%	**43.45**	0.81%	0.99	**0.00%**	**0.00**	0.95%	0.67	0.43%	0.62	1.69%	1.42	1.00%	0.74
0.2%	92.86	0.68%	0.73	**0.00%**	**0.00**	0.93%	0.53	0.25%	0.48	1.40%	**0.77**	0.85%	0.59
0.4%	144.84	0.63%	0.78	**0.00%**	**0.00**	0.74%	0.56	0.26%	0.47	1.36%	0.93	**0.80%**	0.78
0.6%	201.36	0.78%	0.90	**0.00%**	**0.00**	0.88%	0.56	0.32%	0.54	1.69%	1.10	1.02%	0.78
0.8%	238.56	**0.62%**	0.81	**0.00%**	**0.00**	0.80%	0.55	0.23%	0.42	**1.26%**	1.26	0.83%	**0.56**
1%	293.75	0.75%	1.28	**0.00%**	**0.00**	0.86%	0.48	0.42%	0.60	1.55%	1.19	0.90%	1.40
Adapt	775.32	1.60%	**1.49**	0.25%	1.11	1.87%	0.57	0.66%	0.63	3.38%	1.33	1.83%	1.24
Best	69.08	0.67%	0.88	**0.00%**	**0.00**	**0.63%**	**0.45**	**0.23%**	**0.37**	1.34%	1.33	1.16%	0.71

Table 14.: Performance of the algorithm for different values of θ_2. $\bar{t}$ denotes the average time in seconds, $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

In the adaptivity test the method is considered as a destroy heuristic, and thus 'competes' with the 5 other destroy heuristic. As can be seen from table 14, the adaptivity does not gives good results. The time is almost 20 times as high with only one fifth of the iterations. The exact Destroy/insertion method is only used 14% of the times, on average, which suggests that the method does not perform as good as expected.

Comparing the result when using the method as solely as an improvement heuristic and the configuration $\theta_2 = 0\%$, it seems that the improvement heuristic improves the performance a lot. But looking into the data, it is revealed that the method only manages to improve the solution in 7 out of 177 tries, and hence the difference in performance must be due to the random nature of the algorithm.

Even when using the method only with 0.2% probability the overall time almost doubles, and there is not any significant difference in the quality of the solutions, therefore $\theta_2 = 0\%$ is chosen in the final algorithm, thus the method is not used.
Throughout this test the model was in total used 165000 times, and the average time was 1.04 seconds.

6.2.5 Set Partitioning Insert - θ_3

The following two parameters for the set partitioning insert method were set by ad hoc testing,

$$\hat{k} = 3, \qquad Numrounds = 50,$$

The probability parameter for the set partitioning insert method have been tested for the following values.

$$\theta_3 = \{0\%, 1\%, 2\%, 3\%, 4\%, 5\%\}$$

Furthermore the method have been included in the adaptiveness of a algorithm with fewer iterations, just as in the previous two sections.
Table 15 shows the results, where the performance, as well as the time is reported.

	Overall			C204		R112		R208		RC103		RC206	
θ_3	$\bar{t}$	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0%	**70.16**	0.91%	0.99	**0.00%**	**0.00**	0.96%	0.64	0.45%	0.58	1.86%	1.31	1.25%	0.71
1%	82.41	0.73%	0.82	**0.00%**	**0.00**	0.80%	0.54	0.34%	**0.46**	1.51%	1.09	0.98%	0.60
2%	97.84	0.78%	0.85	**0.00%**	**0.00**	0.82%	0.58	0.37%	0.56	1.70%	1.02	1.01%	**0.54**
3%	111.26	0.78%	0.94	**0.00%**	**0.00**	0.99%	0.59	**0.31%**	0.49	1.80%	1.23	0.82%	0.69
4%	126.65	0.70%	**0.71**	**0.00%**	**0.00**	0.99%	**0.52**	0.42%	0.60	**1.32%**	**0.63**	0.79%	0.71
5%	146.96	**0.64%**	1.25	**0.00%**	**0.00**	**0.75%**	0.58	0.41%	0.57	1.59%	1.21	**0.44%**	1.37
Adapt	163.12	1.52%	1.42	**0.00%**	**0.00**	2.22%	**0.37**	1.09%	0.69	3.13%	1.79	1.17%	0.80

Table 15.: Performance of the algorithm for different values of θ_3. $\bar{t}$ denotes the average time in seconds, $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

Table 15 shows that $\theta_3 = 5\%$ gives the best results. However, the small increase in the quality of the solutions do not justify the large increase in the time spent compared to $\theta_3 = 1\%$. $\theta_3 = 1\%$ gives good results with little added time.
The adaptive test does not provide good results, but this is most likely due to the few iterations used. The underlying data shows that the SPInsert method were used 51% of the times, which means it must have outperformed both of the other heuristics. Inspired by this, $\theta_3 = 1\%$ is chosen in the final algorithm as this should add the smallest amount of time, but still the method is used, as the adaptive test showed promising results.

6.2.6 Set Partitioning Destroy Insert - θ_4

The parameter, *NumDestroys* for the set partitioning destroy insert method were, as previously described set to 10.
θ_4 describes the probability of using the set partitioning destroy insert method. This method is quite slow, and therefore only small values for θ_4 have been tested.

$$\theta_4 = \{0\%, 0.2\%, 0.4\%, 0.6\%, 0.8\%, 1\%\}$$

As with the rest of the methods it has also been included in the adaptivity (labelled as a destroy heuristic), in a test with 5000 iterations. Similar as with the Exact Destroy/Insert method it has also been used solely as an improvement heuristic, when finding a new best solution after 20000 iterations. Table 16 shows the result. Out of the 8 tests the one not using the method performs best. The adaptivity test is approximately 10 times as slow using only one fifth of the iterations. However the method is on average being used in 49% of the iterations. This is quite high considering it 'competes' with 5 other destroy heuristics. In the 'Best' test the method is used 197 times, and only manages to find a new best solution in 12 of these cases.
The quality of the solutions from the rest of the tests are very much alike the quality of solutions for $\theta_4 = 0\%$, which might be due to that the method is not used often enough. However, the times reported shows that the method increases the overall time considerably. $\theta_4 = 0\%$ is chosen in the final algorithm, hence the method is not used.

	Overall			C204		R112		R208		RC103		RC206	
θ_2	$\bar{t}$	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
0%	**44.11**	**0.65%**	0.82	**0.00%**	**0.00**	0.85%	0.65	0.30%	0.47	1.24%	1.24	0.85%	**0.51**
0.2%	66.75	0.70%	0.96	**0.00%**	**0.00**	0.83%	0.64	**0.16%**	0.31	1.59%	1.33	0.91%	0.87
0.4%	89.08	0.69%	0.86	**0.00%**	**0.00**	1.01%	0.63	0.17%	**0.31**	**1.22%**	1.30	1.04%	0.59
0.6%	106.72	0.76%	0.98	**0.00%**	**0.00**	**0.61%**	0.49	0.21%	0.46	1.85%	1.10	1.11%	0.99
0.8%	134.09	0.70%	0.92	**0.00%**	**0.00**	0.91%	0.64	0.36%	0.58	1.61%	1.27	**0.64%**	0.72
1%	156.31	0.67%	1.15	**0.00%**	**0.00**	0.70%	0.53	0.29%	0.50	1.28%	1.28	1.08%	1.21
Adapt	478.54	1.39%	1.26	**0.00%**	**0.00**	1.59%	**0.45**	1.11%	0.70	3.01%	1.47	1.24%	0.71
Best	66.90	0.66%	**0.69**	**0.00%**	**0.00**	0.84%	0.61	0.45%	0.54	1.27%	**0.70**	0.75%	0.62

Table 16.: Performance of the algorithm for different values of θ_4. $\bar{t}$ denotes the average time in seconds, $\bar{x}$ denotes the average gap and σ is the standard deviation. The numbers in bold marks the best result obtained throughout the test.

7 RESULTS

This chapter reports the computational results. Section 7.1 reports the results for the Solomon instances for several algorithms, where some of the previously described methods have been added gradually to better measure the quality of the individual methods.
Section 7.2 reports the results, for the final algorithm, for the Gehring & Homberger instances with 200, 400 and 600 customer.

7.1 SOLOMON INSTANCES

This section reports the results for 5 different algorithms for the 100 customer Solomon instances.

1. **ALNS Reloc**
   Standard ALNS only using the Relocate neighbourhood as the local search method.

2. **ALNS Ra-1/2o**
   Standard ALNS using the Ra-1o local method and the Ra-2o method when a new best solution is found.

3. **ALNS SPP**
   ALNS Reloc with the addition of the set partitioning problem.

4. **ALNS SPInsert**
   ALNS SPP using the SPInsert method with 1% probability in each iteration.

5. **ALNS BCP**
   ALNS SPP using the heuristic Branch Cut & Price method as a postoptimisation step.

Table 17 shows the overall result, where each of the 56 Solomon instances were solved 10 times.

		Best Solution		**Average Solution**		**Optimals**	
	$\bar{t}$	$\bar{x}$	$\max x$	$\bar{x}$	$\max x$	Total	Instances
ALNS Reloc	59.01	0.13%	1.64%	0.43%	2.45%	264	35
ALNS Ra-1/2o	208.16	0.10%	1.55%	0.32%	2.07%	295	37
ALNS SPP	90.55	0.03%	0.54%	0.13%	0.74%	353	42
ALNS SPInsert	98.58	0.03%	0.41%	0.15%	1.14%	355	43
ALNS BCP	109.87	0.01%	0.54%	0.11%	0.70%	387	49

Table 17.: Performance of the algorithms for the Solomon instances. $\bar{t}$ is the average time used for the algorithm, $\bar{x}$ denotes the average gap and $\max x$ is the worst achieved solution both when considering the best solution, or the average solution. The Optimal columns denotes how many optimal solutions were found, and for how many instances the algorithm found an optimal solution. Each instance were solved 10 times for each of the algorithms

Table 17 shows that ALNS Ra-1/2o improves the average gap by 0.11% when comparing with

ALNS Reloc. 0.11% is equivalent to an improvement of 25%. However, the runtime is approximately 3.5 times as great, which is far too much considering the improvement.

ALNS SPP improves the gap for the best solutions considerably, from 0.13% to 0.03%, and the average gap is 0.13%. The time is however increased by 50%, but this is an reasonable increase in time considering the huge improvement. Comparing ALNS SPInsert with ALNS SPP there is not any significant distinction beside the extra time.

ALNS BCP further improves results, and more than halves the average gap for the best solutions comparing with ALNS SPP (0.033% to 0.014%). The ALNS BCP method finds in total the optimal solution for 49 out of the 56 test instances. A more detailed solution summary for the ALNS BCP algorithm is shown in table 18.

	Best found solution		Average solution					Best found solution		Average solution			
	TD	x^b	TD	x	$\bar{t}$	Opt		TD	x^b	TD	x	$\bar{t}$	Opt
C101	827.30	0.00%	827.30	0.00%	32.31	10	C201	589.10	0.00%	589.10	0.00%	50.77	10
C102	827.30	0.00%	827.30	0.00%	30.34	10	C202	589.10	0.00%	589.10	0.00%	47.36	10
C103	826.30	0.00%	826.30	0.00%	32.34	10	C203	588.70	0.00%	588.70	0.00%	50.37	10
C104	822.90	0.00%	822.90	0.00%	33.97	10	C204	588.10	0.00%	588.10	0.00%	62.24	10
C105	827.30	0.00%	827.30	0.00%	30.98	10	C205	586.40	0.00%	586.40	0.00%	52.01	10
C106	827.30	0.00%	827.30	0.00%	28.72	10	C206	586.00	0.00%	586.00	0.00%	49.08	10
C107	827.30	0.00%	827.30	0.00%	29.13	10	C207	585.80	0.00%	585.80	0.00%	54.44	10
C108	827.30	0.00%	827.30	0.00%	29.16	10	C208	585.80	0.00%	585.80	0.00%	53.28	10
C109	827.30	0.00%	827.30	0.00%	29.83	10							
R101	1637.70	0.00%	1637.70	0.00%	35.01	10	R201	1143.30	0.01%	1143.30	0.01%	41.77	0
R102	1467.70	0.08%	1467.70	0.08%	35.20	0	R202	1029.60	0.00%	1031.22	0.16%	75.23	1
R103	1208.70	0.00%	1208.70	0.00%	35.42	10	R203	870.80	0.00%	873.12	0.27%	205.00	4
R104	971.50	0.00%	971.50	0.00%	81.68	10	R204	731.30	0.00%	735.61	0.59%	167.31	2
R105	1355.80	0.04%	1355.80	0.04%	37.36	0	R205	949.80	0.00%	949.80	0.00%	96.71	10
R106	1234.60	0.00%	1234.60	0.00%	66.45	10	R206	880.60	0.54%	880.95	0.58%	89.85	0
R107	1064.60	0.00%	1064.91	0.03%	182.57	9	R207	794.00	0.00%	797.77	0.47%	149.61	7
R108	932.10	0.00%	937.26	0.55%	502.59	3	R208	701.00	0.00%	701.94	0.13%	104.11	5
R109	1146.90	0.00%	1146.90	0.00%	118.64	10	R209	854.80	0.00%	856.21	0.16%	139.08	2
R110	1068.00	0.00%	1068.00	0.00%	173.61	10	R210	900.50	0.00%	906.83	0.70%	101.15	1
R111	1048.70	0.00%	1048.70	0.00%	186.20	10	R211	746.70	0.00%	748.10	0.19%	80.63	3
R112	948.90	0.03%	952.25	0.38%	344.05	0							
RC101	1619.80	0.00%	1619.80	0.00%	103.27	10	RC201	1261.80	0.00%	1261.93	0.01%	38.46	8
RC102	1457.40	0.00%	1457.40	0.00%	228.94	10	RC202	1092.30	0.00%	1092.30	0.00%	58.75	10
RC103	1258.00	0.00%	1259.07	0.09%	246.13	7	RC203	923.70	0.00%	923.70	0.00%	50.96	10
RC104	1132.30	0.00%	1132.30	0.00%	213.40	10	RC204	783.90	0.05%	785.61	0.27%	64.79	0
RC105	1513.70	0.00%	1514.05	0.02%	163.48	3	RC205	1154.00	0.00%	1154.00	0.00%	42.56	10
RC106	1373.50	0.06%	1373.54	0.06%	441.76	0	RC206	1051.10	0.00%	1053.85	0.26%	52.31	5
RC107	1207.80	0.00%	1208.76	0.08%	138.40	1	RC207	962.90	0.00%	968.59	0.59%	119.08	1
RC108	1114.20	0.00%	1118.49	0.39%	376.48	5	RC208	776.10	0.00%	776.10	0.00%	68.19	10
	CTD	$\bar{x}^b$	CTD	$\bar{x}$	$\bar{t}$ (s)	Opt		CTD	$\bar{x}^b$	CTD	$\bar{x}$	$\bar{t}$ (s)	Opt
C1	7440.30	0.00%	7440.30	0.00%	31	90	C2	4699.00	0.00%	4699.00	0.00%	52	80
R1	14085.20	0.01%	14094.02	0.09%	150	82	R2	9602.40	0.05%	9624.85	0.30%	114	35
RC1	10676.70	0.01%	10683.41	0.08%	239	46	RC2	8005.80	0.01%	8016.08	0.14%	62	54
All	54509.40	0.01%	54557.66	0.11%	110	387							

Table 18.: Solution statistics for every Solomon data set for ALNS BCP. TD is the travel distance. x^b is the gap for the best solution, and x is the gap for the average solution. $\bar{t}$ is the time spent in seconds, and Opt denotes how many optimal solution was found for each of the instances.

Table 18 shows that the algorithm finds an optimal solution in all of the 170 tests for the C-Class, and in total the algorithm finds 387 optimal solutions out of the 560 tests. For both the R and RC-Class at least half of the instances are solved with an average gap below 0.08%, but the average gap goes as high as 0.70% for the R-Class, and 0.59% for the RC-Class - see Figure 11.

Figure 11 figure shows a box plot for the average solutions for the three considered instance classes and the overall average performance for each data set. The figure makes it clear that the

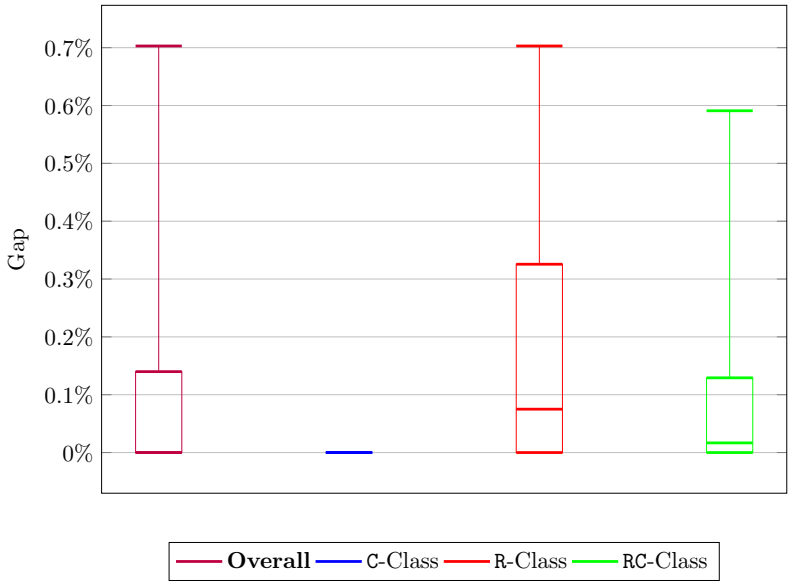


Figure 11.: A box plot based on the the average performance for each data set. The plot shows the 25% quartile, median, 75% quartile and the maximum gap for each of the three instances classes and overall.

ALNS BCP algorithm have the best performance on the clustered instance. It also shows that 75% of the instances are solved with an average gap below 0.15%.

The ALNS BCP algorithm can be considered as three different methods combined; an ALNS part, the set partitioning model and the Branch Cut & Price heuristic. Table 19 shows the average time for each instance for each part, as well as the total time. Furthermore the table also shows the average number of routes considered in the last set partitioning problem, and the percentage of edges considered by BCP the heuristic.
Table 19 shows that the ALNS run time is fairly consistent, and uses 41 seconds on average. The run time is larger for the instances with wider time windows, which was also to be expected as a customer can be inserted more places. The set partitioning and Branch Cut & Price methods are fast for the C-instances, which are known to be 'easy'. For the R-instances both the SP and BCP method are fast for most of the instances, but slow for few. More routes are generated for the instances with tight time windows, but there is not a clear relation between the number of routes and the time it takes to solve the set partitioning problem. Furthermore the table shows that only between 1%-3% of the total number of edges is considered in the BCP heuristic. In average, the time spent in the ALNS method is equivalent to 38% of the total time, 45% for the set partitioning method and 18% for the BCP heuristic.

Figure 12 illustrates the average number of routes in the end of the algorithm. The box plot is visual illustration of the data presented in table 19, and shows that the average number of routes is highly instance dependent.
Table 20 shows information related to the BCP heuristic. The first three columns after the instance name shows how many times out of 10 the heuristic improved the solution, or terminated prematurely due to the time limit (600 s). Hereafter the average percentage of the total number of edges considered is shown. The fifth column is the average time in seconds, for the instance. In the 9 cases where the time limit is reached, the algorithm failed to find a feasible solution. For the instances `R108` and `R203` the algorithm terminated twice due to the time limit. The 8

	Run times (s)							Run times (s)					
	ALNS	SP	BCP	Total	Avg. Routes	Avg. % edges		ALNS	SP	BCP	Total	Avg. Routes	Avg. % edges
C101	31.6	0.4	0.3	32.3	9129	1.31%	C201	50.1	0.1	0.6	50.8	343	1.19%
C102	29.2	0.5	0.6	30.3	7297	1.37%	C202	45.9	0.2	1.2	47.4	2470	1.34%
C103	30.9	0.9	0.6	32.3	9651	1.50%	C203	48.9	0.6	0.8	50.4	7636	1.29%
C104	30.9	2.5	0.6	34.0	33690	1.63%	C204	60.0	1.3	0.9	62.2	16692	1.31%
C105	30.0	0.6	0.3	31.0	16014	1.38%	C205	50.4	0.3	1.3	52.0	4040	1.37%
C106	27.6	0.8	0.3	28.7	12580	1.39%	C206	46.9	0.3	1.9	49.1	4318	1.50%
C107	28.1	0.7	0.4	29.1	15396	1.39%	C207	53.3	0.2	0.9	54.4	2953	1.32%
C108	27.2	1.5	0.5	29.2	24640	1.43%	C208	51.6	0.4	1.3	53.3	4790	1.44%
C109	27.1	2.0	0.7	29.8	30416	1.49%							
R101	30.7	4.0	0.3	35.0	47060	1.63%	R201	35.3	4.8	1.6	41.8	40517	1.68%
R102	31.5	3.6	0.1	35.2	63343	1.59%	R202	42.4	7.0	25.9	75.2	47203	2.47%
R103	29.0	6.3	0.1	35.4	69008	1.83%	R203	51.5	3.0	150.5	205.0	28739	1.55%
R104	28.5	45.6	7.6	81.7	66036	2.85%	R204	69.2	3.9	94.2	167.3	29994	2.11%
R105	24.8	12.0	0.5	37.4	73805	2.09%	R205	61.6	4.4	30.7	96.7	29753	1.85%
R106	37.9	25.8	2.8	66.4	82737	2.47%	R206	72.3	3.8	13.8	89.8	25883	1.86%
R107	40.8	79.2	62.6	182.6	81688	2.80%	R207	93.0	5.1	51.4	149.6	30311	1.89%
R108	41.3	270.5	190.9	502.6	64212	3.09%	R208	89.9	4.6	9.6	104.1	26773	1.78%
R109	34.9	75.1	8.7	118.6	77064	2.26%	R209	49.5	4.3	85.3	139.1	32019	2.18%
R110	37.2	72.1	64.3	173.6	102127	2.82%	R210	49.7	4.7	46.8	101.1	33511	2.69%
R111	39.4	82.3	64.5	186.2	71152	2.54%	R211	63.0	4.4	13.3	80.6	29705	1.67%
R112	32.8	296.7	14.6	344.1	66387	2.86%							
RC101	21.6	81.2	0.4	103.3	59612	2.16%	RC201	32.6	3.7	2.1	38.5	38289	1.75%
RC102	23.0	204.5	1.5	228.9	77682	2.48%	RC202	52.0	4.2	2.5	58.8	43140	1.71%
RC103	22.8	218.5	4.8	246.1	78203	2.92%	RC203	46.6	2.9	1.5	51.0	31905	1.39%
RC104	24.8	180.0	8.6	213.4	82224	2.98%	RC204	60.1	1.4	3.3	64.8	15737	1.45%
RC105	25.2	137.4	0.9	163.5	70614	2.09%	RC205	35.6	3.0	3.9	42.6	36195	1.84%
RC106	22.2	417.9	1.6	441.8	89802	2.62%	RC206	38.2	4.3	9.9	52.3	36625	1.90%
RC107	23.0	111.0	4.4	138.4	80789	2.79%	RC207	46.6	6.1	66.3	119.1	37966	2.83%
RC108	37.6	335.1	3.7	376.5	84124	2.68%	RC208	52.3	3.3	12.7	68.2	32124	1.69%
Avg.	41.4	49.1	19.3	109.9	41359	1.96%							

Table 19.: Time used in the ALNS, SP and BCP parts of the algorithm for the Solomon instances. Avg. Routes is the average number of routes considered in the last set partitioning problem. Avg. % edges is the average percentages of edges considered by the BCP heuristic. The times shown are the average time in seconds for each of the 56 Solomon data sets.

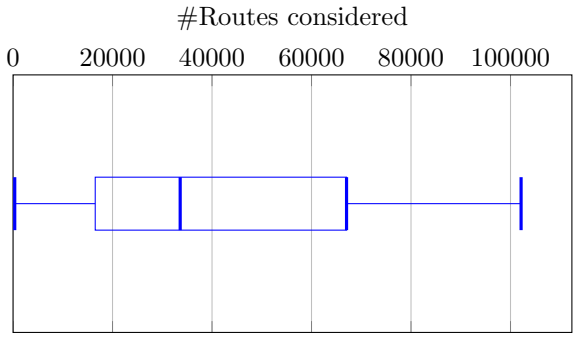


Figure 12.: Number of generated unique routes in the algorithm. The plot shows the minimum, 25% quartile, the median, the 75% quartile and the maximum number of average routes generated for each instance.

remaining runs for these two instances were solved using on average 38 s and 89 s respectively. Therefore these 2 cases must have been 'unlucky' cases, either missing an important edge or having a few extra edges that made the problem considerably harder.

	BCP Improvement						BCP Improvement				
	No	Yes	TLim	Avg. % edges	$\bar{t}$		No	Yes	TLim	Avg. % edges	$\bar{t}$
C101	10			1.31%	0.3	C201	10			1.19%	0.6
C102	10			1.37%	0.6	C202	10			1.34%	1.2
C103	10			1.50%	0.6	C203	10			1.29%	0.8
C104	10			1.63%	0.6	C204	10			1.31%	0.9
C105	10			1.38%	0.3	C205	10			1.37%	1.3
C106	10			1.39%	0.3	C206	10			1.50%	1.9
C107	10			1.39%	0.4	C207	10			1.32%	0.9
C108	10			1.43%	0.5	C208	10			1.44%	1.3
C109	10			1.49%	0.7						
R101	10			1.63%	0.3	R201	5	5		1.68%	1.6
R102	10			1.59%	0.1	R202	6	4		2.47%	25.9
R103	10			1.83%	0.1	R203	10		2	1.55%	150.5
R104	10			2.85%	7.6	R204	7	3	1	2.11%	94.2
R105	10			2.09%	0.5	R205	10			1.85%	30.7
R106	10			2.47%	2.8	R206	10			1.86%	13.8
R107	1	9	1	2.80%	62.6	R207	8	2		1.89%	51.4
R108	6	4	2	3.09%	190.9	R208	7	3		1.78%	9.6
R109	10			2.26%	8.7	R209	10		1	2.18%	85.3
R110	9	1	1	2.82%	64.3	R210	7	3		2.69%	46.8
R111	10		1	2.54%	64.5	R211	6	4		1.67%	13.3
R112	8	2		2.86%	14.6						
RC101	10			2.16%	0.4	RC201	9	1		1.75%	2.1
RC102		10		2.48%	1.5	RC202	10			1.71%	2.5
RC103	9	1		2.92%	4.8	RC203	10			1.39%	1.5
RC104	10			2.98%	8.6	RC204	10			1.45%	3.3
RC105	7	3		2.09%	0.9	RC205	10			1.84%	3.9
RC106	10			2.62%	1.6	RC206	7	3		1.90%	9.9
RC107	6	4		2.79%	4.4	RC207	9	1		2.83%	66.3
RC108	10			2.68%	3.7	RC208	10			1.69%	12.7

Table 20.: Overview of BCP performance for the Solomon instances. $\bar{t}$ is the average time spent for the BCP heuristic, in seconds.

Figure 13 on page 56 is a detailed illustration of the development of the algorithm. The data comes from an average out of 560 executions of ALNS SPP (10 for each of the 56 Solomon instances), where the data were printed in the end of every segment (every 125 iteration).

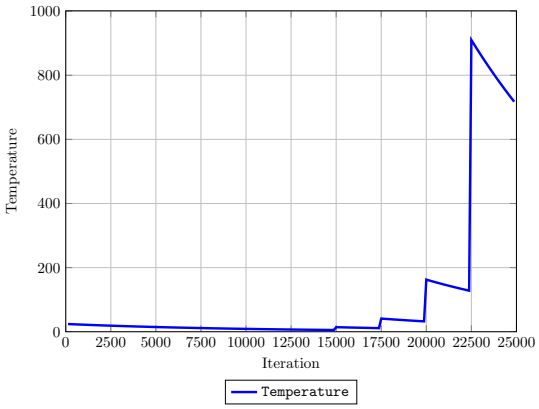
Figure 13a shows the temperature throughout the algorithm. The steep increases are when the algorithm reheats the temperature due to no changes in the best solution. However, adding 5% to what the temperature was when the the best solution was achieved should not lead to these steep increases seen in fig. 13a. A quick review of the how the temperature was actually reheated revealed an error, which lead to the steep increases. Appendix B, on page 69, explains this error and shows how the graphs look after correcting the error.

Figure 13b shows the average number of routes considered in the set partitioning model throughout the algorithm. Here it is clear that more routes are added when the temperature increases, as seen from the gradient.

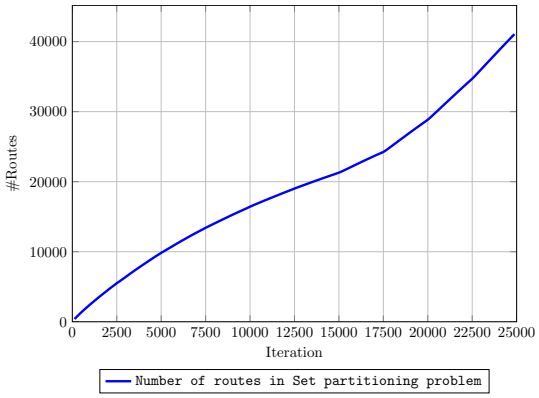
Figure 13c plots the probabilities for the insertion methods. The regret method is best in the beginning of the algorithm, hereafter the greedy method has a 'come back' when the temperature increases as this allows more solutions to be accepted, thus increasing the score.

Figure 13d shows the probabilities for the destroy methods. The figure shows that the `Random Route Removal` destroy heuristic is the worst, and the probability immediately starts decreasing from the beginning. Furthermore, the graph also shows that `Shaw Removal`, `ShawWorst Removal` and `Random Removal` are equally good, with a slight advantage for `Shaw Removal`. The probability for `Worst Removal` slowly decreases in the beginning of the algorithm, but slowly comes back when the temperature increases.

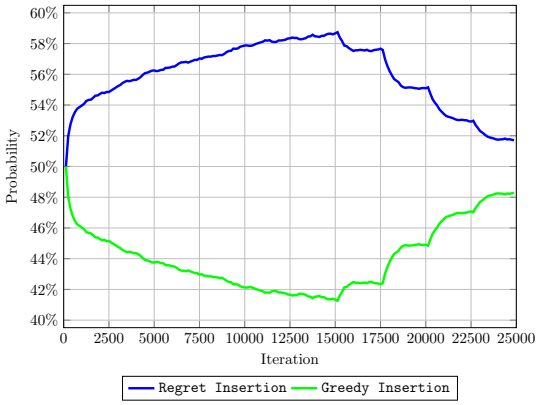
From fig. 13e it is clear when the set partitioning problem is being solved (every 2500 iteration), as the gap clearly becomes smaller. The plot also shows that the algorithm on average only finds minor improvements the last 2500 iterations, and hence more iterations would not be able to improve the quality of the solutions substantially.



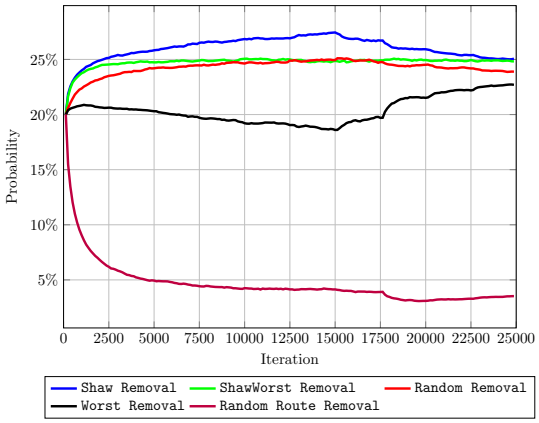
(a) The temperature throughout the algorithm.



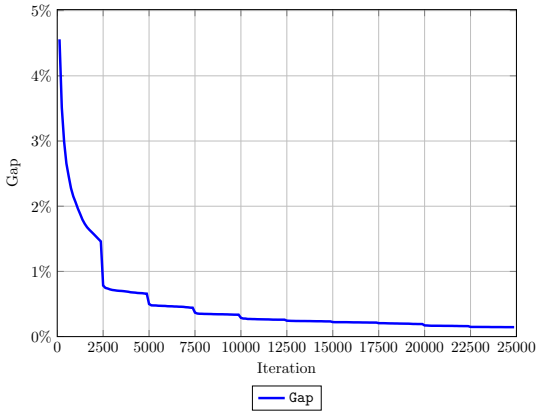
(b) Number of generated unique routes in the algorithm



(c) Probability for the insertion methods throughout the algorithm



(d) Probability for the destroy methods throughout the algorithm



(e) The gap for the current best solution throughout the algorithm.

Figure 13.: Detailed illustration of the development of the algorithm.

7.2 GEHRING & HOMBERGER INSTANCES

This section reports the results for the Gehring and Homberger instances with 200, 400 and 600 customers, for the ALNS Reloc, ALNS SPP and ALNS BCP algorithms.
Not many have tried to solve these instances using exact methods, and the heuristic approaches often tries to minimise the number of vehicles, and then find a minimum cost distribution plan using this number of vehicles. Therefore the algorithms are compared with the best solution found by one of the three algorithms. For the 200 and 400 customer instances the results are compared to an ALNS 75K configuration of the algorithm described in [23]. The results from ALNS 75K were provided by my supervisor, Stefan Røpke.

7.2.1 200 Customers

Table 21 shows the results for the ALNS BCP (referred to as BCP in the table), ALNS SPP (SPP) and ALNS Reloc (ALNS) algorithms. Each instance were solved 10 times.

		BCP		SPP		ALNS				BCP		SPP		ALNS	
	Best	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$		Best	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$
C1_2_1	2698.6	0.00%	149	0.00%	148	0.00%	147	C2_2_1	1922.1	0.00%	290	0.00%	287	0.00%	274
C1_2_2	2694.3	0.00%	155	0.00%	153	0.00%	162	C2_2_2	1851.4	0.26%	325	0.28%	309	0.44%	288
C1_2_3	2675.8	0.00%	384	0.00%	362	0.01%	150	C2_2_3	1768.3	0.68%	707	0.68%	375	1.32%	350
C1_2_4	2625.6	0.03%	1181	0.15%	1120	0.71%	160	C2_2_4	1706.9	0.79%	1348	0.79%	732	0.78%	391
C1_2_5	2696.0	0.00%	168	0.00%	167	0.00%	306	C2_2_5	1869.6	0.41%	324	0.41%	284	0.27%	268
C1_2_6	2694.9	0.00%	155	0.00%	154	0.00%	142	C2_2_6	1844.9	0.00%	333	0.00%	293	0.31%	308
C1_2_7	2694.9	0.00%	179	0.00%	178	0.00%	150	C2_2_7	1842.3	0.24%	344	0.28%	298	0.58%	272
C1_2_8	2684.0	0.00%	261	0.00%	257	0.01%	146	C2_2_8	1817.5	0.25%	329	0.35%	296	1.33%	266
C1_2_9	2639.6	0.01%	545	0.03%	537	0.73%	153	C2_2_9	1815.1	0.28%	537	0.45%	327	0.67%	352
C1_2_10	2627.1	0.73%	1218	1.29%	1150	2.18%	158	C2_2_10	1791.3	0.08%	887	0.16%	353	0.08%	365
R1_2_1	4686.1	0.00%	276	0.01%	270	0.45%	254	R2_2_1	3480.6	0.05%	344	0.29%	340	1.30%	234
R1_2_2	3962.2	0.02%	1019	0.03%	872	0.57%	206	R2_2_2	3019.1	0.10%	413	0.18%	384	0.66%	291
R1_2_3	3389.8	0.54%	1760	0.54%	1156	0.80%	146	R2_2_3	2550.0	0.01%	567	0.03%	452	0.56%	320
R1_2_4	3104.6	1.17%	1779	1.17%	1169	0.93%	104	R2_2_4	1928.5	0.66%	676	0.68%	642	1.08%	432
R1_2_5	4053.2	0.11%	1388	0.11%	1007	0.59%	134	R2_2_5	3063.6	0.28%	762	0.38%	353	0.61%	261
R1_2_6	3569.3	0.47%	1774	0.47%	1168	0.46%	128	R2_2_6	2677.2	0.14%	477	0.14%	427	0.48%	307
R1_2_7	3144.4	0.53%	1763	0.55%	1153	1.76%	115	R2_2_7	2305.7	0.73%	877	0.80%	531	0.87%	360
R1_2_8	3011.1	1.44%	1836	1.44%	1219	0.14%	102	R2_2_8	1842.4	0.89%	978	0.89%	727	1.18%	544
R1_2_9	3739.9	0.21%	1703	0.21%	1090	0.54%	121	R2_2_9	2844.0	0.13%	532	0.29%	402	0.63%	275
R1_2_10	3344.1	0.63%	1748	0.63%	1144	0.88%	100	R2_2_10	2553.0	0.51%	484	0.62%	418	1.29%	286
RC1_2_1	3521.0	0.31%	1651	0.42%	1100	0.90%	107	RC2_2_1	2802.9	0.06%	296	0.06%	283	0.24%	587
RC1_2_2	3231.0	0.40%	1716	0.40%	1151	0.83%	88	RC2_2_2	2483.3	0.18%	451	0.32%	396	0.86%	316
RC1_2_3	3013.3	0.75%	1754	0.75%	1149	1.11%	183	RC2_2_3	2229.1	0.31%	635	0.31%	552	0.37%	408
RC1_2_4	2918.2	1.29%	1768	1.29%	1164	1.19%	149	RC2_2_4	1854.8	0.93%	754	0.98%	562	0.84%	574
RC1_2_5	3326.9	0.29%	1752	0.35%	1246	1.71%	130	RC2_2_5	2491.4	0.11%	449	0.16%	345	0.45%	295
RC1_2_6	3316.3	0.19%	1676	0.21%	1106	0.95%	148	RC2_2_6	2496.4	0.03%	411	0.03%	401	0.07%	340
RC1_2_7	3205.4	1.21%	1774	1.21%	1166	1.18%	132	RC2_2_7	2287.7	0.15%	435	0.19%	399	0.73%	458
RC1_2_8	3134.8	0.73%	1970	0.73%	1363	0.60%	136	RC2_2_8	2151.2	0.15%	508	0.15%	476	0.39%	350
RC1_2_9	3128.9	0.55%	1770	0.55%	1165	0.29%	137	RC2_2_9	2086.6	0.15%	808	0.16%	486	0.40%	479
RC1_2_10	3053.4	0.88%	1778	0.88%	1172	0.54%	106	RC2_2_10	1989.2	0.63%	802	0.64%	550	0.48%	546
C1		0.08%	439	0.15%	423	0.36%	167	C2		0.30%	542	0.34%	355	0.58%	313
R1		0.51%	1505	0.52%	1025	0.71%	141	R2		0.35%	611	0.43%	468	0.86%	331
RC1		0.66%	1761	0.68%	1178	0.93%	132	RC2		0.27%	555	0.30%	445	0.48%	435
All		0.36%	902	0.40%	649	0.66%	253								

Table 21.: Solution statistics for every Gehring/Homberger data set with 200 customers for ALNS BCP, ALNS SPP and ALNS Reloc. Best is the best solution found by one of the three algorithms, $\bar{x}$ is the average gap comparing with the best found solution. $\bar{t}$ denotes the time, in seconds, used. Each instance were solved 10 times for each of the algorithms

The time for ALNS SPP and ALNS Reloc are comparable for most of the C-instances. ALNS

Reloc is slower than ALNS SPP in some cases, and provides the best average result other cases. This is due to the randomness of the algorithms. ALNS Reloc uses on average 253 seconds, and achieves an averages solution 0.66% from the best found. For 190 of the 600 ALNS SPP executions at least 900 seconds were used to solve the set partitioning problems, and thus might hit the time limit several times, which has a huge impact on the overall run time. On average 377 seconds are to used solve the set partitioning problem. ALNS BCP further improves the solution found by ALNS SPP, and almost halves the average gap, comparing with ALNS Reloc. Table 22 shows for how many times out of 10 the BCP heuristic improved the solution found by ALNS SPP, furthermore it shows the average percentage of edges considers, as well as the time spent.

	BCP Improvement						BCP Improvement				
	No	Yes	TLim	Avg. % edges	$\bar{t}$		No	Yes	TLim	Avg. % edges	$\bar{t}$
C1_2_1	10			0.60%	0.9	C2_2_1	10			0.59%	3.2
C1_2_2	10			0.70%	2.1	C2_2_2	9	1		0.65%	16.1
C1_2_3	5	5		0.98%	22.3	C2_2_3	10		4	1.07%	331.9
C1_2_4	1	9		1.31%	60.7	C2_2_4	10		10	1.52%	615.7
C1_2_5	10			0.60%	0.9	C2_2_5	10			0.67%	40.2
C1_2_6	10			0.60%	1.0	C2_2_6	10			0.74%	39.2
C1_2_7	10			0.62%	1.0	C2_2_7	8	2		0.69%	46.8
C1_2_8	9	1		0.81%	3.8	C2_2_8	7	3		0.71%	32.8
C1_2_9	8	2		1.00%	8.1	C2_2_9	6	4	1	0.86%	209.9
C1_2_10	1	9		1.18%	68.1	C2_2_10	8	2	2	0.74%	534.4
R1_2_1	6	4	1	0.76%	6.5	R2_2_1	2	8		0.76%	4.3
R1_2_2	5	5	1	1.00%	146.7	R2_2_2	6	4		0.80%	29.0
R1_2_3	10		10	1.42%	604.3	R2_2_3	9	1	1	0.85%	114.7
R1_2_4	10		10	1.69%	610.2	R2_2_4	7	3		0.64%	33.4
R1_2_5	9	1	3	1.08%	381.6	R2_2_5	6	4	4	1.09%	408.6
R1_2_6	9	1	9	1.40%	606.7	R2_2_6	9	1		0.75%	49.8
R1_2_7	9	1	9	1.40%	610.5	R2_2_7	9	1	4	1.05%	346.4
R1_2_8	10		10	1.79%	617.3	R2_2_8	10		3	0.82%	250.3
R1_2_9	10		10	1.25%	612.9	R2_2_9	4	6		0.98%	129.7
R1_2_10	10		10	1.39%	604.1	R2_2_10	8	2		0.83%	65.6
RC1_2_1	6	4	6	1.13%	551.1	RC2_2_1	9	1		0.72%	12.5
RC1_2_2	10		9	1.38%	565.6	RC2_2_2	8	2		0.81%	55.0
RC1_2_3	10		10	1.45%	604.7	RC2_2_3	10			0.90%	83.7
RC1_2_4	10		10	1.68%	604.2	RC2_2_4	9	1	2	0.93%	192.3
RC1_2_5	6	4	5	1.29%	506.1	RC2_2_5	4	6		0.86%	104.0
RC1_2_6	9	1	8	1.28%	570.3	RC2_2_6	9	1		0.66%	10.7
RC1_2_7	10		10	1.43%	607.8	RC2_2_7	7	3		0.73%	36.0
RC1_2_8	10		10	1.53%	607.6	RC2_2_8	10			0.67%	32.5
RC1_2_9	10		10	1.49%	604.5	RC2_2_9	9	1	4	1.05%	322.3
RC1_2_10	10		10	1.58%	606.1	RC2_2_10	8	2	4	0.99%	252.1

Table 22.: Overview of BCP performance for the GH200 instances. $\bar{t}$ is the average time spent for the BCP heuristic, in seconds.

In total the BCP heuristic terminated due to the time limit in 200 out of the 600 executions, and found an improvement in 106 runs. In 197 out of the 200 executions where the full 600s seconds were used, the heuristic failed to find a feasible solution. Table 22 also shows that between 0.59% and 1.79% of the total number of edges is considered, and on average only 1.02% is included in the problem. This is almost half of the percentage of edges considered for the Solomon instances (1.96%), but the average number of edges considered is more than doubled (from 200 to 410), which is the main reason for the longer run time.

Table 23 compares ALNS BCP with results from an ALNS 75K configuration of the algorithm

presented in [23].[1] However, there is a slight difference between the algorithms. ALNS 75K uses truncated precision for both the travel times and costs, whereas ALNS SPP uses full precision for both the travel times and costs throughout the algorithm, and in the end uses truncated precision to calculate the cost of the best solution. The BCP heuristic uses truncated precision, but ALNS SPP might have failed to adds some arcs as these were considered infeasible when using full precision. This favours ALNS 75K in the comparison.

	ALNS BCP			ALNS 75K				ALNS BCP			ALNS 75K		
	Best	$\bar{x}$	$\bar{t}$	x^b	$\bar{x}$	$\bar{t}$		Best	$\bar{x}$	$\bar{t}$	x^b	$\bar{x}$	$\bar{t}$
C1_2_1	2698.60	0.00%	149	0.00%	0.00%	22	C2_2_1	1922.10	0.00%	290	0.00%	0.72%	29
C1_2_2	2694.30	0.00%	155	0.00%	0.00%	28	C2_2_2	1851.40	0.26%	325	0.03%	1.05%	35
C1_2_3	2675.80	0.00%	384	0.02%	0.06%	31	C2_2_3	1768.30	0.68%	707	-0.17%	0.72%	40
C1_2_4	2625.60	0.03%	1181	0.00%	0.05%	31	C2_2_4	1706.90	0.79%	1348	-0.01%	0.49%	41
C1_2_5	2696.00	0.00%	168	-0.04%	-0.04%	24	C2_2_5	1869.60	0.41%	324	0.00%	0.62%	35
C1_2_6	2694.90	0.00%	155	0.00%	0.00%	24	C2_2_6	1844.90	0.00%	333	0.29%	0.85%	38
C1_2_7	2694.90	0.00%	179	0.00%	0.00%	27	C2_2_7	1842.30	0.24%	344	0.03%	0.37%	38
C1_2_8	2684.00	0.00%	261	0.00%	0.00%	28	C2_2_8	1817.50	0.25%	329	-0.03%	0.60%	41
C1_2_9	2639.60	0.01%	545	0.00%	0.00%	30	C2_2_9	1815.10	0.28%	537	0.37%	0.58%	40
C1_2_10	2627.10	0.73%	1218	0.00%	0.00%	32	C2_2_10	1791.30	0.08%	887	0.18%	0.69%	43
R1_2_1	4686.10	0.00%	276	-0.36%	-0.25%	24	R2_2_1	3480.60	0.05%	344	1.88%	3.10%	36
R1_2_2	3962.20	0.02%	1019	-1.02%	-0.71%	27	R2_2_2	3019.10	0.10%	413	0.57%	3.54%	39
R1_2_3	3389.80	0.54%	1760	-0.29%	0.27%	31	R2_2_3	2550.00	0.01%	567	0.17%	1.31%	41
R1_2_4	3104.60	1.17%	1779	-1.27%	-0.77%	33	R2_2_4	1928.50	0.66%	676	0.54%	2.79%	45
R1_2_5	4053.20	0.11%	1388	0.28%	0.46%	24	R2_2_5	3063.60	0.28%	762	0.89%	1.70%	41
R1_2_6	3569.30	0.47%	1774	-0.07%	0.13%	27	R2_2_6	2677.20	0.14%	477	1.96%	3.61%	43
R1_2_7	3144.40	0.53%	1763	0.66%	1.00%	30	R2_2_7	2305.70	0.73%	877	0.43%	2.72%	44
R1_2_8	3011.10	1.44%	1836	-1.93%	-1.67%	32	R2_2_8	1842.40	0.89%	978	1.03%	2.73%	46
R1_2_9	3739.90	0.21%	1703	0.00%	0.28%	26	R2_2_9	2844.00	0.13%	532	1.17%	2.73%	44
R1_2_10	3344.10	0.63%	1748	-1.22%	-0.98%	28	R2_2_10	2553.00	0.51%	484	1.79%	3.30%	47
RC1_2_1	3521.00	0.31%	1651	0.01%	0.28%	24	RC2_2_1	2802.90	0.06%	296	-0.11%	0.42%	34
RC1_2_2	3231.00	0.40%	1716	-0.05%	0.13%	27	RC2_2_2	2483.30	0.18%	451	0.86%	1.23%	38
RC1_2_3	3013.30	0.75%	1754	-0.30%	0.07%	30	RC2_2_3	2229.10	0.31%	635	0.20%	0.75%	40
RC1_2_4	2918.20	1.29%	1768	-2.04%	-1.74%	32	RC2_2_4	1854.80	0.93%	754	0.37%	1.98%	42
RC1_2_5	3326.90	0.29%	1752	0.20%	0.37%	25	RC2_2_5	2491.40	0.11%	449	0.41%	1.07%	40
RC1_2_6	3316.30	0.19%	1676	-0.03%	0.14%	25	RC2_2_6	2496.40	0.03%	411	0.17%	0.89%	40
RC1_2_7	3205.40	1.21%	1774	-0.86%	-0.47%	26	RC2_2_7	2287.70	0.15%	435	0.35%	2.15%	45
RC1_2_8	3134.80	0.73%	1970	-2.04%	-1.51%	27	RC2_2_8	2151.20	0.15%	508	0.66%	1.77%	44
RC1_2_9	3128.90	0.55%	1770	-1.53%	-1.17%	27	RC2_2_9	2086.60	0.15%	808	0.23%	1.76%	45
RC1_2_10	3053.40	0.88%	1778	-1.86%	-1.53%	29	RC2_2_10	1989.20	0.63%	802	0.31%	2.80%	48
Average	2699.18	0.36%	902	0.01%	0.69%	34							

Table 23.: Comparison between ALNS BCP and ALNS 75K for the GH200 instances. The column 'Best' denotes the best solution found by the ALNS BCP heuristic, $\bar{x}$ is the average gap, comparing with the column 'Best'. $\bar{t}$ is the average time. For ALNS 75K x^b denotes the gap between the best found solution by ALNS BCP and ALNS 75k.

Table 23 shows that ALNS 75K is more than 30 times faster than ALNS BCP, while still achieving best solutions close to the best solutions found by ALNS BCP. However, ALNS BCP finds solutions that on average are better than ALNS 75K, and the average gap for the solutions found by ALNS BCP is half of the average gap for ALNS 75K.

7.2.2 400 Customers

Table 24 shows the results for the ALNS BCP (referred to as BCP in the table), ALNS SPP (SPP) and ALNS Reloc (ALNS) algorithms for the Gehring and Homberger instances with 400 customers. Each instance were solved 10 times.

1 In the test ALNS 75K used 8 threads, and the actual times have thus been multiplied with 8 to make the comparison more fair.

		BCP		SPP		ALNS				BCP		SPP		ALNS	
	Best	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$		Best	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$
C1_4_2	7113.30	0.00%	797	0.00%	771	0.00%	848	C2_4_2	3914.10	0.34%	1826	0.36%	1576	0.94%	1368
C1_4_3	6932.70	0.38%	1941	0.44%	1419	2.52%	740	C2_4_3	3761.20	1.22%	2906	1.22%	2318	1.54%	1497
C1_4_4	7238.80	1.62%	2159	1.62%	1543	0.65%	715	C2_4_4	3596.10	1.21%	3017	1.21%	2413	1.40%	1490
C1_4_5	7138.80	0.00%	575	0.00%	565	0.00%	789	C2_4_5	3923.30	0.22%	1651	0.36%	1564	1.84%	1273
C1_4_6	7140.10	0.00%	518	0.00%	508	0.12%	841	C2_4_6	3860.20	0.07%	2062	0.53%	1900	2.09%	1154
C1_4_7	7136.20	0.00%	526	0.00%	517	0.06%	738	C2_4_7	3876.90	2.58%	2702	2.93%	2137	2.88%	1260
C1_4_8	7098.00	0.10%	1448	0.13%	1431	2.32%	731	C2_4_8	3827.80	2.39%	3056	2.39%	2451	2.29%	1462
C1_4_9	7025.30	1.73%	2134	1.75%	1572	3.29%	729	C2_4_9	3910.90	1.00%	2809	1.00%	2203	0.66%	1276
C1_4_10	7210.30	3.19%	2230	3.19%	1625	0.74%	687	C2_4_10	3753.00	2.51%	3183	2.51%	2514	1.93%	1049
R1_4_1	10357.00	0.14%	2429	0.14%	1823	1.38%	884	R2_4_1	7523.30	0.94%	2605	1.02%	2006	2.14%	1355
R1_4_2	9068.90	1.52%	2521	1.52%	1905	0.72%	626	R2_4_2	6488.40	1.33%	2917	1.46%	2289	1.78%	1657
R1_4_3	8061.50	3.01%	2425	3.01%	1788	1.14%	521	R2_4_3	5400.90	0.62%	3835	0.62%	3318	0.94%	1952
R1_4_4	7623.00	3.23%	2173	3.23%	1568	1.44%	507	R2_4_4	4223.40	0.49%	3465	0.50%	3116	0.82%	2063
R1_4_5	9314.40	0.98%	2244	0.98%	1615	0.83%	605	R2_4_5	6568.00	0.58%	2244	0.73%	1652	1.37%	1357
R1_4_6	8592.70	1.84%	2462	1.84%	1814	0.69%	554	R2_4_6	5848.70	0.98%	2588	1.00%	2061	1.39%	1712
R1_4_7	7956.00	1.51%	2446	1.51%	1843	0.81%	496	R2_4_7	4921.00	0.98%	3882	0.98%	3328	0.78%	1942
R1_4_8	7621.90	2.43%	2428	2.43%	1824	1.24%	486	R2_4_8	4011.00	0.32%	2787	0.32%	2286	1.25%	2137
R1_4_9	9001.00	1.14%	2243	1.14%	1597	0.46%	542	R2_4_9	6081.30	0.80%	2473	0.82%	1953	1.75%	1474
R1_4_10	8434.50	1.86%	2191	1.86%	1550	0.74%	500	R2_4_10	5675.70	1.12%	3242	1.14%	2676	2.14%	1572
RC1_4_1	8729.90	1.84%	2629	1.84%	2014	1.30%	587	RC2_4_1	6148.10	0.07%	1678	0.23%	1480	1.05%	1340
RC1_4_2	8202.50	1.53%	2210	1.53%	1588	0.81%	545	RC2_4_2	5407.50	0.01%	2133	0.06%	2056	0.76%	1509
RC1_4_3	7815.40	2.10%	2192	2.10%	1588	0.96%	486	RC2_4_3	4573.00	0.19%	2096	0.22%	1946	1.00%	2697
RC1_4_4	7617.80	1.59%	2468	1.59%	1867	1.20%	503	RC2_4_4	3606.10	0.23%	3369	0.23%	2894	0.71%	2931
RC1_4_5	8512.60	1.66%	2539	1.66%	1926	1.17%	576	RC2_4_5	5401.70	0.49%	2591	0.50%	2037	1.55%	1949
RC1_4_6	8491.40	1.94%	2509	1.94%	1904	1.25%	566	RC2_4_6	5342.10	0.24%	2368	0.25%	1881	1.12%	1844
RC1_4_7	8348.10	1.40%	2513	1.40%	1897	0.81%	553	RC2_4_7	5002.90	0.34%	2450	0.35%	2003	1.19%	1731
RC1_4_8	8180.00	1.84%	2498	1.84%	1873	0.98%	528	RC2_4_8	4712.50	0.34%	3008	0.34%	2500	1.27%	1612
RC1_4_9	8087.60	1.79%	2457	1.79%	1855	1.16%	518	RC2_4_9	4513.90	1.02%	2750	1.02%	2251	1.84%	2285
RC1_4_10	7958.20	1.56%	2514	1.56%	1912	0.84%	494	RC2_4_10	4284.20	0.68%	2919	0.70%	2637	1.24%	2135
C1		0.70%	1310	0.71%	1072	0.97%	762	C2		1.15%	2465	1.25%	2049	1.59%	1323
R1		1.77%	2356	1.77%	1733	0.95%	572	R2		0.82%	3004	0.86%	2469	1.44%	1722
RC1		1.72%	2453	1.72%	1842	1.05%	535	RC2		0.36%	2536	0.39%	2169	1.17%	2003
All		1.09%	2354	1.12%	1889	1.19%	1153								

Table 24.: Solution statistics for every Gehring/Homberger data set with 400 customers for ALNS BCP, ALNS SPP and ALNS Reloc. Best is the best solution found by one of the three algorithms, $\bar{x}$ is the average gap comparing with the best found solution. $\bar{t}$ denotes the time, in seconds, used. Each instance were solved 10 times for each of the algorithms

The run times for ALNS SPP for the R1 and RC1 instances is considerably larger than the run times for ALNS Reloc. In each run the set partitioning problem is solved 10 times, each with a time limit on 100 seconds. The average run time for ALNS SPP is approximately 1200 seconds slower for these two instance classes, which is a clear indication that the set partitioning problem is too hard to be solved, within the time limit. ALNS Reloc gives the best results for the R1 and RC1 classes, which could be explained by the combination of returning too often to the best solution in the set partitioning method and the faulty way of reheating the temperature, which is incorporated in the set partitioning method. For the rest of the classes ALNS SPP gives better results than ALNS Reloc, and the difference in time is not as drastic as seen for the R1 and RC1 classes. For 21 out of 60 instance the overall best found solution is found by the ALNS Reloc heuristic. Overall ALNS BCP gives the best average results, but is only 0.10% better than ALNS Reloc and the average run time is more than doubled.
Table 25 shows how many times out of 10 the BCP heuristic improves the solution found by the ALNS SPP, as well as the average time spent and the average percentage of edges considered.

	BCP Improvement						BCP Improvement				
	No	Yes	TLim	Avg. % edges	$\bar{t}$		No	Yes	TLim	Avg. % edges	$\bar{t}$
C1_4_1	10			0.28%	4.1	C2_4_1	7	3		0.30%	22.9
C1_4_2	10			0.33%	26.0	C2_4_2	7	3	1	0.39%	250.1
C1_4_3	7	3	7	0.53%	522.0	C2_4_3	9	1	9	0.54%	588.6
C1_4_4	10		10	0.86%	615.5	C2_4_4	10		10	0.79%	603.5
C1_4_5	10			0.29%	9.5	C2_4_5	8	2		0.33%	86.6
C1_4_6	10			0.30%	9.7	C2_4_6	6	4		0.35%	162.1
C1_4_7	10			0.29%	9.1	C2_4_7	9	1	8	0.54%	565.4
C1_4_8		10		0.37%	17.4	C2_4_8	10		10	0.62%	604.6
C1_4_9	9	1	9	0.56%	561.7	C2_4_9	10		10	0.68%	605.6
C1_4_10	10		10	0.69%	604.6	C2_4_10	10		10	0.69%	668.9
R1_4_1	10		10	0.45%	606.2	R2_4_1	9	1	9	0.52%	598.9
R1_4_2	10		10	0.73%	615.8	R2_4_2	8	2	8	0.61%	628.0
R1_4_3	10		10	0.89%	637.1	R2_4_3	10		7	0.58%	516.7
R1_4_4	10		10	1.04%	604.9	R2_4_4	8	2	5	0.57%	349.5
R1_4_5	10		10	0.59%	628.4	R2_4_5	6	4	6	0.55%	591.6
R1_4_6	10		10	0.79%	648.7	R2_4_6	9	1	8	0.73%	527.3
R1_4_7	10		10	0.92%	603.6	R2_4_7	9	1	9	0.81%	553.8
R1_4_8	10		10	1.04%	604.5	R2_4_8	10		8	0.64%	500.6
R1_4_9	10		10	0.72%	646.6	R2_4_9	7	3	7	0.62%	519.7
R1_4_10	10		10	0.79%	640.9	R2_4_10	9	1	9	0.75%	566.1
RC1_4_1	10		10	0.62%	615.3	RC2_4_1	1	9		0.43%	198.3
RC1_4_2	10		10	0.79%	622.0	RC2_4_2	2	8		0.37%	77.0
RC1_4_3	10		10	0.95%	604.0	RC2_4_3	4	6	2	0.40%	149.7
RC1_4_4	10		10	1.03%	601.0	RC2_4_4	8	2	7	0.58%	475.3
RC1_4_5	10		10	0.75%	612.6	RC2_4_5	8	2	8	0.54%	553.6
RC1_4_6	10		10	0.75%	605.5	RC2_4_6	9	1	7	0.50%	487.0
RC1_4_7	10		10	0.79%	616.4	RC2_4_7	8	2	7	0.51%	447.0
RC1_4_8	10		10	0.89%	624.3	RC2_4_8	9	1	8	0.66%	507.9
RC1_4_9	10		10	0.89%	602.0	RC2_4_9	10		8	0.59%	499.8
RC1_4_10	10		10	0.87%	601.6	RC2_4_10	7	3	4	0.53%	281.6

Table 25.: Overview of BCP performance for the GH400 instances. $\bar{t}$ is the average time spent for the BCP heuristic, in seconds.

In total 421 executions terminated due to the time limit, and an improvement were found in 77 executions, most of them for the RC2 instances. This is a clear indication that the problems becomes too large to be solved efficiently using Branch Cut & Price, even if only a small percentage of the edges is considered.
Table 26 compares the results from ALNS BCP with the results from the previously described ALNS 75K algorithm. To make the comparison more fair, the column 'Best' is the best solution found by ALNS BCP, and could thus be different from the best solution reported in Table 24,

as this table reported the best result found by one of the three tested algorithms.

	ALNS BCP			ALNS 75K				ALNS BCP			ALNS 75K		
	Best	$\bar{x}$	$\bar{t}$	x^b	$\bar{x}$	$\bar{t}$		Best	$\bar{x}$	$\bar{t}$	x^b	$\bar{x}$	$\bar{t}$
C1_4_1	7138.80	0.00%	776	0.00%	0.00%	49	C2_4_1	4100.30	0.00%	1437	1.20%	3.84%	56
C1_4_2	7113.30	0.00%	797	0.00%	0.00%	57	C2_4_2	3914.10	0.34%	1826	0.58%	3.52%	73
C1_4_3	6932.70	0.38%	1941	0.01%	0.07%	61	C2_4_3	3761.20	1.22%	2906	2.90%	4.92%	80
C1_4_4	7243.40	1.55%	2159	-6.27%	-6.06%	66	C2_4_4	3596.10	1.21%	3017	1.85%	3.57%	88
C1_4_5	7138.80	0.00%	575	0.00%	0.00%	51	C2_4_5	3923.30	0.22%	1651	0.43%	2.69%	66
C1_4_6	7140.10	0.00%	518	0.00%	0.00%	51	C2_4_6	3860.20	0.07%	2062	1.66%	3.40%	73
C1_4_7	7136.20	0.00%	526	0.00%	0.00%	53	C2_4_7	3876.90	2.58%	2702	0.22%	2.54%	73
C1_4_8	7098.00	0.10%	1448	-0.21%	-0.19%	54	C2_4_8	3871.10	1.24%	3056	-2.08%	1.04%	80
C1_4_9	7025.30	1.73%	2134	-0.88%	-0.73%	56	C2_4_9	3910.90	1.00%	2809	-0.18%	0.69%	75
C1_4_10	7308.00	1.81%	2230	-6.47%	-6.36%	58	C2_4_10	3786.00	1.62%	3183	-0.59%	1.31%	84
R1_4_1	10357.00	0.14%	2429	0.11%	0.71%	49	R2_4_1	7523.30	0.94%	2605	4.36%	7.56%	70
R1_4_2	9121.40	0.94%	2521	-1.49%	-1.12%	54	R2_4_2	6488.40	1.33%	2917	8.31%	10.50%	90
R1_4_3	8198.50	1.29%	2425	-3.83%	-3.35%	58	R2_4_3	5400.90	0.62%	3835	5.87%	8.35%	92
R1_4_4	7803.40	0.84%	2173	-6.02%	-5.57%	60	R2_4_4	4223.40	0.49%	3465	2.89%	6.42%	98
R1_4_5	9314.40	0.98%	2244	-0.60%	-0.16%	49	R2_4_5	6568.00	0.58%	2244	5.03%	7.14%	80
R1_4_6	8673.60	0.89%	2462	-2.86%	-2.18%	54	R2_4_6	5848.70	0.98%	2588	3.95%	7.37%	93
R1_4_7	7956.00	1.51%	2446	-4.01%	-3.32%	58	R2_4_7	4938.60	0.62%	3882	3.86%	5.57%	95
R1_4_8	7694.40	1.47%	2428	-5.25%	-4.66%	62	R2_4_8	4011.00	0.32%	2787	3.75%	6.17%	100
R1_4_9	9026.60	0.85%	2243	-2.51%	-2.25%	50	R2_4_9	6081.30	0.80%	2473	5.65%	7.03%	88
R1_4_10	8494.00	1.15%	2191	-3.79%	-3.27%	52	R2_4_10	5675.70	1.12%	3242	4.59%	6.64%	94
RC1_4_1	8768.00	1.39%	2629	-1.81%	-1.49%	48	RC2_4_1	6148.10	0.07%	1678	1.77%	3.10%	66
RC1_4_2	8276.50	0.62%	2210	-3.75%	-3.37%	54	RC2_4_2	5407.50	0.01%	2133	2.69%	5.10%	79
RC1_4_3	7871.20	1.37%	2192	-3.59%	-3.23%	59	RC2_4_3	4573.00	0.19%	2096	3.07%	5.06%	91
RC1_4_4	7635.50	1.35%	2468	-3.56%	-3.22%	65	RC2_4_4	3606.10	0.23%	3369	2.71%	4.55%	99
RC1_4_5	8543.40	1.29%	2539	-3.52%	-3.11%	47	RC2_4_5	5401.70	0.49%	2591	1.61%	3.46%	77
RC1_4_6	8491.40	1.94%	2509	-3.18%	-2.68%	48	RC2_4_6	5342.10	0.24%	2368	2.00%	3.74%	72
RC1_4_7	8362.50	1.23%	2513	-3.88%	-3.39%	49	RC2_4_7	5002.90	0.34%	2450	2.41%	4.60%	85
RC1_4_8	8235.20	1.16%	2498	-4.51%	-3.92%	50	RC2_4_8	4712.50	0.34%	3008	3.61%	4.53%	88
RC1_4_9	8165.50	0.82%	2457	-4.34%	-4.15%	52	RC2_4_9	4513.90	1.02%	2750	1.77%	5.40%	88
RC1_4_10	7998.20	1.05%	2514	-4.00%	-3.69%	53	RC2_4_10	4284.20	0.68%	2919	3.08%	5.13%	93
Average	6410	0.81%	2354	-0.02%	1.24%	69							

Table 26.: Comparison between ALNS BCP and ALNS 75K for the GH400 instances. The column 'Best' denotes the best solution found by the ALNS BCP heuristic, $\bar{x}$ is the average gap, comparing with the column 'Best'. $\bar{t}$ is the average time. For ALNS 75K x^b denotes the gap between the best found solution by ALNS BCP and ALNS 75k.

The table shows that ALNS 75K generally finds the best solution, and are better for the C1, R1 and RC1 instances, which could be explained by the difference in the travel times used throughout the algorithm. ALNS BCP is however better for the C2, R2 and RC2 instances, and on average finds solutions that are better than the ones found by ALNS 75K.

7.2.3 600 Customers

When solving the C1, R1 and RC1 instances with 600 customers, too many unique routes were generated to consider them all in the set partitioning model, as CPLEX ran out of memory when making the model. Therefore these instances have only been solved using ALNS Reloc. Furthermore each of the instances have only been solved 5 times, due to the larger execution time. The result is shown in table 27

For 17 of the 30 C2, R2 and RC2 instances ALNS Reloc finds the best solution, and have the best average performance for 22 of these instances. This is most likely due to the set partitioning problem getting too hard, thus not being able to find an improved solution. This makes the algorithm return to the best found solution in the last part of the execution. When the number of customers is increased, so is the solution space, and thus it could be that the algorithm re-

		BCP		SPP		ALNS				BCP		SPP		ALNS	
	Best	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$		Best	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$	$\bar{x}$	$\bar{t}$
C1_6_1	14076.6	-	-	-	-	0.00%	1557	C2_6_1	7752.20	0.01%	3242	0.08%	3176	0.49%	2736
C1_6_2	13960.5	-	-	-	-	0.21%	1697	C2_6_2	7471.50	0.05%	3699	0.12%	3579	1.00%	2472
C1_6_3	14426.8	-	-	-	-	1.06%	1641	C2_6_3	7337.30	1.07%	5173	1.07%	4565	1.11%	2783
C1_6_4	15069.0	-	-	-	-	0.46%	1589	C2_6_4	7037.20	1.34%	6172	1.34%	5570	1.07%	2990
C1_6_5	14066.9	-	-	-	-	0.00%	1698	C2_6_5	7616.00	0.38%	5420	0.44%	4938	1.92%	2176
C1_6_6	14120.8	-	-	-	-	0.11%	1758	C2_6_6	7713.40	1.44%	4434	1.44%	3827	1.02%	2282
C1_6_7	14066.9	-	-	-	-	0.72%	1665	C2_6_7	7888.60	1.67%	4279	1.67%	3676	0.48%	2642
C1_6_8	15111.0	-	-	-	-	0.88%	1865	C2_6_8	7572.70	2.83%	4595	2.83%	3931	0.44%	2417
C1_6_9	14910.9	-	-	-	-	1.25%	1745	C2_6_9	7624.10	2.03%	4920	2.03%	4167	1.34%	2542
C1_6_10	15071.8	-	-	-	-	1.24%	1875	C2_6_10	7482.60	1.87%	5854	1.87%	5252	1.10%	2490
		-	-	-	-										
R1_6_1	22087.4	-	-	-	-	0.25%	2897	R2_6_1	15154.30	1.61%	5530	2.67%	4993	2.27%	5023
R1_6_2	19504.1	-	-	-	-	0.41%	1780	R2_6_2	13256.40	1.10%	6602	1.10%	5906	0.59%	6168
R1_6_3	18151.8	-	-	-	-	0.49%	1486	R2_6_3	10639.00	1.58%	5961	1.58%	5348	0.54%	5481
R1_6_4	17235.0	-	-	-	-	0.67%	1353	R2_6_4	7991.30	0.90%	6540	0.90%	5916	0.59%	6120
R1_6_5	19850.6	-	-	-	-	1.05%	1840	R2_6_5	14303.10	0.70%	5688	0.70%	5085	0.23%	4988
R1_6_6	18766.3	-	-	-	-	0.50%	1575	R2_6_6	12290.60	0.80%	6092	0.80%	5489	0.25%	6543
R1_6_7	17944.2	-	-	-	-	0.36%	1429	R2_6_7	10029.50	0.86%	6404	0.86%	5796	0.40%	6388
R1_6_8	17033.9	-	-	-	-	1.40%	1389	R2_6_8	7620.00	0.97%	6500	0.97%	5892	0.48%	7204
R1_6_9	19209.3	-	-	-	-	1.11%	1677	R2_6_9	13050.80	0.95%	5141	0.95%	4539	0.62%	4022
R1_6_10	18673.1	-	-	-	-	0.66%	1518	R2_6_10	12223.80	1.19%	5353	1.19%	4750	0.36%	6606
		-	-	-	-										
RC1_6_1	17822.0	-	-	-	-	0.55%	2260	RC2_6_1	12187.80	1.01%	4730	1.01%	4106	0.37%	4158
RC1_6_2	17000.9	-	-	-	-	0.40%	2399	RC2_6_2	10472.80	1.15%	5588	1.15%	4948	0.89%	5714
RC1_6_3	16535.3	-	-	-	-	0.62%	2099	RC2_6_3	9018.70	0.77%	7479	0.77%	6877	0.42%	6579
RC1_6_4	16201.8	-	-	-	-	0.84%	1952	RC2_6_4	7042.10	1.00%	8047	1.00%	7423	1.21%	6016
RC1_6_5	17741.9	-	-	-	-	0.31%	2220	RC2_6_5	11252.50	1.44%	8543	1.44%	7941	0.42%	5077
RC1_6_6	17600.3	-	-	-	-	0.49%	2099	RC2_6_6	11087.30	1.30%	6022	1.30%	5421	0.71%	4693
RC1_6_7	17286.0	-	-	-	-	0.80%	1949	RC2_6_7	10513.40	1.07%	6941	1.07%	6339	0.31%	5221
RC1_6_8	17232.9	-	-	-	-	0.43%	1656	RC2_6_8	9953.90	1.20%	7411	1.20%	6809	0.78%	6072
RC1_6_9	17141.4	-	-	-	-	0.35%	1628	RC2_6_9	9558.30	1.23%	7686	1.23%	7085	1.60%	6154
RC1_6_10	17067.4	-	-	-	-	0.70%	1562	RC2_6_10	9127.30	1.21%	7827	1.21%	7223	1.22%	6162
C1						0.59%	1709	C2		1.27%	4779	1.29%	4268	1.00%	2553
R1						0.69%	1694	R2		1.07%	5981	1.17%	5372	0.63%	5854
RC1						0.55%	1982	RC2		1.14%	7027	1.14%	6417	0.79%	5585
*1						0.61%	1795	*2		1.16%	5929	1.20%	5352	0.81%	4664
All		1.16%	5929	1.20%	5352	0.71%	3230								

Table 27.: Solution statistics for every Gehring/Homberger data set with 600 customers for ALNS BCP, ALNS SPP and ALNS Reloc. Best is the best solution found by one of the three algorithms, $\bar{x}$ is the average gap comparing with the best found solution. $\bar{t}$ denotes the time, in seconds, used. ALNS SPP and ALNS BCP failed to solve the C1, R1, RC1 instances due an out of memory error when making the set partitioning models in CPLEX. Each instance were solved 5 times for each of the algorithms

turns too often to the best found solution. Moreover, if the algorithm often returns to the best solution the temperature is increased too much (due to the previously mentioned error when reheating the temperature), which makes the algorithm more unpredictable and more random. Table 28 shows how many times out of 5 the BCP heuristic improved the solution.

	BCP Improvement				
	No	Yes	TLim	Avg. % edges	$\bar{t}$
C2_6_1		5		0.20%	66.4
C2_6_2	2	3		0.21%	120.5
C2_6_3	5		5	0.46%	607.8
C2_6_4	5		5	0.64%	602.8
C2_6_5	2	3	2	0.24%	481.5
C2_6_6	5		5	0.38%	607.3
C2_6_7	5		5	0.46%	602.4
C2_6_8	5		5	0.46%	664.1
C2_6_9	5		5	0.48%	753.0
C2_6_10	5		5	0.56%	601.7
R2_6_1	3	2	3	0.34%	536.6
R2_6_2	5		5	0.47%	696.3
R2_6_3	5		5	0.64%	612.4
R2_6_4	5		5	0.75%	624.2
R2_6_5	5		5	0.47%	603.0
R2_6_6	5		5	0.62%	602.2
R2_6_7	5		5	0.75%	608.1
R2_6_8	5		5	0.74%	608.0
R2_6_9	5		5	0.47%	601.2
R2_6_10	5		5	0.58%	602.4
RC2_6_1	5		5	0.36%	623.8
RC2_6_2	5		5	0.46%	640.0
RC2_6_3	5		5	0.67%	602.4
RC2_6_4	5		5	0.72%	623.4
RC2_6_5	5		5	0.47%	602.8
RC2_6_6	5		5	0.45%	600.7
RC2_6_7	5		5	0.54%	601.5
RC2_6_8	5		5	0.64%	602.0
RC2_6_9	5		5	0.59%	600.9
RC2_6_10	5		5	0.62%	604.0

Table 28.: Overview of BCP performance for the GH600 instances. $\bar{t}$ is the average time spent for the BCP heuristic, in seconds.

From table table 28 it is clear that the BCP problems are getting too hard, to be solved within 10 minutes, and an improved solution is only found in 13 out of the 150 tests. That is despite the fact that only 0.52% of the edges are considered on average.

Figure 14 shows how the run times scales with the number of customers. The data for ALNS SPP and ALNS BCP with 600 customers are based on data only solving the instances with wide time windows (C2, R2, RC2).
For the instances with 100, 200 and 400 customers the average run time scales linearly for the ALNS SPP and ALNS BCP. Hereafter the set partitioning problems become significantly harder as well as the ALNS run time almost triples, as can be seen from the ALNS Reloc graph. These are the two main reasons for the drastic increase in the run time.

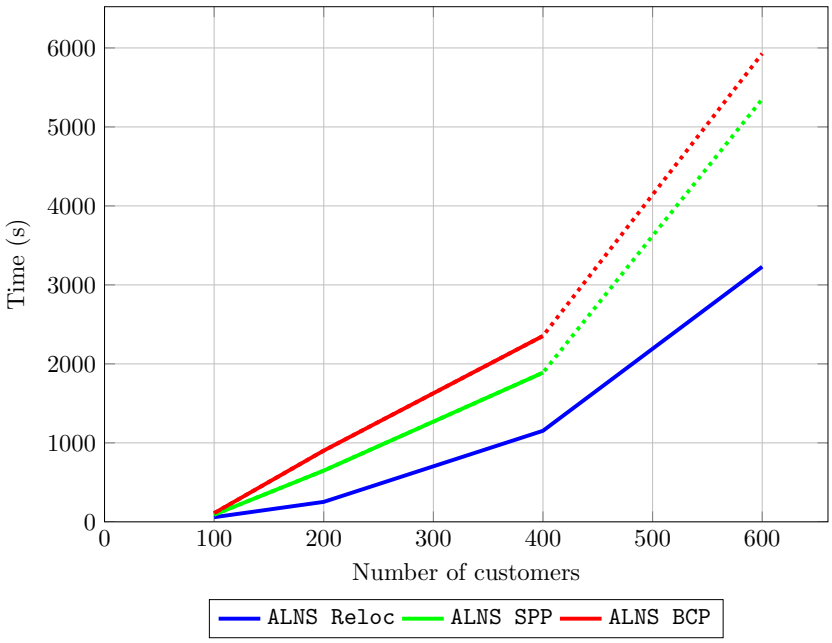


Figure 14.: Average run times for ALNS Reloc, ALNS SPP, ALNS BCP for 100, 200, 400 and 600 customers. The dotted lines are based on data only solving half of the instances.

8 CONCLUSION

This thesis have considered the VRPTW, and have tested different matheuristic approaches used within an Adaptive Large Neighbourhood Search algorithm. The results showed that combining a set partitioning method and a Branch Cut & Price heuristic gave the best results.

The final algorithm considers all of the generated routes by solving a set partitioning problem several times throughout the algorithm, and in the end a Branch Cut & Price heuristic is used as a postoptimisation step. The algorithm solved all of the 100 customer Solomon instances with an average best gap of 0.01%, and found the optimal solution for 49 out of the 56 instances, using on average 110 seconds. The average gap were 0.11%. The algorithm were also tested on the 200, 400 and 600 customer Gehring and Homberger instances, and showed promising results for the 200 customer instances, but the set partitioning and heuristic Branch Cut & Price methods failed to find improvements within the timelimit for most of the 400 and 600 customer instances.

By considering all of the generated routes through a set partitioning problem, the average gap for the 100 customer instances were more than halved, but incurred an additional 55% extra time. When the number of customers grew larger CPLEX v12.1 (released 2009) were unable to solve most of the problems within 100 seconds. This might be attributed to the larger number of routes generated. However, when solving the 100 customer instances on another computer using CPLEX version 12.4 the inclusion of the set partitioning approach only added 16.9% to the overall execution time, while maintaining the same overall quality of the solutions. Therefore I have reason to believe that CPLEX has gotten alot better at solving set partitioning problems, which would improve the solution quality of the larger instance as well as reduce the run times. The developers of CPLEX, IBM, supports the fact that they have gotten better at solving general MIP models since version 12.1 [16].

For the largest of the instances CPLEX ran out of memory when making the set partitioning models, and hence a filtering on the routes is necessary. Such a filtering could be not to include routes from solutions that, for example, are 20% worse than the current best solution. However doing so, the decision to include a route is taken based on the full solution, and not the route itself. Instead, dual information from the LP-solution to the set partitioning problem could be extracted and used to calculate the reduced costs of a route. Doing so the decision can be based on the actual routes and not the solution which it is a part of.

The results shows that the inclusion of the set partitioning method, improved the solution considerably for most of the medium-large to large sized instances, while only incurring little extra time. As many combinatorial optimisation problems can be formulated as a (generalised) set partitioning/set cover problem, this result have wider applicability than just for the Vehicle Routing Problems, and the set partitioning approach should be included in any metaheuristic where the solution quality is the key factor.

The Branch Cut & Price heuristic were successfully used as a postoptimisation step for the medium-large sized 100 customer instances, and also provided several improvements for the 200 customer instances. For the 400 and 600 customer instances there were little to no improvements

within the timelimit. However, the results proves the effectiveness of using Branch Cut & Price in a mathheuristic manner for further optimising good solutions, for problems where the exact methods are mature enough to efficiently solve the problems.

The Branch Cut & Price heuristic were applied solely as a postoptimisation step in the end of the algorithm. An idea for further research could be to include it in the ALNS algorithm, and feed any new feasible improved solution back to the ALNS algorithm. The ALNS heuristic could then be used to further explore these new neighbourhoods and use Branch Cut & Price again to possibly further improve any new best solutions found by ALNS. This way it would still be used as a postoptimisation step, but would have greater interaction with the ALNS algorithm, and thus the strength for both of these methods can be combined.

Most of the VRTPW heuristic approaches uses a prioritised evaluation function, first minimising the number of vehicles and then the total travel distance. This thesis only minimises the total travel distance. Thus making it hard to compare the described algorithm with the state-of-the art. An idea for further work could be to change the evaluation function in order to compare the algorithm with the best known solutions using the prioritised evaluation function.

The standard ALNS method were far slower than an ALNS 75K configuration of the algorithm presented in [23], even though ALNS 75K executed three times as many iterations. When developing the algorithm I have had a lot of focus on doing things the best way with respect to computational cost, but I am sure there exists parts that can be optimised. For example, whether or not a customer can be inserted into a route can be checked in constant time, instead of going through the full route, as is done in the current version. Furthermore the current algorithm runs sequentially, and could be updated to run in parallel to further utilise the power of modern computers.

A 3Opt* local search neighbourhood, were tested and it improved the quality of the solutions compared to using a more simple relocate neighbourhood. Though, the computational cost associated with this 3Opt* neighbourhood were far too high. The 2Opt* neighbourhood is commonly used as a local search method in state-of-the-art metaheuristic for the VRPTW, as described in the recent VRPTW survey [9]. The 2Opt* neighbourhood is part of the 3Opt* neighbourhood. An idea for further research could be to compare the 2Opt* neighbourhood with the 3Opt* neighbourhood, both with respect to the quality of the solutions and the incurred time.

In the parameter tuning phase the randomness had a huge impact on the quality of solutions. This might be attributed to the relative few number of times each instance were solved, and thus each instance should have been solved more times to give a clear view of the performance for the parameter value, and to even out the effect of the randomness. This was not done as time was rather spent on developing, and testing new solution methods. However, the parameter values still gave good results.

A LIST OF PARAMETERS

Method		Parameter	Description	Value	Source
Construction					
		α_1	Distance weight	1	[4]
		α_2	Time weight	0	[4]
		μ	Importance of savings in distance	0.9	[4]
		λ	Importance of distance to depot	0.75	[4]
Local Search					
		κ	When to use `Local Search`(x)	4%	Tuned
SA					
		w	Setting T_{start}	0.015	Tuned
		ζ	Cooling factor	0.9999	Tuned
Resetting Mechanism					
		γ	When to Reset	5000	Tuned
Destroys					
	Max Removal Percentage				
		π	Max Removal Percentage	35%	Tuned
	Worst Removal				
		p_{worst}	Probability factor	4	[19]
	Shaw Removal				
		φ	Distance weight	9	[29]
		χ	Time weight	3	[29]
		ψ	Demand weight	2	[29]
		ω	Same route weight	5	[29]
		p_{shaw}	Probability factor	4	[30]
Exact Methods					
	Set partitioning problem				
		ρ	When to the solve SPP model	2500	Tuned
	Probabilities				
		θ_1	Exact Insertion Probability	0%	Tuned
		θ_2	Exact Destroy/Insertion probability	0%	Tuned
		θ_3	SPInsert probaility	1%	Tuned
		θ_4	SPInsert/Destroy probaility	0%	Tuned
	SPInsert & SPDestroy/Insert				
		$\hat{k}$	Number of considered insert places	3	Ad hoc
		Numrounds	Number of generated solutions	50	Ad hoc
		NumDestroys	Number of destroys	10	Ad hoc
	Branc cut & Price				
		z	Number of best solutions used	10	Ad hoc
		τ	Max number of routes from LP-solution used	0.50	Ad hoc
Overall					
		σ_1	New global best reward	33	[29]
		σ_2	Better current solution reward	9	[29]
		σ_3	New current solution reward	13	[29]
		r	Reaction factor	0,1	[29]
		η	Segmentsize	125	Tuned
		ψ	Noise Parameter	0%	Tuned

Table 29.: List of Parameters. The value of a parameter is either tuned or found in the literature. Source shows whether or not the parameter is tuned, and if not the source for the value.

B A BRIEF NOTE ON REHEATING

As briefly noted in chapter 7, the reheating did not work as intended, this brief note explains what was actually done instead.
Let $\hat{T}$ denote the temperature when the best solution was found. When reheating it was the intention that the temperature should be reheated to 5% above this temperature

$$T \leftarrow \hat{T} \cdot 1.05$$

What was actually done was to remember which iteration the best solution were updated, and then use this to set the temperature.
Let i denote the current iteration, B the iteration in which the best solution were found, and ζ the cooling factor. The temperature have been cooled by ζ, $i - B$ times and hence,

$$T \leftarrow \frac{T}{\zeta^{i-B}} \cdot 1.05 \tag{9}$$

would set the correct temperature the first time the temperature is reheated. However this does not work if the best solution is not updated before reheating again, as it assumes that the only impact on the temperature is the cooling effect. This is not the case as the temperature has been reheated.

Figure 15 illustrates what happens when reheating the temperature. Consider a not so extreme case where the best solution is obtained in iteration 14000, which is illustrated by the red vertical line. The dotted curves illustrates what the temperature would have been considering the temperature now and then tracking backwards as described by eq. (9).

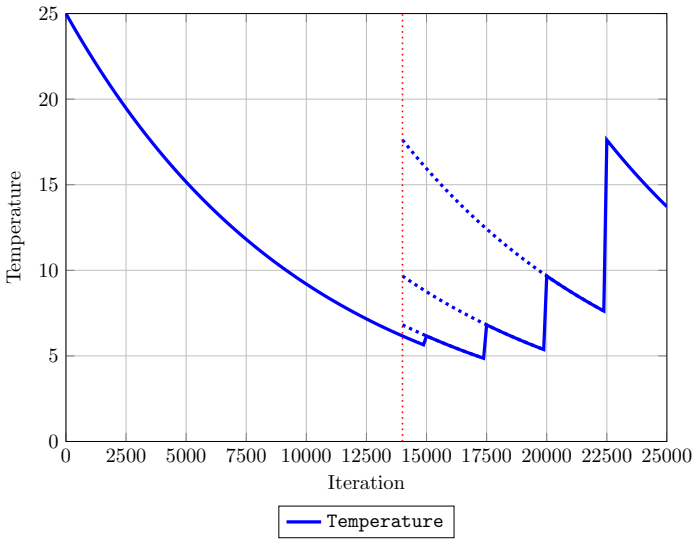


Figure 15.: Illustration of how the reheating works.

In the figure the temperature, when reheating, is set to the previous curves value at iteration

14000 (without adding the extra five percent). If the best solution is found in the beginning of the algorithm, the temperature changes are obviously more extreme, thus leading to the extreme change shown in fig. 13, shown on 56.
This error was revealed too late in order to rerun all the results. In order to have the results in chapter 7 based on the same algorithm, the results for the Solomon instances were not changed to how it would have looked if the reheating was implemented correctly. However the overall results for ALNS SPP on the Solomon instances, with the correct reheating, is shown in table 30.

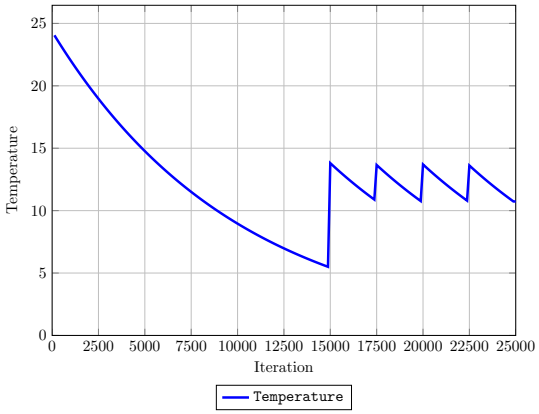
	Best Solution		**Average Solution**		**Optimals**	
	$\bar{x}$	$\max x$	$\bar{x}$	$\max x$	Total	Instances
ALNS SPP	0.03%	0.54%	0.13%	0.74%	353	42
ALNS SPP correct reheat	0.05%	0.48%	0.16%	1.77%	366	44

Table 30.: Comparison between ALNS SPP and ALNS SPP with the correct reheating of the algorithm for the Solomon instances. $\bar{x}$ denotes the average gap and $\max x$ is the worst achieved solution both when considering the best solution, or the average solution. The Optimal columns denotes how many optimal solutions were found, and for how many instances the algorithm found an optimal solution.

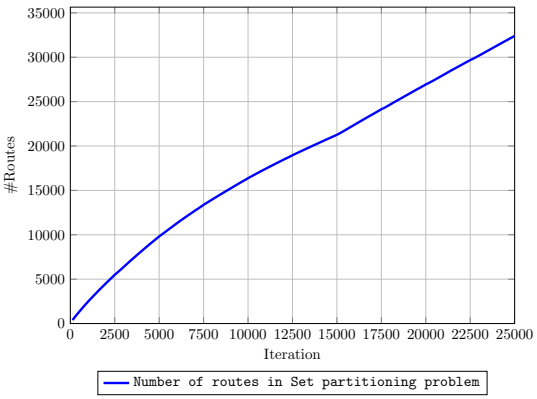
The small differences shown in table 30 can be explained by the random factor, and if anything the old, faulty, way of reheating is best.

Figure 16 shows the development of the algorithm, with the reheating fixed. The figure almost shows the same picture as fig. 13 on page 56, but without the drastic jumps in the temperature. In the end of the algorithm fewer routes are added to the set partitioning problem, and the `Greedy insertion` and `Worst removal` still has a comeback, but not a big as before. For the figure showing the gap, there is not any noticeable difference compared to before.

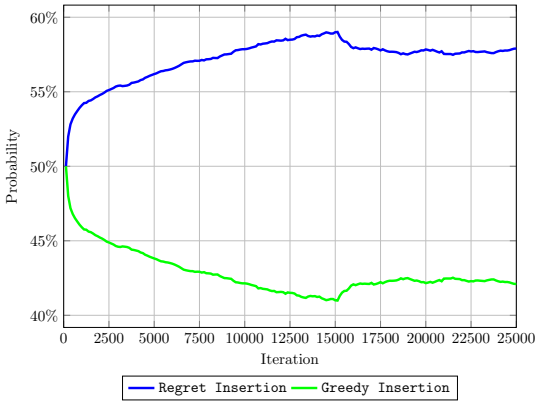
For the Solomon instances the best obtained solution were on average found in iteration 11500, leading to the extreme case shown in fig. 13a. For the Gehring & Homberger instances with 200 customers the best obtained solution were on average found in iteration 12200 thus a similar steep increase in the temperature is expected for those instances. However, for the Gehring & Homberger instances with 400 customers the best obtained solution were on average found in iteration 17700, and the impact is thus expected to be less, and even less for the Gehring & Homberger instances with 600 customers as the best obtained solution on average were found in iteration 20500.



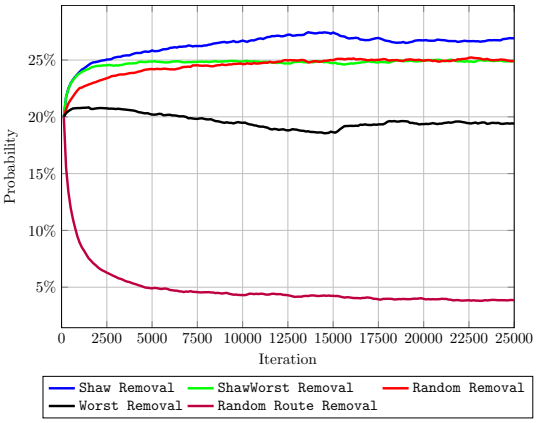
(a) The temperature throughout the algorithm.



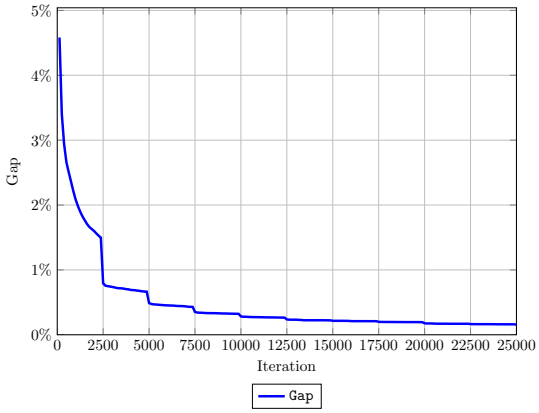
(b) Number of generated unique routes in the algorithm



(c) Probability for the insertion methods throughout the algorithm



(d) Probability for the destroy methods throughout the algorithm



(e) The gap for the current best solution throughout the algorithm.

Figure 16.: Detailed illustration of the development of the algorithm, with the reheating fixed.

C LEARNING OBJECTIVES

DTU's Study handbook defines the following general Learning Objectives for a master thesis.[1]

An MSc from DTU:

- Can identify and reflect on technical scientific issues and understand the interaction between the various components that make up an issue
- Can, on the basis of a clear academic profile, apply elements of current research at international level to develop ideas and solve problems
- Masters technical scientific methodologies, theories and tools, and has the capacity take a holistic view of and delimit a complex, open issue, see it in a broader academic and societal perspective and, on this basis, propose a variety of possible actions
- Can, via analysis and modelling, develop relevant models, systems and processes for solving technological problems
- Can communicate and mediate research-based knowledge both orally and in writing
- Is familiar with and can seek out leading international research within his/her specialist area.
- Can work independently and reflect on own learning, academic development and specialisation
- Masters technical problem-solving at a high level through project work, and has the capacity to work with and manage all phases of a project - including preparation of timetables, design, solution and documentation

1 See http://shb.dtu.dk/Default.aspx?documentid=3200&Language=en-GB&lg=&version=2013/2014, section 11.

BIBLIOGRAPHY

[1] G.B. Alvarenga, G.R. Mateus, and G. de Tomi. A genetic and set partitioning two-phase approach for the vehicle routing problem with time windows. Computers & Operations Research, 34(6):1561–1584, Jun 2007.

[2] Claudia Archetti and M. Grazia Speranza. A survey on matheuristics for routing problems.

[3] Claudia Archetti, M. Grazia Speranza, and Martin W. P. Savelsbergh. An optimization-based heuristic for the split delivery vehicle routing problem. Transportation Science, 42(1):22–31, Feb 2008.

[4] Karen Arentoft and Jonas Christensen. An adaptive large neighborhood search for vrptw. 2013.

[5] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New route relaxation and pricing strategies for the vehicle routing problem. Operations research, 59(5):1269–1283, 2011.

[6] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. New state-space relaxations for solving the traveling salesman problem with time windows. INFORMS JOURNAL ON COMPUTING, 24(3):356–371, 2012.

[7] Olli Braysy and Michel Gendreau. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. Transportation Science, 39(1):104–118, Feb 2005.

[8] Guy Desaulniers, François Lessard, and Ahmed Hadjar. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. Transportation Science, 42(3):387–404, 2008.

[9] Guy Desaulniers, Oli B.G Madsen, and Stefan Ropke. Vehicle routing problems with time windows. In Paolo Toth and Daniele Vigo, editors, Vehicle routing: Problems, methods, and applications, second edition, MOS-SIAM Series on Optimization, chapter 5, pages 121–162. SIAM, 2014.

[10] KarlF. Doerner and Verena Schmid. Survey: Matheuristics for rich vehicle routing problems. In MariaJ. Blesa, Christian Blum, Gunther Raidl, Andrea Roli, and Michael Sampels, editors, Hybrid Metaheuristics, volume 6373 of Lecture Notes in Computer Science, pages 206–221. Springer Berlin Heidelberg, 2010.

[11] Moshe Dror. Note on the complexity of the shortest path models for column generation in vrptw. Operations Research, 42(5):977–978, 1994.

[12] Matteo Fischetti and Andrea Lodi. Local branching. Mathematical Programming, 98(1-3):23–47, 2003.

[13] Roberto De Franceschi, Matteo Fischetti, and Paolo Toth. A new ILP-based refinement heuristic for vehicle routing problems. Math. Program., 105(2-3):471–499, Feb 2006.

[14] H. Gehring and J Homberger. A parallel two-phase metaheuristic for routing problems with time windows. 1999.

[15] Bruce Golden, S. Raghavan, and Edward A. Wasil, editors. The Vehicle Routing Problem: Latest Advances and New Challenges. Operations research/Computer science interfaces series, 43. Springer, 2008.

[16] IBM. Cplex mip optimizer performance improvement since 2000. http://www-01.ibm.com/software/commerce/optimization/cplex-performance/.

[17] James P. Kelly and Jiefeng Xu. A set-partitioning-based heuristic for the vehicle routing problem. INFORMS Journal on Computing, 11(2):161–172, May 1999.

[18] Glaydston Mattos Ribeiro and Gilbert Laporte. An adaptive large neighborhood search heuristic for the cumulative capacitated vehicle routing problem. Computers & Operations Research, 39(3):728–735, Mar 2012.

[19] Niels Peter Meyn Milthers. Solving vrp using voronoi diagrams and adaptive large neighborhood search. Master's thesis, University of Copenhagen, Department of Computer Science, 2009.

[20] A. MINGOZZI N. CHRISTOFIDES and P. TOTH. The vehicle routing problem. In P. Toth N. Christofides, A. Mingozzi and C. Sandi, editors, Combinatorial Optimization, chapter 11, pages 315–338. Wiley, 1979.

[21] Puca Huachi Vaz Penna, Anand Subramanian, and Luiz Satoru Ochi. An iterated local search heuristic for the heterogeneous fleet vehicle routing problem. Journal of Heuristics, 19(2):201–232, 2013.

[22] Sandro Pirkwieser and Gunther R. Raidl. Matheuristics for the periodic vehicle routing problem with time windows. 2010.

[23] David Pisinger and Stefan Ropke. A general heuristic for vehicle routing problems. Computers & Operations Research, 34(8):2403 – 2435, 2007.

[24] Eric Prescott-Gagnon, Guy Desaulniers, and Louis-Martin Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. Networks, 54(4):190–204, Dec 2009.

[25] Jakob Puchinger and GuntherR. Raidl. Combining metaheuristics and exact algorithms in combinatorial optimization: A survey and classification. In Jose Mira and Jose R. Alvarez, editors, Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, volume 3562 of Lecture Notes in Computer Science, pages 41–53. Springer Berlin Heidelberg, 2005.

[26] Roberto Roberti. Exact Algorithms for Different Classes of Vehicle Routing Problems. PhD thesis, University of Bologna, 2012.

[27] Y. Rochat and É.D. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. Journal of Heuristics, 1(1):147–167, 1995.

[28] Stefan Ropke. Branching decisions in branch- and-cut-and-price algorithms for vehicle routing problems. 2012.

[29] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science, 40(4):455–472, Nov 2006.

[30] Paul Shaw. A new local search algorithm providing high quality solutions to vehicle routing problems. 1997.

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

[32] Anand Subramanian. Heuristic, exact and hybrid approaches for vehicle routing problems. PhD thesis, Universidade Federal Fluminense, 2012.

[33] Anand Subramanian, Eduardo Uchoa, and Luiz Satoru Ochi. A hybrid algorithm for a class of vehicle routing problems. Computers & Operations Research, 40(10):2519 – 2531, 2013.

[34] Paolo Toth and Daniele Vigo, editors. The Vehicle Routing Problem. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[35] U.Mahir Yildirim and Bülent Çatay. A time-based pheromone approach for the ant system. Optimization Letters, 6(6):1081–1099, 2012.

[36] Umman Mahir Yildirim and Bulent Çatay. A parallel matheuristic for solving the vehicle routing problems. In Jorge Freire de Sousa and Riccardo Rossi, editors, Computer-based Modelling and Optimization in Transportation, volume 262 of Advances in Intelligent Systems and Computing, pages 477–489. Springer International Publishing, 2014.