A Vehicle Routing Problem with Time Windows and Shift Time Limits

Irina Kupriyanova – s000328

March 2006

Master Thesis
IMM, DTU

Abstract

In this thesis a vehicle routing problem with time windows and shift time limits is investigated. An additional specific feature of the problem is a non homogeneous fleet with a limited number of vehicles of each type.

A two-phase solution method is developed and implemented. The first phase deals with construction of a large set of good quality routes. In the second phase a Lagrangian heuristic is used to solve a mixed set covering/packing problem where columns of the constraint matrix correspond to the routes generated in the first phase.

The developed solution approach is tested on a number of problems of different size. The computational results show that optimal solutions can be found for the set covering/packing problems of small size, i.e. 120 rows and up to 500 columns. As the problem size increases the performance of the Lagrangian heuristic becomes worse.

Keywords: *Vehicle routing*, *time windows*, *shift time limits*, *Lagrangian relaxation*, *set covering*, *set packing*.

Contents

Preface

This project is a Master Thesis required for obtaining the degree of Master of Science in Engineering. The project is carried out at Informatics and Mathematical Modelling (IMM), Technical University of Denmark in cooperation with Transvision A/S. The supervisor at IMM, DTU is Jesper Larsen and the supervisor at Transvision A/S is Jakob Birkedal Nielsen. The project has been conducted from 15 September 2005 to 15 March 2006.

I am thankful for the opportunity of carrying out this project in cooperation with Transvision A/S. I would like to thank my supervisors, especially Jesper Larsen for his ideas and suggestions throughout the project.

I also appreciate the help I received from some of the operations researchers around the world, namely Martin Savelsbergh and Ann Melissa Campbell.

Irina Kupriyanova

List of Tables

List of Figures

Chapter 1

Introduction

In the following, an introduction to the problem considered in this thesis, the aim of the project and the structure of the report are presented.

1.1 Motivation and Background

In the past years the distribution cost have become one of the largest cost components for all businesses. The distribution costs can account for almost half of the total logistic costs and even more in some industries such as food and drink business [6]. Thus, how a company transports and delivers its products is essential to its ability to maintain and increase profitability.

A key element of any distribution system is routing of vehicles through a set of customers requiring service, i.e. deliveries of products. Careful and effective route planning has therefore a crucial impact on a company's distribution costs and hence on profitability.

The company considered in this thesis is one of the largest retails distributors in Denmark with its 5 large terminals and more than 3000 customers throughout the country. The challenge for the company is to coordinate a huge production with consecutive distribution of the products to the customers subject to the strict delivery time restrictions imposed by the customers. Obviously, solving transportation problems of this size and complexity is not an easy task and will not be particularly efficient if done manually.

For a number of years development of transportation solutions for the company considered in this thesis has been outsourced to Transvision A/S. One of Transvision's standardised products *Route Planner* is now successfully used by the company leading to the significant savings in distribution costs.

Due to the complexity of the problem, there are still some issues that can be investigated further in order to produce better solutions and achieve even larger savings. One of such issues is addressed in this thesis and can be shortly described as follows: The daily distribution of company's products is based on the concept of master plans or fixed routes, constructed based on the available vehicle fleet and geographical distribution of demand/customers. The master plans are revised and can be changed once or twice per year based on the changes in customer demand. A revision like this results in significant savings in transportation costs. The question considered in this thesis is whether a considerably better solution, hence larger savings, can be obtained by revising the master plans more often, e.g. on a daily basis.

In the following section the problem of constructing master plans and the issue mentioned above will be described in more details.

1.2 Problem Description

The problem considered in this thesis is a real-life problem faced by a number of Transvision's clients in connection with daily distribution of products.

The company has a fleet of vehicles available for making deliveries to the customers. The routes for the vehicles are constructed based on the information about geographical distribution of demand/customers with the main objective of minimizing the total transportation costs. The secondary objective is maximizing the utilization of the available vehicles. A *route* is a trip from the depot to a sequence of customers and back to the depot. Each route must satisfy a number of different restrictions and requirements. Obviously, the customer demand must be satisfied, and the capacity of a vehicle assigned to a route must not be exceeded. Each customer has to be serviced within a prespecified time interval, called *time window*. A similar time window restriction apply to the vehicles: Each vehicle is only available for making deliveries during some prespecified hours of the day. In the following, this type of time windows will be referred to as *vehicle availability windows*. Furthermore, there are limits on how many hours the drivers can work – *shift time limits*. This type of restriction can be present due to safety reasons or scheduling reasons.

The routes are constructed once or twice per year and used as basis for the daily distribution of products. The routes are fixed and are not changed until the next revision, which is why they are called *master plans*. Based on the above definitions the problem of constructing master plans can be

classified as a well-known problem called the vehicle routing problem with time windows (VRPTW) and complicating constraints.

On the day of operation the routes are executed according to master plans without any adjustments to reflect the possible changes in demand. The change in demand considered in this problem is cancellation of deliveries to some customers. It should be noted, that in this context possible changes in demand are the changes known prior to the day of operation, e.g. the customers calling in the day before and cancelling their deliveries. Operational issues such as disruption management, handling additional delivery requests on the day of operation etc. are not considered in this thesis.

The current practice is that the drivers get their route plans on the day of operation. As mentioned above, if a number of customers have cancelled their deliveries the day before, these customers will still be included in the route plans given to the drivers. The vehicles will then drive past these 'empty' customers without stopping. In some cases this will not affect the effectiveness of the route. For example, if it is impossible to avoid driving past an empty customer due to topological/geographical issues.

However, in most cases the current practice will result in less effective routes than if the routes were adjusted to reflect the changes. This happens if the empty customers can be eliminated from their respective routes hereby achieving a shorter route length or savings in travel time.

The question is now whether another approach should be used as basis for the daily distribution. One possibility could be to adjust the master plans according to the changes in customer demand prior to the day of operation. Another option is to construct the routes from scratch every day based on the updated information about customer demand on that particular day.

The main objective of this thesis is to develop methods that provide good solutions to the vehicle routing problem arising in connection with a daily distribution of products. The new solutions will then be compared to the existing practice to answer the question stated in the above.

1.3 Outline of the Report

The structure of the report is as follows. In chapter 2 the vehicle routing problem with time windows is discussed, and the mathematical model for the problem is presented. The chapter also includes a review of the existing solution methods for the VRPTW.

In chapter 3 the model presented for the VRPTW is extended to handle the complicating constraints, such as shift time limits and vehicle availability windows. A two-phase solution approach used to solve the problem considered in this thesis is briefly introduced. A case study data provided by Transvision A/S is described in chapter 4.

The first phase of the solution approach is discussed in chapter 5. The aim of this chapter is to describe a route construction heuristic and route improvement techniques used in this thesis. Chapter 6 describes the development of the Lagrangian heuristic used in the second phase of the solution approach.

Chapter 7 presents an approach used to generate master plans and customer demand variations. The chapter also includes a description of the test problems generated based on the case study data presented in chapter 4.

Chapter 8 presents the computational experiments conducted in this thesis. Firstly, the performance of the Lagrangian heuristic both on the generated test problems and the case study problem is discussed. Secondly, the master plans are compared to the solutions generated based on the changes in customer demand.

The future work is discussed in chapter 9 and in chapter 10 the conclusion is presented.

Chapter 2

Theory

In this section the Vehicle Routing Problem with Time Windows is described, and the model for the problem is presented. Furthermore, possible solution methods are discussed based on a literature review.

2.1 The Vehicle Routing Problem with Time Windows

The Vehicle Routing Problem with Time Windows is a well-known problem which has received a considerable attention in recent years. This is due to the fact that the VRPTW is a useful abstraction of many real-life problems dealing with distribution of goods or services. Furthermore, finding good solutions to this problem contributes to reducing transportation and distribution costs of a company.

2.1.1 Problem Formulation

The following definition can be used to describe the problem [31]:

> *The VRPTW is concerned with the design of minimum-cost vehicle routes, originating and ending at a central depot. The routes must service a set of customers with known demands. Each customer is to be serviced exactly once during the planning horizon and customers must be assigned to the vehicles without exceeding vehicle capacities. Furthermore, each customer must be serviced during allowable delivery times or time windows.*

In the following, the definitions and assumptions used to model the problem will be presented.

The problem can be represented as a connected graph $G = (V, E)$ consisting of a set of $n + 2$ nodes. $V = N \cup \{0\} \cup \{n + 1\}$ where nodes 0 and $n + 1$ refer to the depot and N is a set of customers indexed by $i = 1, \ldots, n$ or $j = 1 \ldots n$. For each edge $(i, j) \in E$ a travel time t_{ij} and the associated travel cost c_{ij} are given. In the following it is assumed that the travel cost c_{ij} between customers i and j is proportional to the travel time t_{ij} between these two customers.

Each customer i has a prespecified demand q_i. The service time for customer i is denoted s_i and servicing customer i must begin within a time window $[E_i, L_i]$, where E_i is the earliest allowable time and L_i is the latest allowable time.

In some cases it is allowed to begin servicing a customer after its time window has closed by paying a certain cost [35]. It can also be allowed to begin servicing the customer before the time window opens if the vehicle arrives at customer earlier than E_i. In this case the time windows are said to be *soft*.

For the problem considered in this thesis, the concept of *hard* time windows is applied: If the vehicle arrives at the customer site before the earliest allowable time, it must wait until the time window opens. Hereby a delay or waiting time is incurred. Arrival after the latest allowable time is not permitted and results in an infeasible solution.

All problem parameters such as customer demand and time windows are positive constants and are assumed to be known with certainty. Furthermore, split deliveries and multiple visits are not allowed, i.e. each customer must be serviced by exactly one vehicle.

In many real-life applications the primal objective of the problem is to find the minimal number of tours for a set of vehicles, i.e. to minimize the number of vehicles required to service all customers. In the problem considered in this project, the number of vehicles is fixed a-priori. Furthermore, it is assumed that the fleet of vehicles is not homogeneous. Let M denote the set of all vehicles indexed by $k = 1, \ldots, m$ and let Q_k be the capacity of vehicle k. The objective is now to design a set of minimum-cost routes for these vehicles.

To construct the routes, a sequence of customers to service must be determined for each vehicle, and for each customer on a route the time to begin delivery must be specified. Thus, the following decision variables will be used

in the model:

$x_{ijk} = \begin{cases} 1 & \text{if vehicle } k \text{ travels directly from customer } i \text{ to customer } j \\ 0 & \text{otherwise} \end{cases}$

t_i – the time to begin delivery at customer i

st_k – departure time of vehicle k from the depot

f_k – arrival time of vehicle k at the depot

It is assumed that the planning period begins at time 0 and ends at time T.

2.1.2 The Mathematical Model

Based on the problem formulation in the previous section, the mathematical model for the problem can now be formulated as follows.

$$\min \quad \sum_{(i,j)\in E} \sum_{k\in M} c_{ij} \cdot x_{ijk} \tag{2.1}$$

$$\text{s.t.} \quad \sum_{j\in V} x_{ijk} - \sum_{j\in V} x_{jik} = 0 \qquad \forall i \in N, \forall k \in M \tag{2.2}$$

$$\sum_{i\in N} x_{0ik} = 1 \qquad \forall k \in M \tag{2.3}$$

$$\sum_{i\in N} x_{i,n+1,k} = 1 \qquad \forall k \in M \tag{2.4}$$

$$\sum_{k\in M} \sum_{j\in V} x_{ijk} = 1 \qquad \forall i \in N \tag{2.5}$$

$$\sum_{i\in N} q_i \sum_{j\in V} x_{ijk} \leq Q_k \qquad \forall k \in M \tag{2.6}$$

$$t_i + s_i + t_{ij} - C(1 - x_{ijk}) \leq t_j \qquad \forall i, j \in N, \forall k \in M \tag{2.7}$$

$$st_k + t_{0i} - C(1 - x_{0ik}) \leq t_i \qquad \forall i \in N, \forall k \in M \tag{2.8}$$

$$t_i + s_i + t_{in+1} - C(1 - x_{i,n+1,k}) \leq f_k \qquad \forall i \in N, \forall k \in M \tag{2.9}$$

$$t_i \geq E_i \qquad \forall i \in N \tag{2.10}$$

$$t_i \leq L_i \qquad \forall i \in N \tag{2.11}$$

$$st_k \geq 0 \qquad \forall k \in M \tag{2.12}$$

$$f_k \leq T \qquad \forall k \in M \tag{2.13}$$

$$x_{ijk} \in \{0, 1\} \qquad \forall i, j \in V, \forall k \in M \tag{2.14}$$

The objective (2.1) is to minimize the total costs of the routes. Constraints (2.2) are flow constraints ensuring that if a vehicle visits a customer, it has to

leave the customer again. Constraints (2.3)-(2.4) state that each route starts and terminates at the depot. Constraints (2.5) specify that each customer is visited by exactly one vehicle. Constraints (2.6) are the capacity constraints for the vehicles.
Constraints (2.7) ensure the arrival time compatibility between a pair of customers and work as follows: Let C be a large constant. If $x_{ijk} = 0$, the constraint is not binding. However, if customer j is serviced directly after customer i, i.e. $x_{ijk} = 1$, the constraint becomes binding and ensures that the condition $t_i + s_i + t_{ij} \leq t_j$ holds.
Constraints (2.8)-(2.9) are defined in a similar way for the depot and the first customer on each route and for the depot and the last customer on each route, respectively. Constraints (2.10)-(2.11) are time window constraints. Finally, constraints (2.12)-(2.13) ensure that vehicles are used within the planning period, and constraints (2.14) impose binary restrictions on the x-variables.

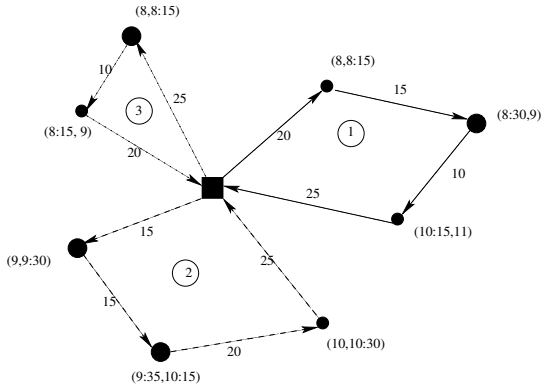
Discussion of the objective function

In real-life applications different objective functions are used when solving the VRPTW. In the following, several optimization criteria and their impact on the obtained solution are briefly introduced.

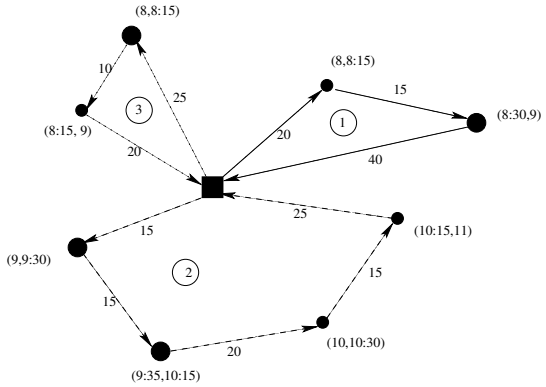
As it was mentioned in section 2.1.1, one of the objective function criteria can be to minimize the number of vehicles required to serve all the customers. This criterion will typically be used when some fixed cost must be paid for each extra vehicle.

In the model presented in the above, the objective is to minimize the total cost of the routes. As it is assumed that the cost of a route is proportional to its length (distance or time), this objective corresponds to minimizing the total travel distance or travel time. Another criterion is minimizing the route duration or the total used time. It should be noted, that the terms *total travel time* and *total used time* are not equivalent. The difference is that the total used time also includes the total waiting time on the routes. Thus, these two optimization criteria can produce different solutions. A solution obtained when minimizing total travel time can contain a lot of waiting time at the customers. On the other hand, a solution obtained by minimizing the total time can suggest a large amount of extra driving. The situation is illustrated in figure 2.1, where the two different solutions for a small problem are displayed. The circles in the figure correspond to the customers. It is assumed that the service time is set to 10 minutes for the customers indicated by the small circles and to 20 minutes for the larger

circles. The travel distances and time windows for each customer are shown in the figure.



(a) Total travel time: 200 minutes, waiting time: 65 minutes.



(b) Total travel time: 225 minutes, waiting time: 0 minutes.

Figure 2.1: Two different solutions produced by employing different objective functions. In figure a) the total travel time is minimized, in figure b) the total used time, i.e. route duration, is minimized.

The solution in figure 2.1.a is obtained by minimizing the total travel time. The solution in figure 2.1.b is obtained by minimizing the total used time. Consider route 1 in the solution in figure 2.1.a. There is a significant gap between the time windows for the first two customers and the last customer on the route. This results in 65 minutes of waiting time on this route. The

other two routes can be executed without any waiting time. The total travel time for this solution is 200 minutes, and the total used time can be calculated as the sum of travel, waiting and service times, i.e. $200+65+4\cdot10+4\cdot20 = 385$ minutes. In the solution in figure 2.1.b route 3 is unchanged and routes 1 and 2 are modified to avoid the long waiting time. It is assumed that there is enough capacity on the vehicle assigned to route 2 to handle the extra request. The new solution has a total travel time of 225 minutes, but now all the routes can be executed without any waiting time. The total travel time for this new solution is $225 + 4 \cdot 10 + 4 \cdot 20 = 345$ minutes.

Generally, long waiting times should be avoided for reasons such as driver wages, driver satisfaction or the condition of the products (e.g. cooling products while waiting). On the other hand, longer travel times imply longer travel distances, and this is not desirable due to the fuel cost. It is the responsibility of the distribution manager to decide which objective is to be used as optimization criterion. In many real-life applications, a multiple-criteria objective function is used – i.e. a weighted sum of several components. For the model presented in the previous section, the objective function could be modified as follows:

$$\min \quad \alpha \sum_{(i,j)\in E} \sum_{k} t_{ij} \cdot x_{ijk} + \beta \sum_{k} (f_k - st_k) \tag{2.15}$$

where α and β are positive constants. The first term of the function corresponds to the total travel time and the second term is the total time, i.e. the sum of all route durations. Alternatively, a term penalizing waiting time at each customer could be included in the objective function, i.e. $\gamma \sum_{i\in N} max\{0, E_i - t_i\}$. Finally, if the policy of soft time windows is employed, the objective function could also contain a term penalizing the violation of the time windows, i.e. $\delta \sum_{i\in N} max\{0, t_i - L_i\}$.

2.2 Literature Review - Solution Methods

Due to its practical significance, the VRPTW has been the subject of intensive research for both heuristic and exact optimization approaches.

Among the exact methods the best results are obtained by approaches based on branch-and-bound, column generation and methods based on Lagrangian decomposition. According to Kallehauge et al. [23] the exact methods are able to solve problems with up to 100 customers. Only two larger problems have been to date solved to optimality: two problems introduced by Gehring and Homberger with 400 and 1000 customers respectively.

The VRPTW is proven to be NP-hard [38], and solving such problems to optimality by use of exact methods is usually very time consuming and often impossible. Thus, most approaches for the large problems are based on heuristics. Heuristic methods can broadly be classified into two main classes: *classical heuristics* developed in 1960-1990 and *metaheuristics* which have become more popular in the last years [26]. An extensive review of methods from both categories can be found in the article of Bräysy and Gendreau [6, 7]. In the following, some of the solution methods will be briefly introduced starting with classical heuristics.

Classical heuristics

Most standard route construction and route improvement heuristics fall into the first category of classical heuristics.

Solomon [34] describes different construction heuristics for the VRPTW: The first two are an extension of the well-known savings method proposed by Clark and Wright and the time-oriented nearest neighbour method. Next, Solomon introduces three different sequential insertion methods and a time-oriented sweep method. The most successful of the three insertion heuristics is called I1, which aims to construct routes which maximize the benefit of inserting a customer into a partially constructed route compared to serving the customer directly from the depot.

Potvin and Rousseau [28] introduce a parallel version of Solomon's insertion heuristic I1, where a set of m routes is initialized at once. The sequential version of the insertion heuristic is used to determine the initial number of routes and the set of seed customers. Foisy and Potvin [17] implemented the method on parallel hardware consisting of 2–6 Sun 3 workstation transputers. The parallelism was utilized in the calculation of the insertion cost for each customer. The authors conclude that the reduction in the total computing time is linear with the number of processors for the distributed part of the heuristic algorithm.

Iannou et al. [22] use the generic insertion framework proposed by Solomon but with different customer selection and insertion criteria. The basic idea is to minimize the impact the insertion of customer u has on the customer itself, on the customers already in the route and on all the unrouted customers. The method is tested on 56 problem instances proposed by Solomon. Compared to the results produced by the original Solomon's heuristic I1, Iannou et al. produce better results for these problems, though at the cost of higher computation times.

The construction heuristics can be considered as simple and fast methods to generate feasible solutions to the VRPTW. However, the quality of solutions produced by these simple procedures is often much worse than more sophisticated approaches. A common approach is a two-phase method, where a construction heuristic is applied first to construct an initial set of routes and an improvement heuristic is applied afterwards to obtain a better solution.

Potvin and Rousseau [29] compare different edge-exchange heuristics for the VRPTW such as 2-opt, 3-opt and Or-opt and introduce a new 2-opt* operator. The 2-opt* operator is similar to the original 2-opt method, but is applied to two different routes. The test results show a hybrid method based on the oscillation between the 2-opt* and Or-opt approaches to be the most successful. The initial routes are created with Solomon's I1 heuristic.

Russel [31] proposes a hybrid algorithm which embeds the route improvement procedures within the route construction process. The parallel insertion heuristic similar to that of Potvin and Rousseau (1993) is applied to construct the routes. An exchange operator which swaps two customers between different routes is then performed on the partially constructed solution after f customers have been inserted. The size of f affects both the solution quality and the computation time. The method is tested on Solomon's test problems, and the experimental results show that the improvement procedure after every $f = 0.10n \ldots 0.16n$ customers added during route construction yields the best results.

Hamacher and Moll [21] describe a heuristic for a real-life VRP with narrow time windows in the context of delivery of groceries to restaurants. The algorithm consists of two stages. In the first stage the clustering procedure based on the minimum spanning tree algorithm is performed, and customers are partitioned into regionally bounded sets. In the second stage route construction and improvement procedures are performed in each cluster. Routes are constructed based on the simple cheapest insertion method and the Or-opt operator is applied as local improvement method.

Shaw [33] presents a large-neighbourhood search (LNS) based on rescheduling some of the customers using constraint programming (CP) techniques. The search is performed by randomly choosing a set of related customers and removing them from the routes. The removed customers are then rescheduled by use of CP coupled with a branch-and-bound method. Due to the large computational requirements, this approach can be applied only to problems with a relatively low number of customers per route.

Schrimpf et al. [32] introduce a method close to the LNS that is named 'ruin and recreate'. The method works as follows: The initial solution is

'ruined' by removing a set of customers from the solution according to one of the proposed types of ruin-moves, i.e. radial, sequential or random ruin. In the next step a new solution is constructed by inserting the removed customers into the routes according to the best insertion criterion. The insertion procedure must not violate any constraints so that the obtained solution is feasible. During the search solutions that worsen the objective function are accepted if the deterioration is within a certain threshold. The 'ruin and recreate' method has performed quite well on solving the Solomon problems in terms of solution quality: The authors compare their results to the best known solutions obtained by the heuristic approaches and to the solutions obtained by the tabu search approach of Rochat and Taillard [30]. The methods are tested on 56 problem instances proposed by Solomon. Compared to the heuristic approaches a better solution is found for 36 problems, and for 12 problems a solution value is the same as the best known one. Comparison to the tabu search described by Rochat et al. shows, that the 'ruin and recreate' method finds the same results in 24 cases and better results in 31 cases.

Bräysy [8] describes several local search heuristics using a new three-phase approach for VRPTW. In the first phase, several initial solutions are constructed by different construction techniques. In the second phase, a new ejection chain-based approach is used to reduce the number of routes. The ejection chain approach has been originally developed by Glover and is described in [6]. Finally, Or-opt exchanges are used in the third phase to minimize the total travel distance. According to Bräysy and Gendreau [6], the three-approach method of Bräysy produces the best results in terms of solution quality for the Solomon test instances, along with the 'ruin and recreate' method of Schrimpf et al.

Metaheuristics

In the last part of this chapter the methods from the second category – metaheuristics – will be presented. Metaheuristics are general solution procedures that explore the solution space in order to identify good solutions and often embed some of the standard construction and improvement heuristics [7]. The main difference between the classical methods and metaheuristics is that metaheuristics allow deterioration of solutions during the search process in order to escape a local optima. Thus, better solutions can be obtained with metaheuristics compared to classical heuristics, though often at the cost of additional computation time. The most well-known metaheuristics

are simulated annealing, tabu search, GRASP, ant optimization and genetic algorithms.

There exist numerous tabu search implementations for the VRPTW. The initial solution in these implementations is usually constructed by some cheapest insertion heuristic. In a few implementations a savings method or a sweep heuristic is used as the route construction heuristics. After creating an initial solution, a local search procedure is performed in order to find a better solution. Neighbourhood structures used in local search are the ones also used in the context of route improvement techniques, such as 2-opt, Or-opt, relocate, CROSS-, GENI- and λ–exchanges.

Different strategies are used to reduce the complexity and hence to speed up the search: Garcia et al. [19] only allow moves involving arcs close in distance. Taillard et al. [1] decompose solutions into a number of subsets and apply tabu search to each set separately. A complete solution is then constructed by merging the new routes found by tabu search. The performance of the approach of Taillard et al. in terms of computation time has been improved by the parallel implementation of the algorithm. To overcome restrictions of the search space created by time window constraints, some authors allow infeasibilities during the search, i.e. accepting solutions where capacity or time windows constraints are violated [7].

A concept of adaptive memory is introduced as a tool of guiding the search by Rochat and Taillard [30]. The adaptive memory is a pool of routes taken from the best solutions visited during the search. New solutions are produced by selection and combination of routes from the adaptive memory. The selection of routes is a probabilistic procedure, and the probability of a route being chosen depends on the value of the objective function in the solution to which the route belongs. The algorithm is designed so as to allow the search to change progressively from a diversification to an intensification process. Diversification in the early iterations of the search allows to explore various regions of the solution space. Intensification in the last stages of the search aims at exploring the most promising regions. As an extra feature, a post-optimization procedure is applied after the search has finished: A set-partitioning problem is solved based on the routes from the adaptive memory. The method proposed by Rochart and Taillard performed well on the benchmark problem from the literature, e.g. they improved or reached quality of about 27 of 56 best solutions published for the Solomon problem instances.

Another diversification and intensification strategy is employed by Chiang and Russell [12]. They propose a reactive tabu search that dynamically

adjusts the length of the tabu list. It is increased if identical solutions occur too often and reduced if a feasible solution cannot be found.

Another widely used approach for the VRPTW is genetic algorithms. Thangiah et al. [37] were the first to apply a genetic algorithm to the VRPTW. Their approach is divided in two phases. In the first phase a genetic algorithm GENSECT is applied to form clusters of customers based on the sweep method. The routing of customers is then performed for each cluster separately using the cheapest insertion method. In this first module of the algorithm, the routes produced by GENSECT are allowed to contain infeasibilities with respect to capacity constraints and time windows. In the second phase, λ–exchanges and relocations are applied to remove the infeasibilities and improve the quality of solution. Since that time many authors have presented numerous implementations of genetic algorithms which differ by representation of solution space, selection rules and recombination and mutation strategies. According to [7] the authors who achieved the best results for the Solomon benchmark problems are Mester (2002), Berger et al. (2003) and Homberger and Gering (2005).

In addition to tabu search and genetic algorithms, the following metaheuristics have also been applied to VRPTW:

Kontoravdis and Bard [25] use a two-phase greedy randomized adaptive search procedure GRASP. In the first phase, the routes are initialized by choosing a set of seed customers. For each unrouted customer the best feasible insertion location is identified, and a penalty value is determined using Solomon's cost function. A customer to be inserted next is randomly selected from a list of customers with the largest penalty value. In the second phase, a local search is performed based on the route elimination method followed by the 2-opt procedure to improve the solution in terms of travel distance. The authors propose three different lower bounds for estimating the number of routes.

Tan et al. [36] develop a fast simulated annealing approach based on the two-interchanges with the best-accept strategy and a monotonously decreasing cooling scheme. Czech and Czarnas (2002) [13] describe a parallel version of a simulated annealing algorithm.

Gambardella et al. [18] use the multiple ant colony system to solve their VRPTW. The first ant colony minimizes the number of vehicles, while the second colony minimizes the travel distance. Cooperation between colonies is performed by updating the best solution found through pheromone updating. Both colonies are reactivated with the new parameters if a new best solution with fewer vehicles is found. According to the authors, the approach proves

to be competitive with the best known existing methods both in terms of solution quality and computation time.

Apart from the metaheuristics mentioned above, a number of hybrid approaches have been designed for the VRPTW, where different methods such as local search, simulated annealing, tabu serach and constraint programming are combined [7].

Conclusion and future development

It is difficult to compare the different approaches used to solve the VRPTW due to several reasons such as different programming languages and hardware used to implement the algorithms, differences in reporting the experimental results, incompleteness of the reported results etc. Thus, it is impossible to identify a single method as the best performing one both in terms of solution quality and computation time. However, usage of memory and employment of different route construction and improvement techniques can be mentioned as main characteristics of the most efficient solution approaches.

In their review paper Bräysy and Gendreau have identified 10 different metaheuristic approaches which performed best on Solomon's problem instances. Five of this approaches are based on genetic algorithms, one of them is an ant optimization algorithm and the others are combination of such methods as simulated annealing, LNS and tabu search. The differences in solution quality of these methods are quite small, i.e. within 0.5% and 1.2% in terms of the number of used vehicles and the travel distance. As the methods were run on different systems, it is difficult to compare their performance in terms of computation time. To make the comparison easier the reported computation times have been scaled to equal Sun Sparc 10, using factors of Dongarra [7]. For Solomon's problem instances with 100 customers the time required to produce the reported solutions varied from 106 minutes to 1458 minutes.

For the classical heuristic described in the first part of section 2.2 the best solutions were obtained by Schrimpf and Bräysy. The quality of solutions in terms of the number of used vehicles and the travel distance differed from the solutions obtained by the best metaheuristics by 0.7% and 5.2%. The computation time reported for the fastest of these two methods were 4.6 minutes on Pentium 400 MHz.

As pointed out by Bräysy and Gendreau one of the possible future trends in developing the solution methods may include tailored solution approaches based on careful analysis of the problem at hand. On the other hand, the

research on simpler and more flexible but effective metaheurtistics will also increase.

Chapter 3

The Vehicle Routing Problem with Time Windows and Shift Time Limits

In this chapter the model for the problem of constructing master plans is presented. As mentioned is section 1.2, the problem of constructing master plans can be characterized as a VRPTW with complicating constraints. Thus, the presented model is an extension of the model for the VRPTW described in section 2.1.2.

The model is extended to handle the vehicle availability and shift time limit constraints described in section 1.2.

In addition to the terminology in section 2.1.2, the following definitions are introduced: Let (A_k, B_k) and ST_k denote the availability time window and shift time limit for vehicle k. Vehicle k cannot be used for deliveries before A_k, and must be back at the depot not later than B_k. This implies the following restrictions on the variables denoting departure time from the depot and arrival time back at the depot for the vehicle:

$$st_k \geq A \tag{3.1}$$

$$f_k \leq B \tag{3.2}$$

Furthermore, the difference between the departure time and the arrival time of the vehicle must lie within the shift time limit, i.e.

$$st_k - f_k \leq ST_k \tag{3.3}$$

As mentioned before, the fleet of vehicles is not homogeneous. Typically, a number of different types of vehicles are available in the fleet. The types can

differ in capacity of the vehicles, availability time windows and the length of shift time. The number of vehicles of each type is fixed a-priori and it is not possible to get access to more vehicles. Thus, an additional constraint must be added to the model ensuring that only available vehicles of each type are used in the solution.

In the following the mathematical model for the problem of constructing master plans is presented. In the rest of the report the problem of constructing master plans will be denoted the *vehicle routing problem with time windows and shift time limits* (VRPTWSTL).

3.1 The Model Formulation

Before presenting the model an overview of the model components is given.

Sets

N indexed by $i = 1 \dots n$ or by $j = 1 \dots n$ is the set of all customers

V indexed by $i = 0 \dots n+1$ or by $j = 0 \dots n+1$ is the set of all customers and the depot, i.e. $V = N \cup \{0, n+1\}$ where 0 and $n+1$ denote the depot

E is the set of all edges of the underlying graph

M_l indexed by $k = 1 \dots m_l$ is the set of vehicles of type l

M indexed by $k = 1 \dots m$ is the set of all vehicles, $M = \cup_{l=0}^{L} M_l$ where L is the number of vehicles types

Parameters

q_i demand of customer i

E_i time window start at customer site i

L_i time window end at customer site i

s_i service time at customer i

Q_k capacity of vehicle k

A_k time from which vehicle k is available for use

B_k time until which vehicle k is available for use

ST_k shift time limt for vehicle k

L number of vehicle types

m_l number of vehicles of the same type

t_{ij} travel time between customers i and j

c_{ij} cost incurred by travelling directly from customer i to j

C a large number

Decision Variables

$$x_{ijk} = \begin{cases} 1 & \text{if vehicle } k \text{ travels directly from customer } i \text{ to customer } j \\ 0 & \text{otherwise} \end{cases}$$

t_i the time to begin delivery at customer i

st_k departure time of vehicle k from the depot

f_k arrival time of vehicle k at the depot

A model for the vehicle routing problem with time windows and shift time limits can then be formulated as follows:

$$\min \quad \sum_{(i,j)\in E}\sum_{k\in M} c_{ij}\cdot x_{ijk} \tag{3.4}$$

$$\text{s.t.} \quad \sum_{j\in V} x_{ijk} - \sum_{j\in V} x_{jik} = 0 \qquad \forall i \in N, \forall k \in M \tag{3.5}$$

$$\sum_{i\in N} x_{0ik} = 1 \qquad \forall k \in M \tag{3.6}$$

$$\sum_{i\in N} x_{i,n+1,k} = 1 \qquad \forall k \in M \tag{3.7}$$

$$\sum_{k\in K}\sum_{j\in V} x_{ijk} = 1 \qquad \forall i \in N \tag{3.8}$$

$$\sum_{i\in N} q_i \sum_{j\in V} x_{ijk} \leq Q_k \qquad \forall k \in M \tag{3.9}$$

$$t_i + s_i + t_{ij} - C(1 - x_{ijk}) \leq t_j \qquad \forall i,j \in N, \forall k \in M \tag{3.10}$$

$$st_k + t_{0i} - C(1 - x_{0ik}) \leq t_i \qquad \forall i \in N, \forall k \in M \tag{3.11}$$

$$t_i + s_i + t_{i,n+1} - C(1 - x_{i,n+1,k}) \leq f_k \qquad \forall i \in N, \forall k \in M \tag{3.12}$$

$$t_i \geq E_i \qquad \forall i \in N \tag{3.13}$$

$$t_i \leq L_i \qquad \forall i \in N \tag{3.14}$$

$$st_k \geq A_k \qquad \forall k \in M \tag{3.15}$$

$$f_k \leq B_k \qquad \forall k \in M \tag{3.16}$$

$$f_k - st_k \leq ST_k \qquad \forall k \in M \tag{3.17}$$

$$\sum_{k\in M_l}\sum_{i\in N} x_{0ik} \leq m_l \qquad l = 1 \ldots L \tag{3.18}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i,j \in V, \forall k \in M \tag{3.19}$$

3.2 Solution Approach

As concluded in section 2.2, it is difficult to determine a single solution method as the best performing one both in terms of solution quality and computation time.

When this thesis was initiated, the intention was to test the developed solution approach on problem faced by the company considered in this thesis, which has up to 3000 customers in Denmark as mentioned in section 1.1. The actual data set provided by Transvision A/S included 500 customers,

whereas the best known exact methods are able to solve the problems with up to 100 customers.

Due to the size of the problem and the fact that the solutions have to be produced within a reasonable time frame, use of exact solution methods is not considered.

Instead, the following 2-phase solution approach is employed in this thesis: In the first phase, a set of feasible routes is constructed using an insertion heuristic with embedded route improvement. The objective of the first phase is to produce a large number of distinctive routes of good quality. In the second phase the problem is formulated as a mixed set covering/packing problem, where the rows of the constraint matrix correspond to the customers to be covered, and the columns are the generated routes. The problem is then solved heuristically based on Lagrangian relaxation, subgradient optimization and a greedy heuristic for generating upper bounds.

In the next chapter the data for a case study problem, provided by Transvision A/S, is described. Chapters 5 and 6 describe the solution approach in more details. In chapter 5 an algorithm for the insertion heuristic used in the route construction phase is presented. Chapter 6 presents the mixed set covering/packing formulation of the problem and a Lagrangian-based heuristic used to solve the problem.

Chapter 4

A Case Study Data

In the following, a case study data for the VRPTWSTL presented in the previous chapter is described.

4.1 Data Set Provided by Transvision A/S

In chapter 1 it has been mentioned that the problem considered in this thesis is a vehicle routing problem faced by one of Transvision's client companies in connection with the daily distribution of products to the customers. When this project was started, the intention was to test the algorithm developed to solve the routing problem based on the real-life information about the changes in demand of the company's customers. The quality of the obtained solutions were then to be compared to the current practice.

As real life data from the company was not available, another data set has been provided by Transvision A/S to function as test data in the project.

The problem described by the obtained data involves 500 customers and a single depot. There are 60 vehicles available for servicing customers. These vehicles are of three different types. An overview of the differences between the vehicle types can be seen in table 4.1.

Vehicle type	**Capacity** Q_k	**Availability window**		**Shift time limit**, ST_k
		$\mathbf{A_k}$	$\mathbf{B_k}$	
311	25	02:00	18:00	600
312	35	03:00	16:00	600
313	35	00:00	14:00	600

Table 4.1: Problem data – Vehicle types. The shift time limit ST is expressed in minutes.

20 vehicles of each type are available. Furthermore, it is assumed that the average speed of the vehicles is 40 km/h.

Products of several types can be delivered to the customers. For each product type information about the unit volume of the product is available.

Product id	**Product type**	**Unit volume**
1534	'Rullepaller'	3
1535	'Helbure'	2
1536	'Halvbure'	0.5
1537	'Kasser'	0.1
1538	'Æsker'	0.0

Table 4.2: Problem data – Product types.

For each customer the following information is provided:

– time windows
– geographical position (x, y) expressed in meters
– customer demand, i.e. the product types and the ordered quantity

The service time at each customer is set to 10 minutes. In agreement with Transvision A/S, the Euclidean distances are used when calculating the distances and travel times between customers. For each pair of customers i and j the travel distance between them is determined as $d_{ij} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$, and the travel time is then computed as $t_{ij} = d_{ij}/speed$.

The geographical distribution of customers can be seen in figure 4.1. The depot coordinates are (530341,6147551), and the figure reveals several small clusters in the distribution of the customers.

It was mentioned in chapter 1 that the time windows for the customers usually are tight. This is not the case for the customers in the obtained data set. The customers' time windows are quite wide with an average time

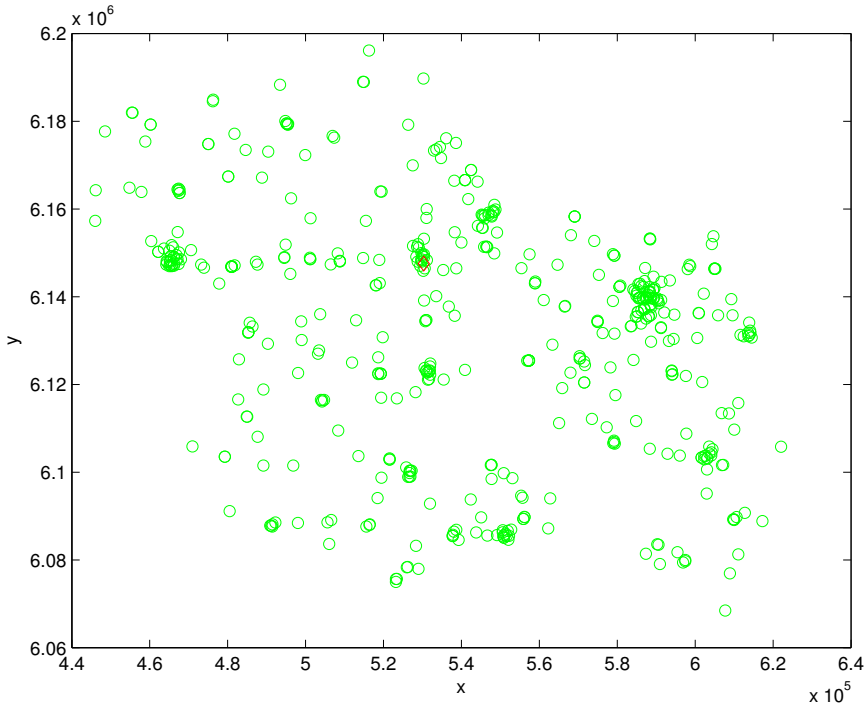


Figure 4.1: Geographical distribution of customers. ◇ denotes the depot.

window length of 384 minutes. The tightest time window has a length of 90 minutes and the widest has a length of 720 minutes.

Chapter 5

Route Generation Phase

In this chapter the route generation phase of the solution approach is described. An insertion heuristic used to construct a feasible set of routes is presented in section 5.1. Section 5.2 describes a post-insertion procedure applied in case where some customers are left uncovered after the insertion procedure. Finally, route improvement techniques applied to the routes both during the insertion procedure and after the initial set of routes is constructed are described in section 5.3.

5.1 Insertion Heuristic

One of the well-known methods for constructing an initial feasible solution in local search and metaheuristics for vehicle routing problems is to use insertion heuristic. The advantages of the insertion heuristics are that they are easy to implement, they are fast and produce decent solutions, and they can be extended to handle complicating constraints [9].

Different variants of the insertion methods are described in the literature. Generally, the existing insertion methods can be divided into broad categories of sequential and parallel methods. Sequential insertion heuristics construct one route at a time, whereas parallel methods construct multiple routes simultaneously. In parallel insertion heuristics the number of routes is either fixed in advance or can be determined by the construction procedure.

For the problem considered in this thesis, a parallel version of the insertion heuristic is implemented, as the number of vehicles and hence the number of routes is known a-priori.

A feasible solution, i.e. a set of feasible routes, is constructed by repeatedly and greedily inserting an unrouted customer into the best position in the partially constructed routes. A route is represented as $(0, 1, 2, \ldots, i, i+1, \ldots, n+1)$, where 0 and $n+1$ denote the depot and i refers to the customer at position i in the route. A pseudocode for the insertion procedure can be seen in algorithm 1 below.

Algorithm 1 Pseudo-code for the insertion procedure.

```
1: N – set of unassigned customers
2: R – set of routes
3: while N ≠ ∅ do
4:   for j=1 to |N| do
5:     c* = ∞
6:     for r=1 to |R| do
7:       for (i, i+1) ∈ r do
8:         if (Feasible(i, j) & Cost(i, j) < c*) then
9:           r* = r
10:          i* = i
11:          j* = j
12:          c* = Cost(i,j)
13:        end if
14:      end for
15:    end for
16:  end for
17:  Insert(r*, i*, j*)
18:  Update(r*)
19:  N = N \ j*
20: end while
```

Initially, N is the set of all customers, and R is the set of empty routes, one for each vehicle. In each major iteration of the algorithm, a customer is selected and inserted into a partial route, which is subsequently updated. The number of major iterations equals the number of customers to be routed, n.

One way of selecting customers for insertion is evaluating all the unrouted customers at every possible insertion point and then choosing the cheapest insertion. As there are $O(n)$ unrouted customers and $O(n)$ possible insertion places, this would lead to the following time complexity of a major iteration:

$$O(n^2) \cdot T(\mathit{Feasible}() + \mathit{Cost}()) + T(\mathit{Update}())$$

Another strategy is to select customers for insertion in a particular order. The following ordering rules can be applied to selecting customers for insertion, e.g.:

- choosing a customer farthest from the depot
- choosing a customer with the earliest allowed starting time

As the number of vehicles is fixed in advance, and both time windows and shift time limit constraints apply to the vehicles, some customers may be difficult or impossible to place in the routes in late iterations of the insertion algorithm. Thus, the motivation for seeding the customers is to ensure that the decision of placing these *difficult* requests into the routes is not postponed until late iterations of the insertion process. If the customers are sorted in advance, the overall complexity of a major iteration can be reduced to $O(n) \cdot T(Feasible() + Cost()) + T(Update())$.

In the above formulas the time used to check the feasibility and compute the cost of an insertion has a crucial impact on the overall time complexity of the algorithm. Straightforward implementations of the feasibility check perform a physical insertion of customer j between two customers i and $i+1$. The route is then traversed to check feasibility, which takes $O(n)$ time and leads to the time complexity of $O(n^4)$(or $O(n^3)$ if seeding is applied) for the overall insertion procedure.

More efficient implementations of checking the feasibility have been suggested in the literature. In this thesis the approach described by Campbell and Savelsbergh [9] is applied when handling vehicle capacity and customer time window constraints. Some of their ideas are then used when implementing feasibility checks with respect to availability windows and shift time limits for vehicles.

5.1.1 Feasibility check

For the VRPTWSTL problem considered in this thesis, there are four types of constraints that have to be checked to verify the feasibility of the insertion. Vehicle capacity constraints and time window constraints for the customers are the typical constraints for the VRPTW. Furthermore, the problem is complicated by the availability windows and shift time constraints for the vehicles. In the work of Campbell and Savelsbergh [9] , efficient implementations of the feasibility checks are suggested for the VRPTW with complicating constraints, shift time limits being one of them. The authors state that

by maintaining the appropriate information about the partially constructed route all feasibility checks can be performed in constant time.

In the following, the implementations of the feasibility checks for different constraint types are discussed. It will be shown that the approach proposed by Campbell and Savelsbergh holds for only three out of the four mentioned constraints.

Vehicle capacity constraints

Verifying this type of constraints can be done in constant time if the information about the sum of demands of customers already assigned to the route is maintained. Let D_r denote the sum of delivery quantities currently assigned to the route, and let q_j denote the demand of the customer to be inserted into the route. The insertion is feasible if $q_j < Q_k - D_r$, where Q_k is the capacity of the vehicle assigned to the route.

Time windows constraints for the customers

For each customer i already in the route the following information is maintained:

e_i earliest time a delivery can begin at i
l_i latest time a delivery can begin at i

These values are used to capture the interactions between customers on the same route. Initially $e_i = E_i$ and $l_i = L_i$ for all $i \in N$, where (E_i, L_i) is a time window of customer i.

For a new customer j to be inserted between customers i and $i + 1$, the earliest and the latest time a delivery can begin at can be computed as follows: $e_j = max(E_j, e_i + s_i + t_{ij})$ and $l_j = min(L_j, l_{i+1} - t_{j,i+1} - s_j)$, see figure 5.1.

Given these quantities, the feasibility of the insertion with respect to time window constraints can be verified by checking whether $e_j \leq l_j$. This can be done in constant time. In figure 5.1 insertion of customer j is feasible with respect to the time window constraint.

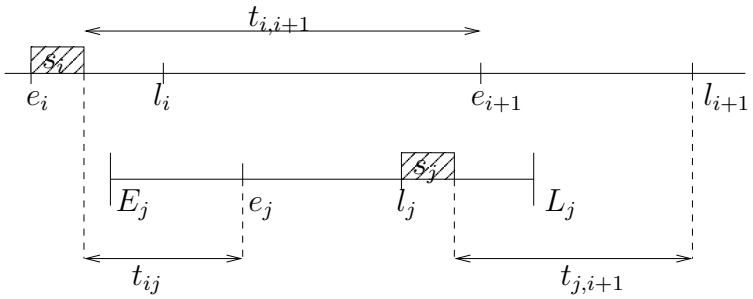


Figure 5.1: Example of feasible insertion wrt. time windows.

In figure 5.2 an example of an infeasible insertion is shown. The insertion is infeasible as the travel time from customer i to j is too large compared to the width of the time window for customer j. Thus, if customer j is to be serviced after customer i, the vehicle will arrive at customer j after its time window has closed. The feasibility check will fail as $e_j > l_j$.

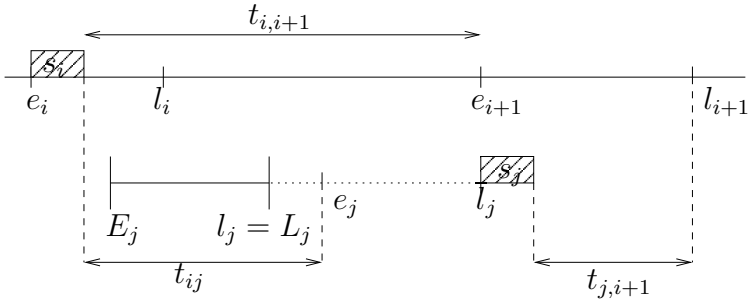


Figure 5.2: Example of infeasible insertion wrt. time windows. $\overline{l}_j = l_{i+1} - t_{j,i+1} - s_j$

Availability windows and shift time limit constraints for the vehicles

The problem considered in this thesis is further complicated by the availability windows and shift time limit constraints for the vehicles. Each vehicle is available for use during some prespecified hours of the day – the *vehicle availability window*. Furthermore, there are limits on how many hours the drivers can work – *shift time limits*. This type of restriction can be placed due to safety reasons or scheduling reasons.

Each time a new customer is inserted in a route one has to ensure that the route can still be completed within the availability window of the vehicle

assigned to the route. Furthermore, the shift time limit must be respected. Let the *route duration* be defined as the difference between the time the vehicle leaves the depot and the time it arrives back at the depot. For an insertion to be feasible, the route duration must not exceed the prespecified shift time limit.

In the following, the approach suggested by Campbell and Savelsbergh for verifying feasibility with respect to availability windows and shift time limits will be introduced. It will also be shown, that in some case the checks proposed by the authors are not sufficient, and the approach will fail to produce feasible routes.

Campbell and Savelsbergh approach

The authors state that to be able to perform the feasibility check in constant time the following information must be maintained for each route:

e_0	earliest possible departure time from the depot
l_0	latest possible departure time from the depot
e_{n+1}	earliest possible arrival time back at the depot or earliest completion time of the route
l_{n+1}	latest possible arrival time back at the depot or latest completion time of the route

Furthermore, for each customer a *cumulative time to the depot* denoted a_i is maintained. The cumulative distance to the depot for customer i is defined as follows: $a_i = s_i + t_{i,i+1} + a_{i+1}$. From this definition it is clear that the cumulative time only includes travel time between customers and their service times. It is important to note that it does not account for any possible waiting time on the route.

Initially, the earliest/latest departure time and the earliest/latest arrival time for each route are set to the availability windows of vehicle k assigned to the route, i.e., $(e_0, l_0) = (A_k, F_k)$ and $(e_{n+1}, l_{n+1}) = (A_k, F_k)$. The value of a_0 is set to 0.

Each time a new customer j is inserted into the route, the values of the earliest and latest times for delivery at customer j are determined. Based on these values, the earliest completion time of the route (earliest arrival time at the depot) is determined as $e_{n+1} = max(e_{n+1}, e_j + s_j + t_{j,i+1} + a_{i+1})$. The earliest completion time may be unchanged, i.e. the first term of the maximum is dominating if there is a significant amount of waiting time on the part of the route from customer $i + 1$ to the depot.

In figure 5.3 the earliest delivery time of the customer in position 2 is determined by E_2, opening of customer's time window, and not by the travel

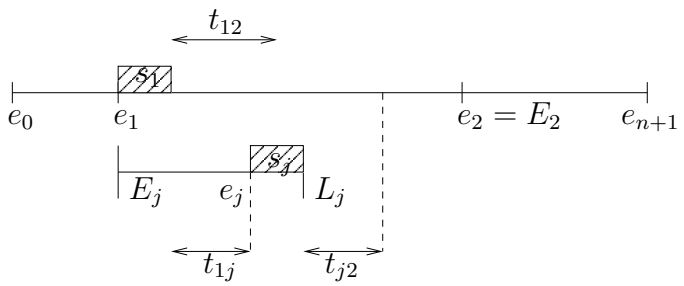


Figure 5.3: Example of insertion where the earliest completion time e_{n+1} of the route is unchanged.

time from the previous customer, i.e. $e_2 = E_2$ as $e_1 + s_1 + t_{12} \ll E_2$. The earliest completion time of the route is then computed as $E_2 + s_2 + t_{2,n+1}$. After customer j is inserted $e_j + s_j + t_{j2}$ is still less than e_2. Thus, this value will not change, and e_{n+1} will also remain the same.

However, if the earliest completion time is changed, the earliest departure time might also change. The implied earliest departure time is computed as $e_0 = max(e_0, e_{n+1} - ST)$. If e_0 is determined by the second term, then insertion of customer j will reduce the amount of time available on the part of the route before customer j.

The latest departure time from the depot is computed as $l_0 = min(l_0, l_j - t_{ji} - s_i - (a_0 - a_i))$, and the implied latest completion time of the route is $l_{n+1} = min(l_{n+1}, l_0 + ST)$. If the latest departure time l_0 is defined by the second term, there is no waiting time on the part of the route from the depot up to customer i. In that case, the latest completion time of the route l_{n+1} may also change. Similarly to the earliest departure time, if l_{n+1} is changed, the added time used to visit customer j will reduce the amount of time available on the part of the route after customer j.

Campbell and Savelsbergh state that the insertion is feasible with respect to vehicle availability windows and shift time limits if the following conditions hold: $e_0 \leq l_0$ and $e_{n+1} \leq l_{n+1}$. Clearly, this check can be performed in constant time.

It can be shown, however, that these conditions are not sufficient to ensure the feasibility of the insertion with respect to shift time limit constraints. The situation is illustrated by a small example in figure 5.4.

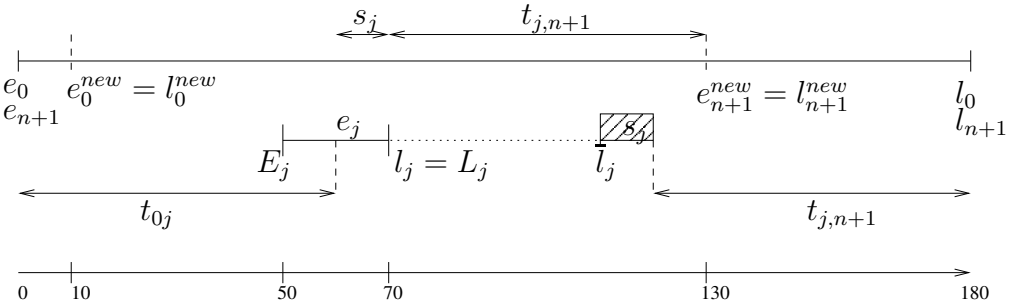


Figure 5.4: Example of infeasible insertion wrt. shift time limit. Shift time limit $ST = 120$, availability window of the vehicle assigned to the route $(A_k, B_k) = (0, 180)$, $\bar{l}_j = l_{n+1} - t_{j,n+1} - s_j$

As the route is empty, the earliest/latest departure times and the earliest/latest arrival times back at the depot are set to the availability window of the vehicle assigned to the route: $(e_0, l_0) = (e_{n+1}, l_{n+1}) = (0, 180)$ and $a_0 = 0$. A customer with time window $(E_j, L_j) = (50, 70)$ and service time $s_j = 10$ is to be inserted into an empty route. The distance between the depot and the customer is 60, and the shift time limit is set to 120.

Firstly, the earliest and latest delivery times for customer j and the cumulative time from customer j to the depot are calculated:

$$
\begin{aligned}
e_j &= max(E_j, e_0 + s_0 + t_{0j}) = max(50, 0 + 60) = 60 \\
l_j &= min(L_j, l_{n+1} - s_j - t_{jn+1}) = min(70, 180 - 10 - 60) = 70 \\
a_j &= s_j + t_{0j} = 10 + 60 = 70
\end{aligned}
$$

Based on e_j and l_j, the earliest completion time of the route and the latest departure time from the depot become:

$$
\begin{aligned}
e_{n+1}^{new} &= max(e_{n+1}, e_j + s_j + t_{j,n+1}) = max(0, 130) = 130 \\
l_0^{new} &= min(l_0, l_j - s_0 - t_{0j}) = min(180, 70 - 60) = 10
\end{aligned}
$$

leading to changes in the earliest departure time e_0 from the depot and the latest completion time l_{n+1} of the route:

$$
\begin{aligned}
e_0^{new} &= max(e_0, e_{n+1}^{new} - ST) = max(0, 130 - 120) = 10 \\
l_{n+1}^{new} &= min(l_{n+1}, l_0^{new} + ST) = min(180, 130) = 130
\end{aligned}
$$

After all these relevant values have been computed, the insertion seems to be feasible according to Campbell and Savelsbergh, as $e_j \leq l_j$, $e_0^{new} \leq l_0^{new}$ and $e_{n+1}^{new} \leq l_{n+1}^{new}$.

However, insertion of customer j is in fact *infeasible* with respect to the shift time limit: If customer j is inserted, the cumulative time a_0 from the depot and back to the depot will become $a_j + t_{0j} = 70 + 60 = 130$ leading to the violation of the shift time limit.

Cambell and Savelsbergh approach – revised

To ensure the feasibility with respect to shift time limit constraints, an additional check must be added to the conditions already mentioned by Campbell and Savelsbergh.

Based on the above example, an obvious suggestion is to check that the value of a_0 does not exceed the prespecified shift time limit, i.e.

$$a_0 \leq ST \tag{5.1}$$

However, it is important to keep in mind the difference between the route duration and the a_0-value maintained for the route during the insertion process. In the current implementation a_0 is defined as the sum of travel times and service times on the route, whereas the route duration also takes possible waiting times into account. Thus, the values of a_0 and the route duration will only be the same if the sum of waiting time on the route is zero as in the above example. In that case, condition (5.1) is sufficient to ensure the feasibility with respect to shift time limit. Each time a new customer j is inserted between the customers i and $i+1$, the new value of a_0 can be computed as follows: $a_0 = a_0 + t_{i,j} + t_{j,i+1} - t_{i,i+1} + s_j$. Clearly, this can be done in constant time.

The next step is to consider a situation, where insertion of customer j causes waiting time on the route. In that case, the updated value of a_0 will not be the same as the route duration, due to the waiting time. In the suggested approach the actual delivery times for the customers and possible waiting times at the customers are calculated after the insertion procedure is completed. Thus, the final value of the route duration is not known during the insertion process.

It is possible to maintain the information about the delivery times and waiting times on the route during the insertion process. However, checking how the insertion of customer j will affect the delivery and waiting times, and hence the route duration, will require a physical insertion of customer j into the route and then traversing the route to update the needed values. After this is performed, the route duration can be determined, and feasibility with

respect to shift time limit can be verified. As there are $O(n)$ customers in the route, this check *cannot* be performed in constant time.

In the following, it will be shown that route duration exceeding shift time limit due to waiting will not be an issue for the problem considered in this thesis. If the sum of waiting times on the route and the value of a_0 should exceed the shift time limit, then the feasibility check would fail on condition $e_{n+1} \leq l_{n+1}$.

Consider the partially constructed route $0, 1, 2, \ldots, i-1, i, n, n+1$ in figure 5.5 and a customer j which is to be inserted between customer i and the depot $n+1$. It is assumed that there is no waiting time on the route and $a_0 < ST$. Furthermore, it is assumed that insertion of customer j will cause waiting time on the route, i.e. $E_j \gg l_i + s_i + t_{ij}$. Let w_j denote waiting time at customer j. Then the value of w_j can be computed as follows:

$$w_j = E_j - (l_i + s_i + t_{ij}) \tag{5.2}$$

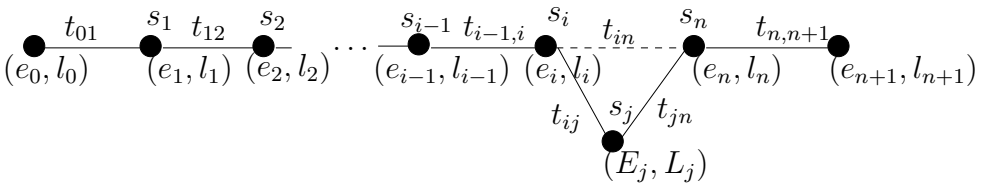


Figure 5.5: Partially constructed route where insertion of customer j will cause waiting time at the customer.

Statement 5.1. *If $a_0 + \sum_{i=0}^{j} w_i = a_0 + w_j > ST$ then $e_{n+1} > l_{n+1}$, i.e., the insertion is infeasible with respect to availability windows.*

Before the proof of statement 5.1 it is important to note the following:

- After the first customer is inserted into the route, l_0 is changed from its initial value of B_k to $L_1 - t_{01}$. If $L_1 - t_{01} + ST < B_k$, then l_{n+1} is changed to $l_0 + ST$. Otherwise, l_{n+1} remains fixed to the upper limit of availability window B_k.
- If the value of l_0 is changed (decreased) as more customers are inserted into the route, l_{n+1} can potentially be decreased to $l_0 + ST$. Thus, the following condition for these two values is valid throughout the whole insertion procedure: $l_{n+1} \leq l_0 + ST$.

- If there are no gaps between the l-values on the part of the route from customer i and back to the depot, then l_0 can be computed as $l_i - t_{i-1,i} - s_{i-1} - (a_0 - a_{i-1})$. No gaps means that l_i for each customer i of the route is determined by travel time to the next customer, and not by the closing of the customer's time window L_i.

- If there are gaps in the l-values, say on the part of the route from customer 2 up to customer i, then l_0 is determined by the l_2 value, i.e. $l_0 = l_2 - t_{12} - s_1 - (a_0 - a_1) = l_2 - t_{12} - s_1 - d_{01}$. Let $\overline{l}_0$ denote the following value $l_i - t_{i-1,i} - s_i - (a_0 - a_{i-1})$. Then the following condition holds:
$$\overline{l}_0 > l_0 \tag{5.3}$$

- If the insertion of customer j is feasible and causes a waiting time on the route, then the earliest completion time of the route is determined by E_j:
$$e_{n+1} = E_j + s_j + t_{jn} + a_n \tag{5.4}$$

Proof of statement 5.1. Assume that insertion of customer j is feasible. Then one of the conditions which have to hold is

$$e_{n+1} \leq l_{n+1} \tag{5.5}$$

Assume now, that the duration of the route exceeds the shift time limit if customer j is inserted, i.e. $a_0 + w_j > ST$. Using the definition of waiting time in equation (5.2), the following can be deduced:

$$\begin{aligned} a_0 + w_j > ST \quad &\Leftrightarrow \quad (5.6) \\ a_0 + E_j - (l_i + s_i + t_{ij}) > ST \quad &\Leftrightarrow \quad (5.7) \\ E_j > ST + l_i + s_i + t_{ij} - a_0 \quad &(5.8) \end{aligned}$$

If the E_j-value in equation (5.4) is replaced by the right hand side of equation (5.8), the following is obtained:

$$e_{n+1} > ST + l_i + s_i + t_{ij} - a_0 + s_j + t_{jn} + a_n \tag{5.9}$$

Remember, that a_0 is defined as the sum of travel times and service times on the route. The value of $a_0 - a_{i-1}$ will then correspond to the sum of the

travel times and services times on the part of the route between the depot and customer $i-1$. Thus, the value of a_0 can also be represented as:

$$a_0 = a_0 - a_{i-1} + s_{i-1} + t_{i-1i} + s_i + t_{ij} + s_j + t_{jn} + s_n + t_{nn+1} \tag{5.10}$$

Substitution of a_0 in (5.9) with the right hand side of equation (5.10) yields the following result:

$$\begin{aligned} e_{n+1} > ST + l_i + s_i + t_{ij} - (a_0 - a_{i-1}) - s_{i-1} - t_{i-1i} \\ -s_i - t_{ij} - s_j - t_{jn} - s_n - t_{nn+1} + s_j + t_{jn} + a_n \quad \Leftrightarrow \end{aligned} \tag{5.11}$$

$$e_{n+1} > \underbrace{l_i - t_{i-1i} - s_{i-1} - (a_0 - a_{i-1})}_{\overline{l}_0} + ST \tag{5.12}$$

If there are no gaps in the l-values on the route, then $\overline{l}_0 = l_0$, otherwise $\overline{l}_0 > l_0$. In both cases the following will hold:

$$e_{n+1} > l_0 + ST \geq l_{n+1} \quad \Rightarrow \tag{5.13}$$

$$e_{n+1} > l_{n+1} \tag{5.14}$$

However, the last equation contradicts the assumption that the insertion of customer j was feasible, i.e. equation (5.5). □

The conclusion is that if the sum of waiting times on the route and the value of a_0 should exceed the shift time limit, then the feasibility check will fail on condition $e_{n+1} \leq l_{n+1}$.

Feasibility checks - summary

Based on the above discussion, verifying feasibility of the insertion for the problem considered in this thesis amounts to checking the following conditions:

$$q_j < Q - D_r \tag{5.15}$$

$$e_j \leq l_j \tag{5.16}$$

$$e_0 \leq l_0 \tag{5.17}$$

$$e_{n+1} \leq l_{n+1} \tag{5.18}$$

$$a_0 \leq ST \tag{5.19}$$

All five conditions can be checked in constant time. The insertion is infeasible if any of these conditions are violated.

5.1.2 Insertion cost

The cheapest insertion criterion is applied in the algorithm to determine the insertion place of an unrouted customer j. The insertion cost for each possible insertion place is determined by the extra travel time caused by inserting customer j between customers i and $i+1$, i.e. $t_{i,j} + t_{j,i+1} - t_{i,i+1}$. Customer j is then inserted into the position in a route with lowest insertion cost, thus minimizing the total travel distance:

$$i^* = \underset{i}{\operatorname{argmin}}\ t_{i,j} + t_{j,i+1} - t_{i,i+1} \tag{5.20}$$

It is obvious that computing the insertion cost can be done in constant time.

5.1.3 Updating the route

An inserted customer can impact the deliveries to customers that appear both before it and after it on the route. For the customers earlier in the route, there might not be as much time to postpone their deliveries, whereas deliveries to the customers later in the route might have to start earlier.

Pseudocode for the *Update()*-procedure is shown in algorithm 2.

After the insertion of customer j in the route, the l-values of the customers prior in the route and the e-values of customers later in the route have to be updated, see lines 4–7 and 9–12 of the pseudocode. In practice, the latest delivery time for the customers preceeding j have to be updated only as long as the value for the recently updated customer has changed. For example, if l_{i-1} does not change, there is no need to recalculate l_{i-2}. The same holds for the earliest delivery time for the customers following j.

Algorithm 2 Pseudo-code for the *Update()*-procedure.

1: {*Update the total load on the route*}
2: $D_r = D_r + q_j$
3:
4: {*Update the latest delivery time for the customers prior in the route*}
5: **for** $c = i \ldots 0$ **do**
6: $\quad l_c = min(l_c, l_{c+1} - t_{c,c+1} - s_c)$
7: **end for**
8:
9: {*Update the earliest delivery time for the customers later in the route*}
10: **for** $c = i + 1 \ldots n + 1$ **do**
11: $\quad e_c = max(e_c, e_{c-1} + s_{c-1} + t_{c-1,c})$
12: **end for**
13:
14: {*Compute the implied earliest departure time and latest completion time*}
15: $\overline{e}_0 = max(e_0, e_{n+1} - ST)$
16: $\overline{l}_{n+1} = min(l_{n+1}, l_0 + ST)$
17:
18: {*If the implied earliest departure time has changed, update the e-values for the customers just before the point of insertion*}
19: **if** $\overline{e}_0 > e_0$ **then**
20: $\quad e_0 = \overline{e}_0$
21: $\quad$ **for** $c = 1 \ldots i$ **do**
22: $\quad\quad e_c = max(e_c, e_{c-1} + s_{c-1} + t_{c-1,c})$
23: $\quad$ **end for**
24: **end if**
25:
26: {*If the implied latest completion time has changed, update the l-values for the customers just after the point of insertion*}
27: **if** $\overline{l}_{n+1} < l_{n+1}$ **then**
28: $\quad l_{n+1} = \overline{l}_{n+1}$
29: $\quad$ **for** $c = n \ldots i + 1$ **do**
30: $\quad\quad l_c = min(l_c, l_{c+1} - t_{c,c+1} - s_c)$
31: $\quad$ **end for**
32: **end if**
33:
34: {*Update the cumulative distance to the depot for the customers later in the route*}
35: **for** $c = i \ldots 0$ **do**
36: $\quad a_c = a_c + (t_{i,j} + t_{j,i+1} - t_{i,i+1}) + s_j$
37: **end for**

Furthermore, if the latest completion time of route l_{n+1} or the earliest departure time e_0 has changed due to the insertion of a new customer j, the additional updates have to be performed.

If e_0 has increased, then after updating the e-values for the customers after the insertion point, the values from e_0 forward to e_i have to be updated. Similarly, if l_{n+1} has changed, then after updating the l-values for the customers prior to the point of insertion, the values from l_{n+1} backward to l_{i+1} must be updated.

There is no further explanation of the updating procedure in the article of Campbell and Savelsbergh, which makes it hard for the reader to see what happens if a certain situation occurs. Consider the partially constructed route in figure 5.6 with route duration less than the shift time ST.

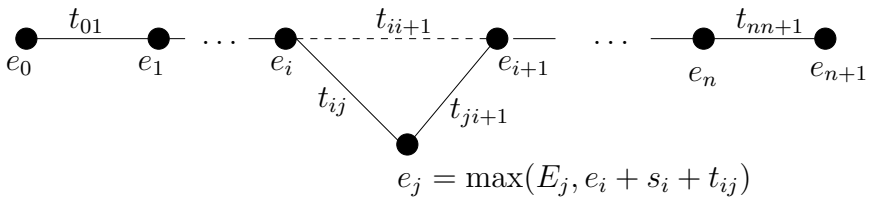


Figure 5.6: Partially constructed route with route duration less than shift time. Only the earliest possible delivery times are displayed.

Customer j is to be inserted into the route between customers i and $i+1$. Assume that e_0 has changed due to the insertion, and that e-value for the customer in position i is changed in the updating process. Remember that e_j was computed as $max(E_j, e_i + s_i + t_{ij})$. Due to the change in e_i, it seems necessary to re-evaluate e_j. If e_j is determined by the second term of the maximum, then the e-values for the customers following customer j will also have to change, including the value of the earliest completion time of the route, e_{n+1}. The last change will cause e_0 to change again due to the shift time limit, and the whole updating procedure will have to be repeated. The question is now how many times the updating procedure have to be repeated and whether an $O(n)$ complexity of the whole updating phase can be preserved. The same doubtful situation occurs if the latest completion time of the route, l_{n+1}, is changed due to the shift time limit, and the l-value for the customer in position $i+1$ is changed during the updating procedure.

In the following, it will be shown that there will be no need to update the values of e_j, or l_j, and thus going through the updating procedure several times. It can be shown that if such updates are necessary, the insertion would not have been feasible in the first place.

To simplify further explanations it is assumed, that service time for each customer is included in travel time from this customer to the next.

Statement 5.2. *Assume that insertion of customer j causes delay of the earliest completion time and thus change of e_0. Assume that e_i is changed in the updating process. The value of e_j will remain unchanged, as otherwise the insertion would have been infeasible with respect to shift time limit.*

Proof of the statement 5.2. Assume that insertion of customer j between customers i and $i+1$ is feasible. Then one of the conditions that have to hold is

$$a_0 \leq ST \tag{5.21}$$

After insertion the new value of e_{n+1} can be computed based on the value of e_j:

$$e_{n+1} = e_j + t_{ji+1} + a_{i+1} \tag{5.22}$$

and e_0 is re-calculated to:

$$e_0 = e_{n+1} - ST \tag{5.23}$$

Due to the updating procedure, the e-value of customer i is changed to

$$e_i^{new} = e_0 + (a_0 - a_i) \tag{5.24}$$

The only way for e_j to change is if condition (5.25) holds.

$$\begin{aligned} & e_i^{new} + t_{ij} > e_j && \Leftrightarrow && (5.25)\\ & e_0 + (a_0 - a_i) + t_{ij} > e_j && \Leftrightarrow && (5.26)\\ & e_0 + (a_0 - a_i) + t_{ij} > e_{n+1} - t_{ji+1} - a_{i+1} && && (5.27) \end{aligned}$$

By substitution of e_0 in (5.27) with the right hand side of equation (5.23) following is deduced:

$$\begin{aligned} & e_{n+1} - ST + a_0 - a_i + t_{ij} > e_{n+1} - t_{ji+1} - a_{i+1} && \Leftrightarrow && (5.28)\\ & a_0 + t_{ij} + t_{ji+1} + a_{i+1} - a_i > ST && \Leftrightarrow && (5.29)\\ & a_0 + t_{ij} + t_{ji+1} - t_{ij} - t_{ji+1} > ST && \Leftrightarrow && (5.30)\\ & a_0 > ST && && (5.31) \end{aligned}$$

The last equation contradicts the assumption that the insertion of customer j was feasible – see equation (5.21).

□

A similar statement can be proved for updating the l-values:

Statement 5.3. *Assume that insertion of customer j causes the latest departure time to be decreased, and that the latest completion time is also changed due to the shift time limit, $l_{n+1} = l_0 + ST$. Assume that l_{i+1} is changed in updating process. The value of l_j will remain unchanged, as otherwise the insertion would have been infeasible with respect to the shift time limit.*

Proof. Similar to the one for statement 5.2 and not presented here, but can be found in appendix A.

□

Apart from the updates mentioned in the above, the total delivery volume of the route and the cumulative travel distance to the depot of the customers preceeding j have to be updated – lines 1-2 and 34-37 of the pseudocode described in algorithm 2. Clearly, all these updates can be done in linear time thus preserving the overall time complexity of $O(n^3)$ – $O(n^2)$ if seeding is applied – for the proposed insertion algorithm.

5.1.4 Finalizing the solution

After a set of feasible routes is generated, the final solution, i.e. the actual delivery times for each customer, can easily be constructed based on the information maintained for each customer. One of the approaches is to make deliveries either at the earliest possible. Another approach is to begin deliveries at the latest possible time for all the customers. Both choices will give a feasible solution and require the same amount of travel time. Consider the route in figure 5.7. For each customer i of the route the final values of (e_i, l_i) are determined. Furthermore, the earliest/latest departure times from the depot and the earliest/latest completion times of the route are calculated.

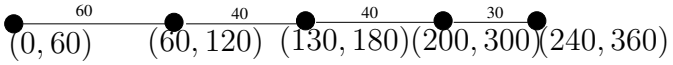


Figure 5.7: Final route. Service times for all customers are set to 10.

In figure 5.8 the final solution for the route is constructed by starting from the depot at the earliest possible time, e_0.

The drawback of this approach is possible long waiting times on the routes, e.g. if a vehicle arrives at the customer site before the time window opens.

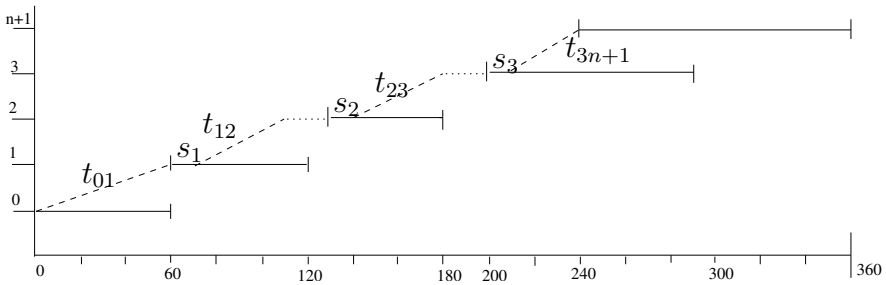


Figure 5.8: Final solution with start at the earliest departure time. Solid lines indicate the final interval (e,l) for the depot and the customers. Dashed lines indicate travel time between customers and the depot. Dotted lines indicate waiting time at customer sites.

According to the solution in figure 5.8, there is waiting time of 10 and 20 minutes at customers 2 and 3, respectively. The total route duration is calculated to be $240 - 0 = 240$ minutes.

To minimize waiting time for a given route, it is best to depart from the depot at the latest possible time (l_0). Deliveries to the following customers must then begin at the earliest feasible time. This approach minimizes the duration of the route, thus minimizing the waiting time. The solution constructed using this approach can be seen in figure 5.9.

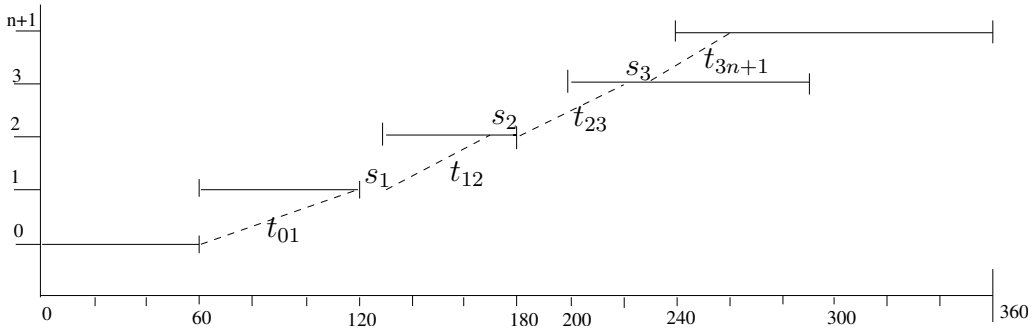


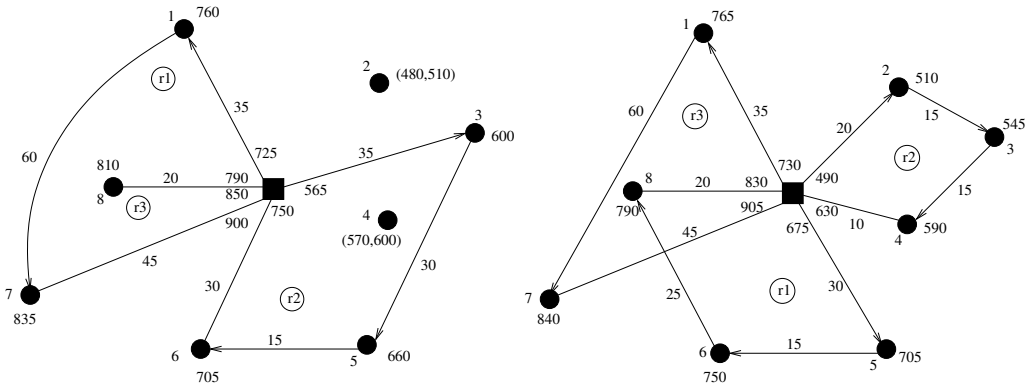
Figure 5.9: Final solution with start at the latest departure time. Solid lines indicate the final interval (e,l) for the depot and the customers. Dashed lines indicate travel time between customers and the depot. Dotted lines indicate waiting time at customer sites.

As it can be seen from the figure, there is no waiting time on the route, and the route duration is $260 - 60 = 200$ minutes.

5.2 Post-insertion Procedure

As the number of vehicles is fixed a-priori, there might still be unrouted customers after the insertion algorithm has finished. In the worst case some of these customers cannot be served by the available vehicles due to the violation of some constraints, e.g. not enough capacity on the vehicles or time window incompatibilities. In that case it is not possible to construct a feasible solution with all customers served.

In most cases, however, some customers will remain unserved due to the particular order in which customers are chosen for insertion. In figure 5.10 two different solutions to the same problem are shown. In figure 5.10.a the next customer chosen for insertion is the one farthest from the depot. In figure 5.10.b the customer with the earliest allowed starting time is chosen in each iteration. The problem data and the detailed insertion procedure for the solution in figure 5.10.a can be found in appendix B. As it can be



(a) Total cost: 420 minutes, two customers unserved.

(b) Total cost: 470 minutes, all customers served.

Figure 5.10: Dependency of final solution on seeding criteria. Service times for the customers are not displayed in the figure.

seen from the figure two customers are left unserved if the *'farthest distance from the depot'*-criterion is applied. Thus, a procedure for handling these customers is required.

In this thesis an ejection chain based approach is applied to construct the final feasible solution. The idea of this approach is trying to insert an unserved customer in some route by removing another customer from the same route [6].

Algorithm 3 Pseudocode for the post-insertion procedure. i^*_{ins} denotes the position in route r^* where the unserved customer j is to be inserted. i^*_{rem} denotes the position of customer j_{out} which is removed from the route.

N_{uns} – set of unserved customers
N'_{uns} – set of still unserved customers after the procedure is finished
R – set of routes
repeatProc – is *true* if the procedure has to be repeated, i.e. if $|N'_{uns}| \neq 0$
for j=1 to $|N_{uns}|$ **do**
  j_{in} = j
  while $j_{in} \neq -1$ **do**
    if ***ExistFeasible***(R, j_{in}) **then**
      {direct insertion possible, chain terminated}
      Insert(r^*, i^*, j_{in})
      Update(r^*)
      $j_{in} = -1$
    else
      {direct insertion not possible, start the chain}
      $c^* = \infty$, $r^* = -1$, $i^*_{ins} = -1$ and $i^*_{rem} = -1$
      $(c^*, r^*, i^*_{ins}, i^*_{rem})$ = ***FindRemovalPlace***(R, j_{in})
      if $(c^* \neq \infty$ & $i^*_{ins} \neq -1)$ **then**
        j_{out} = ***Remove***(r^*, i^*_{rem})
        Insert(r^*, i^*_{ins}, j_{in})
        Update(r^*)
        $j_{in} = j_{out}$
      else
        $N'_{uns} = N'_{uns} \cup \{j_{in}\}$
        $j_{in} = -1$
      end if
    end if
  end while
end for
if $|N'_{uns}| > 0$ **then**
  if $(|N_{uns}| = |N'_{uns}|$ & $N_{uns} \setminus N'_{uns} = \emptyset)$ **then**
    repeatProc = false
  else
    repeatProc = true
  end if
else
  repeatProc = false
end if
return *repeatProc*

A pseudocode for the post-insertion procedure is presented in algorithm 3. In each iteration of the post-insertion procedure an unserved customer j is chosen from the set N_{uns}. The algorithm then tries to insert the customer into existing routes by applying the insertion algorithm described in the previous section – the *ExistFeasible()* procedure. If direct insertion is not possible, the ejection chain is started. Procedure *FindRemovalPlace()* finds a route where customer j can be inserted by removing another customer j_{out}. Afterwards the same procedure is applied to the removed customer. The insertion and removal procedures are repeated until it is possible to insert a customer into a route without removing another customer from that route. The ejection chain (i.e. the *while*-loop in the pseudocode) is therefore terminated and the next unserved customer is chosen from N_{uns}.

A pseudocode for the *FindRemovalPlace()* procedure is shown in algorithm 4.

Algorithm 4 Pseudocode for the *FindRemovalPlace()*-procedure.

```
R – set of routes
j – customer to be inserted into the routes
c* = ∞, r* = −1, i*_ins = −1 and i*_rem = −1
for r=1 to |R| do
   for l ∈ r  do
      r' = Remove(r,l)
      savings^r_l = t_{l−1,l} + t_{l,l+1} − t_{l−1,l+1}
      for (i, i + 1) ∈ r' do
         if (Feasible(i, j) & Cost(i, j) – savings^r_l < c*) then
            i*_ins = i
            i*_rem = l
            r* = r
            c* = Cost(i,j) - savings^r_l
         end if
      end for
   end for
end for
return (c*, r*, i*_ins, i*_rem)
```

If several alternatives for inserting customer j are available then the one with least cost is chosen:

$$(r^*, i^*_{ins}, i^*_{rem}) = \underset{r,i,l}{\operatorname{argmin}}\ Cost(i,j) - savings^r_l \tag{5.32}$$

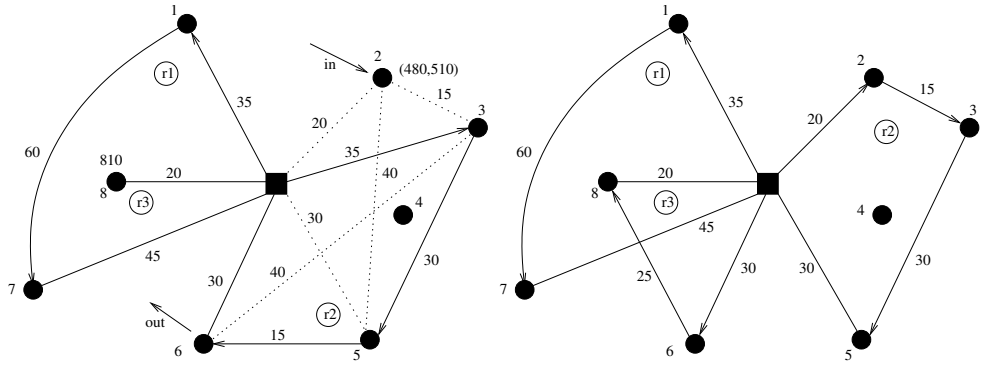
As mentioned before, whether all customers are served after the insertion procedure is finished depends on the order in which the customers are chosen for insertion. The same is true for the post-insertion procedure. Thus, if customer j_{in} cannot feasibly be inserted into the current routes neither directly nor by removing another customer, it is placed in set N'_{uns} – the set of still unserved customers. After all customers of the initial set N_{uns} have been tried for insertion, the algorithm checks whether some of them are still unserved and whether the post-insertion procedure has to be re-run – lines 29 to 37 of the pseudocode in algorithm 3. Whether the post-insertion has to be performed again is controlled by the *repeatProc* parameter. The initial set N_{uns} is compared to the set of still unrouted customers N'_{uns}. If these sets are identical, then the customers in these sets cannot feasibly be inserted into routes. Thus, there is no need to re-run the post-insertion procedure, i.e. $repeatProc = false$. Otherwise, the post-insertion procedure should be re-run on set N'_{uns} and $repeatProc = true$. The value of parameter *repeatProc* is returned to the main programme.

An example of the post-insertion procedure is shown in figure 5.11. The unserved customer 2 is chosen to start the ejection chain. Customer 2 can only be feasibly inserted into route 2 as the first customer on the route (see appendix B for details), and the best alternative is pushing customer 6 out of the route – see table 5.1 below.

Customer Id	Position in the route, i	$Savings_i$	Cost of inserting customer 2	Cost–Savings
3	1	35+30-30=35	20+40-30=30	-5
5	2	30+15-40=5	20+15-35=0	-5
6	3	15+30-30=15	20+15-35=0	-15*

Table 5.1: Savings and insertion cost for the ejection procedure. * indicates the best alternative, i.e. the ejection move with minimal cost.

Customer 6 can be inserted directly into route 3 terminating the ejection chain.



(a) Routes before starting the ejection chain.

(b) Routes after the ejection chain move.

Figure 5.11: Example of an ejection chain move.

5.2.1 Cycling issue in the post-insertion procedure

Consider the following situation: Customer 6 is pushed out of route 2 due to the insertion of customer 2. A new insertion place for customer 6 is now to be found. Obviously, one of the alternatives could be to insert customer 6 in its old position in route 2 and to push customer 2 out again. If that is the best (the cheapest) option, cycling will occur: The algorithm will alternate between pushing one of the customers 2 and 6 out of the route and inserting the other one. Thus, the ejection will never terminate.

To avoid this type of cycling, the following rule can be applied to the post-insertion procedure: When searching for a new insertion place for the customer just pushed out of some route r', only the routes $r \neq r'$ are considered.

However, applying this rule will not be enough in some cases, e.g. in the problem considered in figure 5.12.

Cycling occurs when customer 5 is pushed out of route r_1 in step e). The resulting routes are the same routes as in step a). As there is no rule prohibiting the algorithm from performing such a move, cycling occurs. To avoid this a so called *tabu list* is created and maintained throughout the post-insertion procedure. The tabu list consists of pairs of customers which are forbidden to appear again in the routes. For example, in figure 5.12 after step a) the tabu list will contain two entries – (4,3) and (3,0). The list is expanded during the post-insertion procedure. Each time a customer i is pushed out of the route, the entries added to the tabu list are $(i-1, i)$ and $(i, i+1)$.

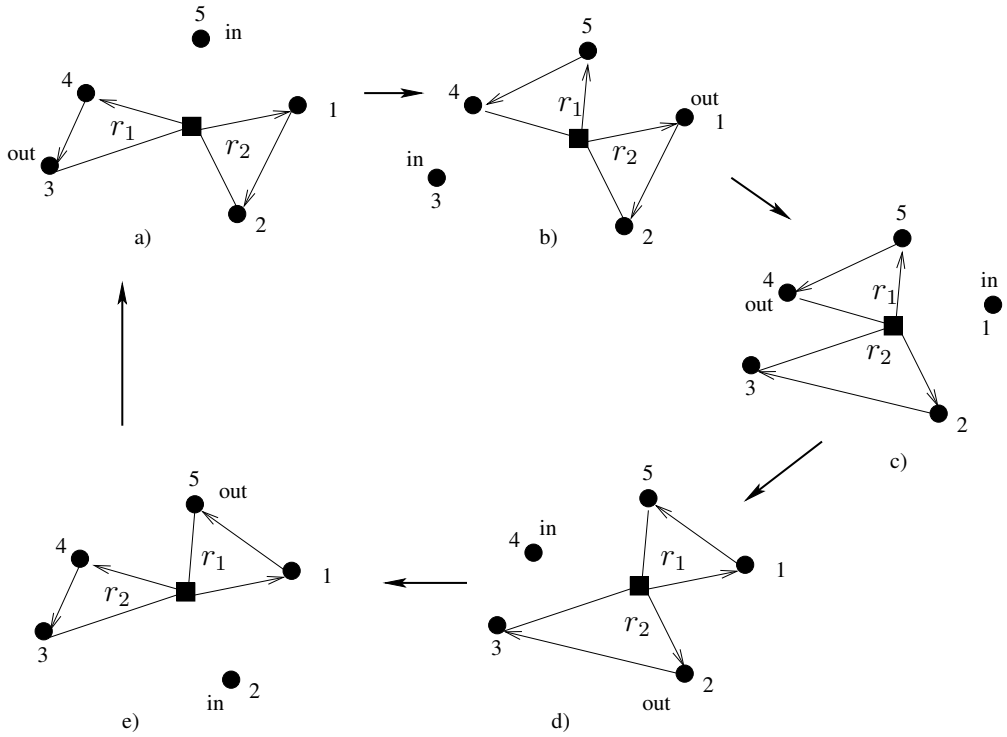


Figure 5.12: Example of cycle in the post-insertion procedure.

In steps d) to e) the tabu list will prevent the algorithm from constructing the same route as one of the routes in step a). Instead, the next-best alternative will be used. In that case, a different set of routes will be produced and cycling will be avoided.

5.2.2 Time complexity of the post-insertion procedure

In the following, the time complexity of the post-insertion algorithm presented in algorithm 3 is discussed.

In each major iteration (the *for*-loop in lines 5 to 28), an unserved customer is selected from set N_{uns}. Let n_{uns} denote the number of unserved customers, i.e. $n_{uns} = |N_{uns}|$. Then, there are n_{uns} major iterations.

Furthermore, let N_{while} denote the length of the ejection chain, that is the number of times the *while*-loop in lines 7 to 27 is performed.

Using this notation, the time complexity of the post-insertion procedure can

be estimated as follows:

$$T_{post\,insertion} = n_{uns} \cdot N_{while} \cdot [T_{direct\,insertion} + T_{ejection\,move}] \tag{5.33}$$

$T_{direct\,insertion}$ in equation (5.33) is the time it takes to check whether a customer can be directly inserted into some route, and to insert the customer if a feasible insertion place has been found. As it was shown in section 5.1, the feasibility check can be performed in constant time. As there are $O(n)$ possible insertion places, the time complexity of the *ExistFeasible()*-procedure is $O(n)$. The physical insertion of a customer into a route takes constant time, and updating the route can be performed in linear time, $O(n)$. Thus, $T_{direct\,insertion}$ takes $O(n)$ time whether an insertion is performed or not.

$T_{ejection\,move}$ is the time it takes to find a route where a customer can be inserted by pushing out another customer and, if such move is feasible, the time it takes to perform this move. That is,

$$\begin{aligned} T_{ejection\,move} =& T(FindRemovalPlace()) + T(Remove()) + \\ &+ T(Insert()) + T(Update()) \end{aligned}$$

To estimate the time complexity of *FindRemovalPlace()*, consider the pseudocode for the procedure shown in algorithm 4. Based on each route r, a number of different routes are constructed by removing one of the customers from the original route. The number of new routes constructed in this process equals to the number of already routed customers, $O(n)$. The following approach is applied in this thesis when removing a customer from a route: If customer i has to be removed from route r, the *Remove()*-procedure will start by removing all the customers from the route and then inserting one customer at a time except customer i. As there are $O(n)$ customers in the route, and updating of the route when inserting a customer is done in linear time, the *Remove()*-procedure can be performed in time $O(n^2)$. Evaluating whether a new customer j can feasibly be inserted into the route can be performed in time $O(n)$, while computing the cost of the move is done is constant time. Thus, the time complexity of the *FindRemovalPlace()*-procedure is calculated as $O(n) \cdot [O(n) + O(n^2)] = O(n^3)$, which gives following time for the ejection move:

$$T_{ejection\,move} = O(n^3) + O(n^2) + O(1) + O(n) = O(n^3)$$

Thus, the time it takes to perform one iteration of the *while*-loop is $T_{while} = O(n) + O(n^3)$, which leads to an overall complexity of

$$\begin{aligned} T_{post\,insertion} &= n_{uns} \cdot N_{while} \cdot [O(n) + O(n^3)] \\ &= n_{uns} \cdot N_{while} \cdot O(n^3) \end{aligned} \tag{5.34}$$

The next step is to find out how many times the *while*-loop is performed in each major iteration of the post-insertion procedure.

In the best case, the length of the ejection chain is 2, i.e. an unserved customer j is inserted into some route by pushing out another customer i, which can be directly inserted into some other route. In the worst case, an unserved customer j is inserted into some route by pushing another customer out and then pushed out of this route again later during the iteration process. In this case a new insertion place has to be found for customer j. Because of the tabu list, customer j cannot be inserted in the position from which it was removed (i.e. the least cost alternative). Thus, the second best alternative for inserting this customer is chosen. If customer j is pushed out of the routes again later in the process, the third best alternative is chosen and so on. If all alternatives have been used, the ejection chain will terminate on the condition that customer j cannot be inserted into the current set of routes. The upper bound on the number of possible alternatives for insertion of the unrouted customer j is $O(n^2)$. This can be derived as follows: Customer j can potentially be inserted at all positions of each route constructed by removing one of the customers from the original route. The number of new constructed routes is $O(n)$ and the number of possible insertion places is also $O(n)$, which gives $O(n^2)$ alternatives. However, due to the tabu list, only $O(n)$ of these alternatives can be used. Thus, each customer can only be removed from and inserted into the routes $O(n)$ times. And as there are $O(n)$ already routed customers, the upper limit on the number of times the *while*-loop is executed is $O(n^2)$.

The overall complexity of the post-insertion procedure becomes

$$T_{post\,insertion} = n_{uns} \cdot O(n^2) \cdot O(n^3) = O(n^5) \tag{5.35}$$

5.3 Embedded Improvement

In the following, the approach used in this thesis to improve the solutions obtained from the route construction procedure is described. It is a common practice when solving vehicle routing problems to apply a two-phase approach in which a route construction procedure is followed by a route improvement procedure. It has been shown that a greater improvement can be achieved by embedding the improvement process into the route construction procedure [31].

The main idea of the embedded improvement approach used in this thesis is to apply the route improvement techniques to the partially constructed

routes each time a certain number of customers have been inserted into the routes. Let f denote the frequency of the improvement, i.e. the number of customers to be inserted into the routes before the improvement procedure is applied. The size of f affects both the solution quality and the computation time of the route generation phase. Small values of f mean more frequent improvement and tend to produce solutions of higher quality at the expense of longer computation time. In the work of Russel it is shown that the performance of embedded improvement is not always monotonic with respect to the frequency of improvement. Thus, there is no specific value of f which is optimal for all types of problems. Experiments conducted in the work of Russel show that the most effective results can be achieved by applying the improvement procedure after every $f = 0.1n \ldots 0.16n$ customers have been routed. Based on this results, the value of $f = 0.1n$, where n is the number of customers, is used in this thesis.

A number of different improvement methods, or improvement moves, have been suggested in the literature. These methods can be applied to one route – intra-route improvement, or to a number of routes simultaneously – inter-route improvement. In this thesis two different inter-route improvement moves are developed and tested on the problem.

5.3.1 Swap move

A *swap move* is performed on a pair of routes by exchanging two nodes between the routes. The move is only accepted if the quality of the solution is improved. Let i and j denote two customers considered for a swap move. Let r_i and r_j denote the original routes for customer i and j. The routes obtained by removing i and j from their original routes are denoted by r'_i and r'_j. The cost of the move $sCost$ is then computed as ($iCost$ stands insertion cost):

$$sCost = iCost(i, r'_j) + iCost(j, r'_i) - savings_i^{r_i} - savings_j^{r_j} \tag{5.36}$$

If $sCost < 0$, the move is accepted, and two customers are swapped between the routes. Otherwise, the move is rejected.

When performing the improvement procedure, the following strategy is applied to each route: Two customers are considered for removal from the route. The *FindRemovalPlace()*-procedure described in algorithm 4 is applied to each considered customer j and each route $r \neq r_j$. If customer j can be inserted into another route r_i by pushing out some customer i, it is checked whether this customer i can be inserted into j's original route r_j.

If such a move is feasible, its cost is calculated using equation (5.36). The feasible move with the lowest cost is then performed. Note, that all positions of route r_i are checked when evaluating feasibility of insertion for customer j, and not only the one from which customer i is removed, and visa versa.

Two different criteria were considered for choosing two customers which are candidates for being removed from the route. By applying the first one, customers with the largest waiting time are considered for removal. The aim of this criterion is to minimize the waiting time on the route, hence minimizing route duration. However, inspection of the routes constructed by the insertion heuristic showed that removing a customer with a large waiting time does not imply significant decrease in total waiting time on the route. This can be explained as follows: If there is a large waiting time on the route at customer i, it is typical that the next customer in the route is located close to the customer i and that their time windows are similar. That means that removing customer i will just 'push' the waiting time to the next customer. Another criterion focuses on minimizing travel time. Thus, the customers considered are the ones by removal of which the largest savings in travel time can be obtained. For each route, information about customers (positions in the route) causing the largest savings if removed from the route is maintained during the insertion procedure. Thus, no extra time is spent on searching after these customers each time the embedded improvement procedure is called.

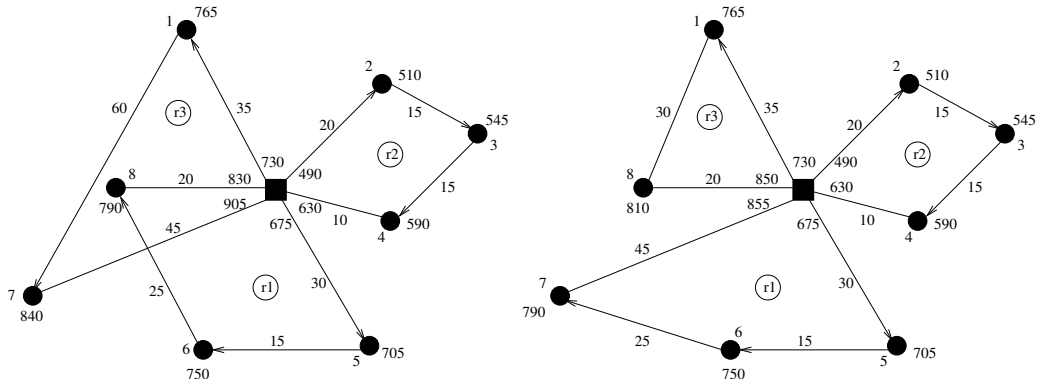
In section 5.2.2 the time complexity of the *FindRemovalPlace()*-procedure was determined to be $O(n^3)$. Evaluating feasibility of insertion of customer i in route r_j is performed in time $O(n)$. Thus, finding the best feasible move takes $2 \cdot [T(findRemovalPlace(R, j)) + T(Feasible(i, r_j))] = 2 \cdot [O(n^3) + O(n) = O(n^3)$. If such move is found, performing it can be done in time $2 \cdot [O(n^2) + O(n)] = O(n^2)$. Let m' denote the number of routes constructed by the time the embedded improvement procedure is called. As each route is considered for improvement, the whole procedure is performed in time $m' \cdot [O(n^3) + O(n^2)]$.

An example of a swap move can be seen in figure 5.13.

Consider route 3 in figure 5.13 which consists of only two customers. Thus, both of them are considered for removal. Savings in travel distance are calculated to be $60 + 35 - 45 = 50$ and $60 + 45 - 35 = 70$ for customer 1 and 7 respectively. First, customer 7 with the largest savings is considered. Insertion of customer 7 into route 2 is infeasible due to the customer time windows. For route 1 and customer 7 the following information is evaluated to find the best move:

Customer id	$savings_i^{r1}$	iCost(i,$r3'$)	iCost(7,$r1'$)	sCost
5	15	45	75	45+75-15-70=35
6	5	Infeasible	–	–
8	15	40	15	40+15-15-70=–30*

Table 5.2: Evaluation example of the swap move. The best feasible move is marked with *.



(a) Total cost: 470 minutes.

(b) Total cost: 440 minutes.

Figure 5.13: Example of the swap improvement move.

If customer 1 is removed from route 3, it cannot feasibly be inserted into another route. Insertion into route 2 will fail due to the time window constraint, and insertion at any position of route 1 will fail due to the exceeded shift time limit. Thus, the swap move performed is removing customer 8 from route $r1$ and inserting it into $r3$ by removing customer 7, which is then inserted into $r1$. As it can be seen from the figure, the total cost of the routes, i.e. sum of all route durations, has decreased as a result of the move.

5.3.2 Re-insertion move

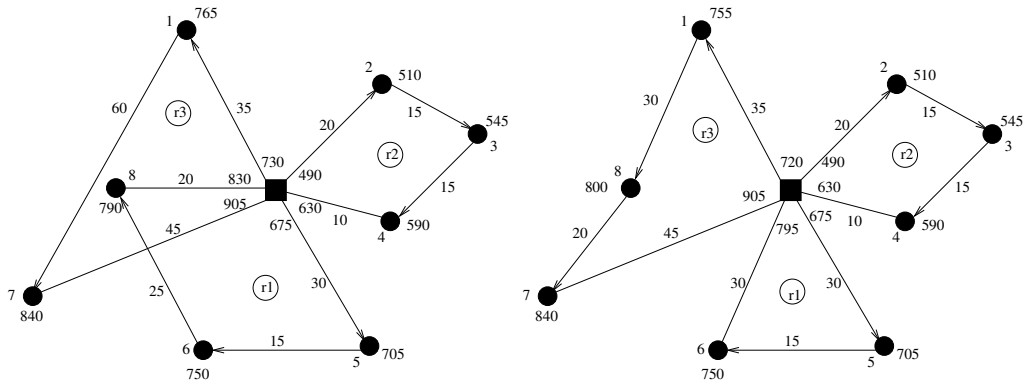
A *re-insertion move* is performed by removing a customer from one route and inserting it into another route. Let j and r_j denote a customer considered for a move and its original route. If r'_j is the route where customer j can be feasibly inserted, the cost of the move can be calculated as follows:

$$riCost = iCost(j, r'_j) - savings_j^{r_j} \tag{5.37}$$

The move is accepted only if $riCost < 0$, i.e. if the solution is improved by the move. The same strategy as for the swap move is applied when performing a re-insertion move: For each route two customers with the largest savings in travel time are considered. For each of these customers insertion in every other route is evaluated and the cost of the move is computed according to equation (5.37). The feasible move with the lowest cost is then performed.

Evaluating the feasibility of inserting customer j in all routes $r \neq r_j$ can be done in $O(n)$ time. Thus, finding the best feasible move takes $2 \cdot O(n) = O(n)$. Performing the move, i.e. removing the customer, inserting it into another route and updating both routes, can be done in time $O(n^2)$, due to the time complexity of the *Remove*-procedure. As for the swap move, all the routes are considered for improvement, and the whole improvement procedure is performed in time $m' \cdot (O(n) + O(n^2))$, where m' is the number of routes at the time.

In figure 5.14 an example of a re-insertion move is shown. Consider route 1 where removing customer 5 and 8 gives the largest savings of 15.



(a) Total cost: 470 minutes.

(b) Total cost: 445 minutes.

Figure 5.14: Example of the re-insertion improvement move.

If customer 5 is removed, it cannot feasibly be inserted into any other route. Insertion into route 2 is impossible due to the vehicle capacity constraints, and insertion into route 3 will fail due to the time window constraints. If customer 8 is removed, it can only be inserted into route 3 at position 2 yielding the following cost of the move: $riCost = iCost(8, r_3) - savings_8^{r_1} = -25$. As it can be seen from figure 5.14.b the total cost of the solution has decreased by exactly this amount.

5.4 Generation of the Pool of Routes

As mentioned earlier, the aim of the route generation phase is to construct a large pool of routes which is then used as input for the Lagrangian heuristic.

After the insertion heuristic with embedded improvement has finished, the improvement procedure is applied to the set of constructed routes. To increase the number of resulting routes, both old and new improved routes are stored in the pool of routes. As the number of vehicles is limited, the number of routes produced by a single execution of the route generation phase is at most $2m$, where m is the number of vehicles. Thus, the route generation phase has to be repeated a number of times to obtain a sufficient number of routes for the next phase of the solution approach.

After each execution run of the route generation phase, the original problem data is permutated in the following way: P pairs of customers with the shortest distances between them are chosen from each constructed route. The distances between these customers are then set to a large number. The purpose of such a permutation is to prevent these customers from appearing together in the routes. Hereby a set of routes different from the previous ones are constructed.

The larger the value of P is, the more constrained the route construction problem becomes in the next execution of the route generation phase. Thus, too large values of P can potentially result in a set of routes with some customers left unserviced. On the other hand, small values of P will increase the probability of producing routes which are similar to the other routes in the pool.

5.5 Route Generation Phase – Summary

The general framework of the route construction phase can be summarized as follows:

a. Perform the insertion heuristic with embedded improvement after every $0.1n$ customers have been routed.

b. Run the post-insertion procedure to handle the unserviced customers, if any.

c. Apply the improvement procedure to the resulting set of routes.

d. Permutate the problem data based on the constructed routes and repeat the route generation phase

As mentioned above the route generation phase has to be executed a number of times to produce a large number of good quality routes. The obtained pool of routes is used as input for the second phase of the solution approach which is presented in the next chapter.

Chapter 6

A Lagrangian-based Heuristic for the Set Covering/Packing Problem Formulation

In this chapter the second phase of the solution approach is described. In section 6.1 the VRPTWSTL is formulated as a mixed set covering/packing problem. Section 6.2 presents a Lagrangian-based heuristic used to solve the problem.

6.1 Set Covering/Packing Problem Formulation

Set covering, set packing and set partitioning problem formulations can be applied to a variety of problem types such as delivery and routing problems, scheduling and location problems. For the VRPTW the solution methods based on the set partitioning problem formulation usually involve column generation or decomposition methods. In the work of Desrochers et al., a set covering formulation of the VRPTW is used with column generation to generate optimal or near-optimal solutions [14]. Their approach was able to solve VRPTW instances with up to 100 customer to optimality.

Another approach based on the set covering problem formulation is to generate a large number of feasible routes and then use a heuristic to select the set of minimum cost routes. After a set covering solution is obtained from the heuristic, a set partitioning solution can be constructed by removing multiple covered customers from all but one route. Effective heuristic approaches for

the set covering problem (SCP) based on Lagrangian relaxation have been proposed in the literature: From the quite simple approaches of Beasley [3] and Haddadi [20] to the more sophisticated algorithms of Ceria et al. [11] and Carpara et al. [10] for solving large scale set covering problems arising from crew scheduling at the Italian railways.

A set covering problem formulation for the VRPTW is presented below:

$$\min \quad \sum_{r \in R} c_r \cdot y_r \tag{6.1}$$

$$\text{s.t.} \quad \sum_{r \in R_i} y_r \geq 1 \qquad \forall i \in I \tag{6.2}$$

$$y_r \in \{0,1\} \qquad \forall r \in R \tag{6.3}$$

where R is the set of columns (feasible routes), I is the set of rows (customers) and R_i is the set of columns covering row i. The binary variable y_r indicates whether column r is in the solution or not. The advantage of such formulation is that it is generic and can be applied to the different problems, as all the problem specific constraints are 'hidden' in the route generation phase.

The VRPTWSTL considered in this thesis is further complicated by different vehicle types and the fact that there is a limited number of vehicles of each type. Thus, the SCP formulation from above has to be extended to ensure that the number of available vehicles is not exceeded in the final solution. Adding these extra constraints lead to a mixed set covering/packing model formulation for the VRPTWSTL – equations (6.4)–(6.7) below.

$$\min \quad \sum_{r \in R} c_r \cdot y_r \tag{6.4}$$

$$\text{s.t.} \quad \sum_{r \in R_i} y_r \geq 1 \qquad \forall i \in I^1 \tag{6.5}$$

$$\sum_{r \in R_i} y_r \leq m_i \qquad \forall i \in I^2 \tag{6.6}$$

$$y_r \in \{0,1\} \qquad \forall r \in R \tag{6.7}$$

The set of rows $I^1 = \{0, \ldots n-1\}$ corresponds to the set of customers N. For each vehicle type an extra row is added to the problem. The set of these rows is then denoted I^2. The number of available vehicles of each type is denoted m_i, and there are L different vehicle types. Each route in R is a feasible route with respect to all constraints of the original problem, i.e. equations (3.9) –(3.17) of the model presented in section 3.1.

The objective (6.4) is to minimize the total route cost. The set covering constraints (6.5) ensure that each customer is covered at least once in the final solution, whereas the set packing constraints (6.6) ensure that the number of available vehicles is not exceeded in the final solution. Finally, constraints (6.7) impose binary restrictions on the decision variables.

To illustrate the problem, consider the example with 3 vehicle types and 8 customers described in the previous section. After the route generation phase, i.e. the insertion heuristic and subsequent improvement procedure, the following routes are constructed:

Vehicle type	Customers in the route
2	2–3–4
1	1–7
3	5–6–8
1	1–8
3	5–6–7

Table 6.1: Set of routes constructed during the route generation phase.

As it can be seen from the table, the number of routes after the route generation phase is 5, while only 3 vehicles are available. This is due to the fact that both old and new improved routes are stored in the pool of routes while performing the improvement procedure, as mentioned in section 5.4.

Based on these routes, the constraint matrix for the mixed set covering/packing problem (SCPP) can be presented as follows:

		Columns, r						
	Rows, i	0	1	2	3	4		
c1	0		1		1		$\geq$	1
c2	1	1					$\geq$	1
c3	2	1					$\geq$	1
c4	3	1					$\geq$	1
c5	4			1		1	$\geq$	1
c6	5			1		1	$\geq$	1
c7	6		1			1	$\geq$	1
c8	7			1	1		$\geq$	1
vt1	8		1		1		$\leq$	1
vt2	9	1					$\leq$	1
vt3	10			1		1	$\leq$	1

Table 6.2: The constraint matrix for the SCPP.

The three last rows correspond to the set packing constraints (6.6) in the model. There is one row for each vehicle type and for each column (route) there is only a single 1 entry in one of these rows depending on the type of the vehicle driving the route.

The data for the problem considered in this thesis includes 500 customers and 3 vehicles types, see chapter 4 for the details. Thus, the number of rows in the constraint matrix for this problem is 503: 500 set covering rows corresponding to the customers and 3 set packing rows, one for each vehicle type. The number of columns in the matrix is equal to the number of routes produced by the route generation phase.

In the next section the approach applied in this thesis to solve SCPP is described. The general framework is based on the Lagrangian algorithm designed by Beasley [3]. Some of the ideas proposed in Fisher et al. [16] and Carpara et al. [10] are used in the primal heuristic for generating feasible solutions and calculating upper bounds.

6.2 A Lagrangian-based Heuristic

The general framework of the solution approach can shortly be described as follows:

a. Generate a lower bound for the SCPP via Lagrangian relaxation

b. Use a primal heuristic to generate a feasible solution and an upper bound for the SCPP

c. Attempt to maximize the lower bound via subgradient optimization

The algorithm terminates if either an optimal (or near-optimal) solution to the problem is found or by another stopping criterion – e.g. the number of subgradient iterations. In the following, the three major components of the Lagrangian heuristic are discussed.

6.2.1 Lower bound generation

The Lagrangian relaxation of the set covering/packing problem described above is discussed in the following.

Both types of constraints are chosen for relaxation. In order to bring them into objective function Lagrangian multipliers are attached to each constraint:

λ_i – for the set covering constraints, $\lambda_i \geq 0$, $\forall i \in I^1$
μ_i – for the set partitioning constraints, $\mu_i \leq 0$, $\forall i \in I^2$

The *Lagrangian lower-bound program* (LLBP) is given by

$$\min \quad \sum_{r \in R} c_r \cdot y_r + \sum_{i \in I^1} \lambda_i \cdot (1 - \sum_{r \in R_i} y_r) + \sum_{i \in I^2} \mu_i \cdot (m_i - \sum_{r \in R_i} y_r) \tag{6.8}$$

$$\text{s.t.} \quad y_r \in \{0,1\}, \qquad \forall r \in R \tag{6.9}$$

i.e.

$$\min \quad \sum_{r \in R} (c_r - \sum_{i \in I^1} \lambda_i - \sum_{i \in I^2} \mu_i \cdot m_i) \cdot y_r + \sum_{i \in I^1} \lambda_i + \sum_{i \in I^2} \mu_l \cdot m_i \tag{6.10}$$

$$\text{s.t.} \quad y_r \in \{0,1\}, \qquad \forall r \in R \tag{6.11}$$

Let C_r be defined as $C_r = c_r - \sum_{i \in I^1} \lambda_i - \sum_{i \in I^2} \mu_i \cdot m_i$. The LLBP then becomes

$$\min \quad \sum_{r \in R} C_r \cdot y_r + \sum_{i \in I^1} \lambda_i + \sum_{i \in I^2} \mu_i \cdot m_i \tag{6.12}$$

$$\text{s.t.} \quad y_r \in \{0,1\}, \qquad \forall r \in R \tag{6.13}$$

Based on the formulation (6.12)–(6.13), the solution to the LLBP can be found by inspection. Let Y_r denote the solution values of y_r. Then

$$Y_r \ = \begin{cases} 1 & \text{, if } C_r \leq 0 \\ 0 & \text{, otherwise} \end{cases} \tag{6.14}$$

and the solution value of the LLBP is given by:

$$Z_{LB} = \sum_{r \in R} C_r \cdot Y_r + \sum_{i \in I^1} \lambda_i + \sum_{i \in I^2} \mu_i \cdot m_i \tag{6.15}$$

The solution value of the LLBP, Z_{LB}, is a lower bound on the optimal solution to the original problem SCPP. Clearly, one is interested in finding a lower bound that is as close as possible to the optimal integer solution. This problem is called the *Lagrangian dual problem* and involves finding multipliers such that

$$\max_{\substack{\lambda \geq 0 \\ \mu \leq 0}} \left\{ \begin{array}{ll} \min & \sum_{r \in R} C_r \cdot y_r + \sum_{i \in I^1} \lambda_i + \sum_{i \in I^2} \mu_i \cdot m_i \\ \text{s.t.} & y_r \in \{0,1\}, \qquad \forall r \in R \end{array} \right\}$$

To find the initial values of the Lagrangian multipliers, a property of the SCPP is exploited. As the solution to the LLBP for all possible values of the Lagrangian multipliers is unchanged by replacing the integrality constraint $y_r \in \{0, 1\}$ by its linear relaxation $0 \leq y_r \leq 1$, the Lagrangian relaxation of the SCPP is said to have the *integrality property*. Due to the integrality property of SCPP the following holds: Any optimal solution $\mathbf{u}^*$ to the dual of the *Linear Programming* (LP) relaxation of SCPP is also an optimal solution to the Lagrangian lower-bound program [4].

The dual problem of the LP relaxation of the SCPP is given by:

$$\max \quad \sum_{i \in I^1} u_i + \sum_{i \in I^2} m_i \cdot u_i \tag{6.16}$$

$$\text{s.t.} \quad \sum_{i \in I_r} u_i \leq c_r \qquad \forall r \in R \tag{6.17}$$

$$u_i \geq 0 \qquad \forall i \in I^1 \tag{6.18}$$

$$u_i \leq 0 \qquad \forall i \in I^2 \tag{6.19}$$

In the above model I is the set of all matrix rows, $I = I^1 \cup I^2$. The set I_r is then defined as the set of rows covered by column r.

Two approaches for solving the dual problem are implemented and tested in this thesis (for more details see appendix C):

- The dual greedy heuristic proposed by Fisher and Kedia [16] is used to generate a feasible solution to the problem
- The optimization package COIN is used to solve the problem to optimality

For large scale instances the second approach can be time consuming, but as the problem considered in this thesis is of limited size, COIN is able to find an optimal solution in reasonable time. On the other hand, the problem size is sufficiently large to note the difference in quality of solutions produced by these two approaches. Tests performed during the implementation of both approaches showed that the performance of the Lagrangian heuristic was better when using the solution obtained by COIN as the initial values for the Lagrangian multipliers rather than the solution obtained from the dual heuristic. This is not surprising as solving the dual problem to optimality leads to a better lower bound compared to the heuristic approach. Furthermore, in some cases the lower bound obtained by COIN is equal to

the optimal solution value of the original integer problem, i.e. no *duality gap* exists between the original and the LP dual problem.

In the final implementation the second approach is used to find the initial values of the Lagrangian multipliers. The first $|I^1|$ entries of the optimal solution vector obtained from COIN correspond to the multipliers of the set covering constraints, λ_i, for all $i \in I^1$. The last $|I^2|$ entries of the solution vector are the multipliers for the set packing constraints, μ_i, for all $i \in I^2$. Based on the multiplier values, the Lagrangian costs C_r are computed, the solution to the LLBP is determined according to the rule in (6.14), and the lower bound is calculated as in (6.15).

Let Z_{max} represent the maximum lower bound found. Initially, Z_{max} is set to $-\infty$ and then updated in each iteration of the Lagrangian heuristic according to the following rule: $Z_{max} = max(Z_{max}, Z_{LB})$.

6.2.2 Primal heuristic

The purpose of the primal heuristic is to construct a feasible solution to the problem and to compute the upper bound on the value of the optimal solution. In the algorithm described by Beasley [3], a feasible solution to the problem can be constructed by modifying the solution of the LLBP. However, due to the set packing constraints another approach inspired by the work of Fisher and Kedia [16] is used in this thesis. The problem considered by Fisher et al. is a mixed set covering/partitioning problem, but in their primal heuristic the partitioning constraints are treated as packing constraints. The main difference and challenge of the problem considered in this thesis compared to the problem in Fisher et al. is that the right hand side of the packing constraints can be any positive integer, and not only 1.

The primal heuristic starts by setting y_r to 0 for all $r \in R$. In each iteration y_r for a selected column r is incremented to 1 to satisfy the covering constraints while keeping $\sum_{r \in R_i} y_r \leq m_i$ for all $i \in I^2$. Given the current partial solution $\mathbf{y}$, the following is defined:

$U(\mathbf{y})$	set of uncovered rows, $U(\mathbf{y}) = \{i \in I : \sum_{r \in R_i} y_r = 0\}$
$S(\mathbf{y})$	set of already covered rows, $S = I \setminus U(\mathbf{y})$.
$I_r(\mathbf{y})$	set of rows covered by column r and not covered in the solution yet, $I_r(\mathbf{y}) = I_r \cap U(\mathbf{y})$
cvr_i	number of times row i is covered in the solution, for all $i \in I^2$
$J(\mathbf{y})$	set of columns that can feasibly be selected into solution, $J(\mathbf{y}) = \{r \in R : cvr_{I_r \cap I^2} + 1 \leq m_{I_r \cap I^2}\}$

In addition to the above notation let F be the set of solution columns selected by the primal heuristic and Z_{UB} be the value of the best feasible solution found so far. Z_{UB} is the upper bound on the value of the optimal solution to the original problem. Initially $Z_{UB} = \infty$, but the value is updated each time the primal heuristic is applied to the problem.

A pseudocode for the primal heuristic is shown in algorithm 5. The following pre- and post-processing procedures are applied in the primal heuristic:

- If there are any rows covered by only one column, these columns are included in the final solution before any other columns.
- After the final solution has been constructed, it is checked for redundant columns. A column is redundant if the solution obtained by removing the column is still a feasible solution for the SCPP.

Algorithm 5 Pseudocode for the primal heuristic.

1: $U(\mathbf{y}) = I$
2: $S(\mathbf{y}) = \emptyset$
3: $cvr_i = 0, \quad \forall i \in I^2$
4: $J(\mathbf{y}) = R$
5: $F = \emptyset$
6: **for** $\forall i \in \{i \in I : |R_i| = 1\}$ **do**
7: $\quad y_r^* = 1$, $J(\mathbf{y}) = J(\mathbf{y}) \setminus \{r^*\}$ and $cvr_{I_r \cap I^2} = cvr_{I_r \cap I^2} + 1$
8: $\quad$ Update sets $U(\mathbf{y})$, $S(\mathbf{y})$ and $J(\mathbf{y})$
9: **end for**
10: **while** $U(\mathbf{y}) \neq \emptyset$ or $J(\mathbf{y}) \neq \emptyset$ **do**
11: $\quad$ Select the best column r^* according to some rule
12: $\quad y_r = 1$, $F = F \cup \{r^*\}$ and $cvr_{I_r \cap I^2} = cvr_{I_r \cap I^2} + 1$
13: $\quad$ Update sets $U(\mathbf{y})$, $S(\mathbf{y})$ and $J(\mathbf{y})$
14: **end while**
15: remove redundant columns from the solution F
16: $Z_{UB} = min(Z_{UB}, \sum_{r \in F} c_r)$

The key step of the primal heuristic is to select the best column among all the other columns to add to the solution. Several different approaches for selecting columns have been proposed in the literature. Some of them just choose a column with the lowest cost c_r or with the lowest cost per covered row $\frac{c_r}{|I_r|}$. The most successful of these methods are the ones taking into account the dual information associated with the still uncovered rows [10].

Different rules implemented and tested in this thesis are discussed below:

a) Fisher and Kedia [16] suggest first choosing an uncovered row $i^* \in U(\mathbf{y})$ to maximize $u_i \cdot |R_i|$. No further explanation is presented for this criterion. After some row i^* has been chosen, one of the columns covering that row will be selected into the solution. The best column is determined as follows:

$$r^* = \underset{r \in R_{i^*} \cap J(\mathbf{y})}{\operatorname{argmin}} \frac{(c_r - \sum_{i \in I_r(\mathbf{y})} u_i)}{|I_r(\mathbf{y})|}$$

b) Carpara, Fischetti and Toth [10] use a similar approach, except that any column $r \in J(\mathbf{y})$ can be selected as the next solution column, i.e. they do not restrict the search to a set of columns covering only one selected row.

Let γ_r be defined as follows:

$$\gamma_r = \frac{(c_r - \sum_{i \in I_r(\mathbf{y})} u_i)}{|I_r(\mathbf{y})|}$$

The cost σ_r for each column is then calculated as

$$\sigma_r = \begin{cases} \frac{\gamma_r}{|I_r(\mathbf{y})|} & \text{, if} \gamma_r > 0 \\ \gamma_r \cdot |I_r(\mathbf{y})| & \text{, otherwise} \end{cases}$$

c) The most critical row i^* is chosen first, i.e. the row covered by the least number of columns. Afterwards a column is chosen according to rule b).

d) The columns are selected according to rule a) but without choosing a row first.

All the implemented rules give priority to the columns having low cost and covering a large number of uncovered rows.

It is important to keep in mind that if the primal heuristic terminates because $J(\mathbf{y}) = \emptyset$, the obtained solution may not be feasible, i.e. some rows are still left uncovered. This proved to be an issue for the problem considered in this thesis. Computational experiments showed especially poor performance of rule a). Applying this rule leads to selecting a row with no more feasible columns to choose from already in the early iterations of the primal heuristic.

In fact, none of the above rules could produce a feasible solution for the problem considered in this case. Further investigations revealed the following two issues which both stemmed from the route construction phase of the solution approach.

The routes generated during the route construction phase proved to be quite similar. Two routes are considered similar if they have a prespecified number of customers in common. For the problem considered in this thesis it was checked how many routes obtained from the route generation phase had at least 3 customers in common (the average route length was 8 customers). It was found that 81% of all generated routes had at least one similar route, and 31.5% had at least 3 similar routes. Thus, choosing a new solution column quite often implied an overlap, i.e. some of the rows covered by the column were already covered by other columns in the solution. Such overlaps are allowed according to the set covering constraints, i.e. each row can be covered more than once in the final solution. However, an overlap can also be interpreted as an unnecessary use of the vehicle capacity. As the number of vehicles is limited for the problem considered here, a large amount of overlap leads to lack of capacity for covering the rest of the rows.

Another issue arises if a customer i' is only covered by the routes executed by vehicles of only one type. As there is a limited number of vehicles of each type, this represents a problem. Assume that during the iteration process of primal heuristic all m_i columns having an entry of 1 in the row $i \in I^2$ are selected and none of them covers the row corresponding to the customer i'. Then row i' can not be covered in the final solution, as all the columns covering i' have been marked as infeasible for selection due to the set packing constraints.

To avoid the first problem the following approach is applied when choosing the columns: An upper bound on the overlap that can be allowed for the column to be selected is defined. If the column found by applying one of the above rules has an overlap greater than the allowed value, the algorithm tries to find another column to select. The selection rule is applied to the rest of the columns until a column with none or less than the allowed overlap is found. Ties are broken to maximize the number of new covered rows, i.e. $|I_r(\mathbf{y})|$. If no columns satisfying these conditions can be found, a column with the least amount of overlap is chosen.

To solve the problem with the customers covered by only one or several but not all vehicle types, the following extensions are introduced in the route construction phase: After a set of routes for all available vehicles is generated, a list of uncovered customers is constructed for each vehicle type. A set of

extra routes covering the customers in these lists is then constructed for each vehicle type and added to the pool of routes.

Furthermore, the following iterative procedure was developed to produce even more routes: For each vehicle a number of routes equal to the number of customers was initialized, i.e. one route for each customer. In the next iteration, n' customers are considered for insertion at the last position of each route. These customers are the nearest neighbours of the last customer of a route. The routes constructed by choosing the nearest neighbour as the next customer in the route are expected to be of good quality with respect to the objective of the problem considered in this case, i.e. minimizing the travel distance. In the current implementation n' is set to 2. If both nearest neighbours can feasibly be inserted, the original route is transformed into two new routes. If only one of the customers can be inserted, the original route is added to the final pool of routes, and the new transformed route becomes an original route for the next iterations. The procedure is repeated until no more customers can be inserted into the routes.

The drawback of this approach was that the number of routes produced was too large to handle in reasonable time, especially when constructing the constraint matrix for COIN to compute the initial values for the multipliers. Therefore, the approach was modified in the following way: The LP-relaxation of the SCPP with the columns produced only by the route generation phase was solved using COIN. The obtained fractional solution was examined in order to find out which customer sequences were the most common for the solution and therefore had the potential to also be in the integer optimal solution for the problem. For each customer, information about all its successors was stored and used when constructing the additional routes. A nearest neighbour of each customer was only considered for insertion if it was among the successors of that customer in the LP-solution.

Technically, a situation where some of the customers can only be served by the vehicles of one type is possible, e.g. due to the incompatibility of the time windows of these customers and the availability windows for the vehicles. To ensure that such customers are covered in the final solution, the pre-processing procedure is modified in the following way: If a row is covered only by columns of the same type, the best column to cover this row is chosen and included in the solution. Columns are said to be of the same type if they have an entry of 1 in the same set packing row $i \in I^2$.

6.2.3 Subgradient optimization

Each time a feasible solution is obtained from the primal heuristic, the value of Z_{UB} is updated and compared to value of the maximum lower bound Z_{max}.

If there is a gap between these values, the algorithm will try to improve the lower bound via *subgradient optimization*. Subgradient optimization is a procedure which from a set of initial Lagrangian multipliers generates a set of new multipliers in a systematic way.

Before calculating new values for the multipliers, a problem reduction procedure is performed based on the values of the Lagrangian cost C_r and the solution to the LLBP obtained in the lower bound generation phase.

Let P_r denote the lower bound when column r is forced to be in the solution. Initially, P_r is set to c_r for each column $r \in R$. If a particular column r' is forced to be in the optimal solution, the resulting lower bound becomes $Z_{LB} + C_{r'}$ if $Y_{r'} = 0$. The lower bound remains unchanged if column r' is already in the solution to the LLBP, i.e. if $Y_{r'} = 1$. Hence, P_r for all $r \in R$ can be updated using:

$$P_r \quad = \begin{cases} max(P_r, Z_{LB} + C_r) & \text{, if } Y_r = 0 \\ max(P_r, Z_{LB}) & \text{, otherwise} \end{cases}$$

All the columns with $P_r > Z_{UB}$ can be removed from the problem, since the lower bound corresponding to an optimal solution containing these columns is greater than the upper bound. Thus, these columns cannot be in the improved feasible solution. To remove a column from the problem its cost c_r is set from its original value to infinity. In this way the column will not appear in the solution to the LLBP, nor in the solution generated by the primal heuristic.

After the problem reduction procedure has been performed, new values for the Lagrangian multipliers are calculated. First, subgradients G_i are computed for each row using

$$G_i = 1 - \sum_{r \in R_i} Y_r, \qquad \forall i \in I^1 \tag{6.20}$$

$$G_i = m_i - \sum_{r \in R_i} Y_r, \qquad \forall i \in I^2 \tag{6.21}$$

Based on the findings in the work of Beasley [3], the subgradients are adjusted according to the following rule:

$$G_i = 0, \quad \text{if } \lambda_i = 0 \text{ and } G_i < 0, \quad \forall i \in I^1 \tag{6.22}$$

$$G_i = 0, \quad \text{if } \mu_i = 0 \text{ and } G_i > 0, \quad \forall i \in I^2 \tag{6.23}$$

The step size T is defined by

$$T = \frac{\theta(1.05Z_{UB} - Z_{LB})}{\sum_{i \in I}(G_i)^2} \tag{6.24}$$

where $\theta = 2$ initially. If Z_{max} has not increased during the last K iterations of the subgradient procedure, the initial value of θ is halved. Due to the lack of time no experiments are conducted to find the best value of K for the problem under consideration. Both the strategy for reducing θ and the value of $K = 30$ are taken from the original Lagrangian-based heuristic designed by Beasley [3].

The reason for adjusting the subgradients is that under conditions (6.22)–(6.23) the corresponding multipliers λ or μ will not be altered, thus there is no reason to include the term G_i^2 in the calculation of the step size T.

The new values of the Lagrangian multipliers are calculated as follows

$$\lambda_i = max(0, \lambda_i + TG_i) \quad \forall i \in I^1 \tag{6.25}$$

$$\mu_i = min(0, \mu_i + TG_i) \quad \forall i \in I^2 \tag{6.26}$$

and the problem is resolved.

The subgradient optimization procedure is applied to the problem until one of the stopping criteria is met.

- If $Z_{UB} = Z_{max}$ the algorithm is terminated as the optimal solution has been found (the optimal solution is F with the objective value of $Z_{UB} = Z_{max}$).

- If $\sum_{i \in I} G_i^2 = 0$, then the suitable step size T cannot be determined and the procedure is terminated. In that case, if the current LLBP solution is feasible for the original problem, then it is the optimal solution for the original problem. Otherwise, value of Z_{max} presents the lower bound on the optimal solution to the original problem.

- The subgradient procedure is terminated if the value of θ becomes too small, i.e. a significant number of iterations have been performed. In the implemented algorithm, the condition of $\theta \leq 0.0005$ proposed by Beasley is used as one of the stopping criteria.

6.3 The Lagrangian Heuristic – Further Extensions

The tests performed during the implementation of the Lagrangian heuristic described above showed that the primal heuristic was still unable to produce a feasible solution. This caused the whole algorithm to fail already in the first iteration. The primal heuristic failed in spite of constructing the additional routes and trying to minimize the overlap with the other columns in the solution when selecting a new column. The number of uncovered rows after the primal heuristic finished was still surprisingly large. Among the rules described in section 6.2.2, the best performing ones were rule b) and c) with an average of 101 and 64 uncovered rows (of 503). In fact, applying a deterministic rule, in which the column with the highest number of yet uncovered rows was chosen as the best one, yielded the best result with 34 uncovered rows. Clearly, this rule cannot be applied in the final implementation, as it is independent of both the cost of the columns and the multiplier values and will produce the same result in each iteration.

Based on the above, further action was required to solve the infeasibility issue. It was noticed during the tests that the primal heuristic often chose columns with only 2 or 3 covered rows, for example if the columns covering a larger number of rows had an overlap greater that the allowed value. Choosing the columns with a small number of covered rows can also be considered as unnecessary use of capacity. Thus, a situation can occur where a large number of rows are still uncovered and and where there are not enough columns to cover them. The solution to this problem can be to produce the additional routes *'on the fly'* and then restarting the primal heuristic. This approach requires that a problematic situation can be detected during the primal heuristic, i.e. the situation leading to an infeasible solution. In that case the primal heuristic is terminated, and the additional routes are generated only for the customers corresponding to uncovered rows and only using the vehicles which are still available. The additional routes are produced by the insertion heuristic with embedded improvement followed by the post-insertion procedure.

The drawback of this approach is that the overall performance of the Lagrangian heuristic becomes much slower. Typically, the additional routes have to be produced at least once in each iteration. The time it takes to produce the routes is highly dependent on how many customers are yet uncovered and on how many and which type of vehicles are left available. In the worst case the unserviced customers are the those which are difficult to

serve by the remaining vehicles. This can lead to the post-insertion procedure being repeated a number of times and, as discussed in section 5.2.2, the running time of the post-insertion procedure is $O(n^5)$ in the worst case.

Several conditions were tested as a termination criterion for the primal heuristic. The conditions were based on the number of new covered rows that a selected column would bring into the solution compared to the amount of overlap with the other solution columns. The primal heuristic was stopped if the number of the new covered rows was less than nr_{new}, and at the same time the value of overlap was greater than the allowed value ovl_{all}. Different values were tried for these two parameters: $nr_{new} = \{3, 4, 5\}$ and $ovl_{all} = \{0, 1, 2\}$. A condition where the primal heuristic was stopped if the value of overlap exceeded the allowed limit was also tested. The best result was obtained for the most strict condition forbidding overlaps, i.e. where the set covering constraints of the problem were treated as set partitioning constraints. The primal heuristic was terminated as soon as the selected column had at least one common row with the columns already in the solution. Furthermore, it was noticed that the best performing rule for selecting new columns was the one proposed by Carpara, Fischetti and Toth – rule b) described in section 6.2.2. This rule is therefore used in the final implementation of the Lagrangian heuristic.

In algorithm 6 a pseudocode for the final version of the Lagrangian heuristic is presented.

It should be noted that each time new columns are added to the constraint matrix, the initial problem is changed. In some cases the optimal solution to the new problem will be the same as for the initial problem. However, it is reasonable to assume that after a substantial number of columns has been added to the constraint matrix, the problem will change so much that a new optimal solution can now be found. This situation occurs e.g. if the primal heuristic returns a feasible solution with the objective value Z_{UB} less than the value of Z_{max} (the maximum value of the lower bound found so far). However, another optimal solution can exist even if the new upper bound is larger than Z_{max}. Thus, the following strategy is applied in this thesis: Let Z_{min} denote the value of the lowest upper bound found so far. In each iteration the objective value Z_{UB} of the solution returned by the primal heuristic is compared to Z_{min}. If $Z_{UB} < Z_{min}$, there is a possibility that a new optimal solution to the problem can now be found. In this case, a new set of multipliers is generated by COIN based on the new constraint matrix. A new solution to the LLBP is then constructed, and the values of Z_{max} and Z_{min} are updated – lines 14-19 of the pseudocode. It should be noted, that since a new set of multipliers is generated, the multipliers will not be

Algorithm 6 Pseudocode for the Lagrangian heuristic. *ite* is the current iteration, number *max_ite* denotes the upper limit on the number of performed iterations. *status* is a parameter indicating by which stopping criterion the algorithm is terminated. $feasible$ is a parameter indicating whether a feasible solution is found by the primal heuristic.

R – set of initial columns
R_{ite} – set of columns in iteration *ite*
$Z_{max} = Z_{LB} = -\infty$, $Z_{min} = Z_{UB} = \infty$ and $\theta = 2$
The initial multipliers are computed using COIN
$stop = false$, $ite = 0$ and $status = -1$
while $stop \neq true$ & $ite < max\_ite$ **do**
  Solve LLBP with the current set of multipliers and compute Z_{LB}
  $Z_{max} = max(Z_{max}, Z_{LB})$
  $feasible = false$
  while $feasible \neq true$ **do**
    Run the primal heuristic
    if feasible solution is returned **then**
      feasible=true
      if $Z_{UB} < Z_{min}$ **then**
        $Z_{min} = Z_{UB}$
        Compute new values for the multipliers based on R_{ite}
        Re-solve LLBP and set $Z_{max} = Z_{LB}$
        $P_r = c_r$ for all $r \in R_{ite}$
      end if
    else
      if all vehicles are used in the returned solution **then**
        $Z_{UB} = \infty$ and $feasible = true$
      else
        Construct extra routes for the available vehicles and the remaining customers
      end if
    end if
  end while
  if $Z_{max} = Z_{UB}$ **then**
    $status = 0$
    $stop = true$
  else
    Update P-cost and fix columns to 0
    Calculate subgradients G_i for all $i \in I$
    if $\sum_{i \in I} G_i^2 = 0$ **then**
      $stop = true$ and $status = 1$
    else
      if $\theta \leq 0.005$ **then**
        $stop = true$ and $status = 2$
      else
        Calculate the step size T and update the multipliers
      end if
    end if
  end if
end while

re-computed again in this iteration, i.e. updating procedure in line 40 of the pseudocode will not be performed.

If the primal heuristic returns an infeasible solution, it is accepted but the upper bound is set to infinity – lines 20-22 of the pseudocode.

6.4 Summary

In this chapter the second phase of the solution approach used in this thesis has been described. An overview over both phases of the solution approach is given in figure 6.1.

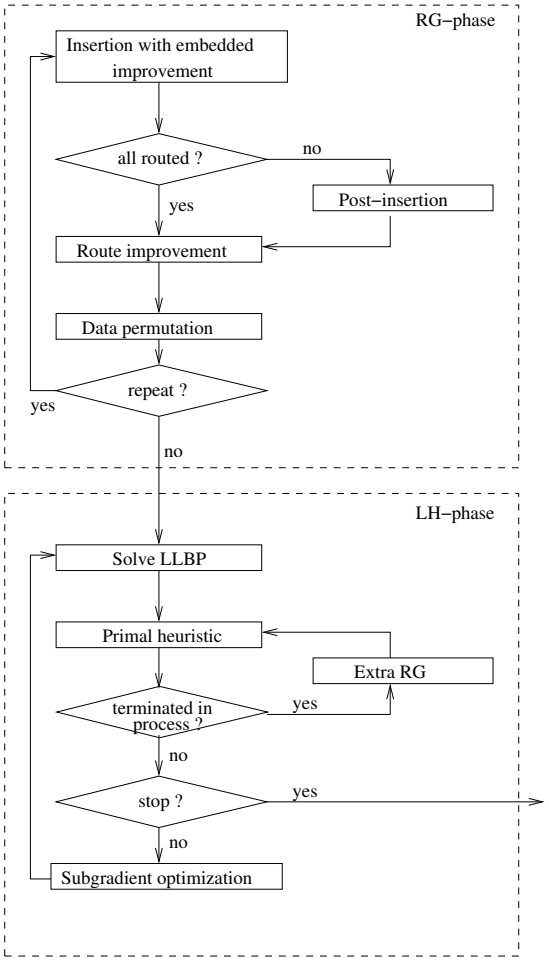


Figure 6.1: The solution approach – overview. *RG* is the route generation phase. *LH* is the Lagrangian heuristic phase.

The general framework of the Lagrangian heuristic originally proposed by Beasley was modified to take the set packing constraints into account. During the implementation of the heuristic, it was tested both on the example described in the beginning the chapter and on the problem considered in this thesis.

The Lagrangian heuristic was able to find the optimal solution for the example problem introduced in the beginning of the chapter. This is not surprising due to the size of the problem in the example. However, for the problem considered in this thesis the Lagrangian heuristic failed to produce feasible solutions. This led to the further extensions of the heuristic, i.e. generation of new columns (routes) *'on the fly'*. To investigate the performance of the Lagrangian heuristic a number of test problems were constructed based on the data set provided by Transvision A/S. In the next chapter the approach used to generate the test problems is described. Chapter 8 presents the results obtained by the Lagrangian heuristic both for the generated data sets and for the problem considered in this thesis.

Chapter 7

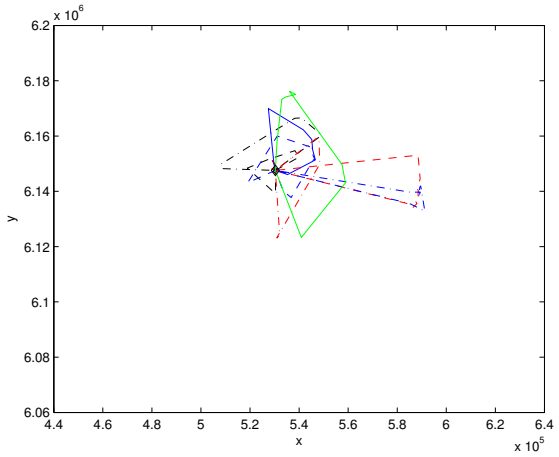
Data generation

The problem data provided by Transvision A/S and described in chapter 4 does not include any information about the master plans or variations in the customer demand. In the first two sections of this chapter the approach used to generate master plans and customer demand variations will be described. In the last section the data sets generated for the test purposes are briefly introduced. These data sets are used to evaluate the performance of the Lagrangian heuristic described in the previous chapter.

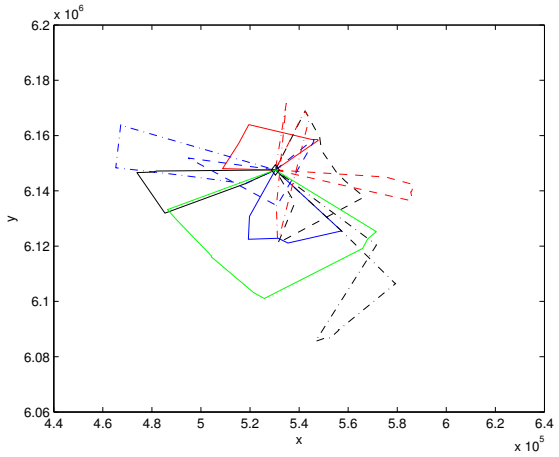
7.1 Master Plans

During the test of the improvement strategies described in section 5.3, a number of different solutions to the problem has been generated. The best found solution used 59 vehicles and had the total cost of 21245.2 minutes. The solution was produced using re-insertion moves in the embedded improvement and swap moves after the insertion procedure.

In the following the routes of the best solution found will be referred to as *master plans*. In figures 7.1-7.3 the master plans are displayed.

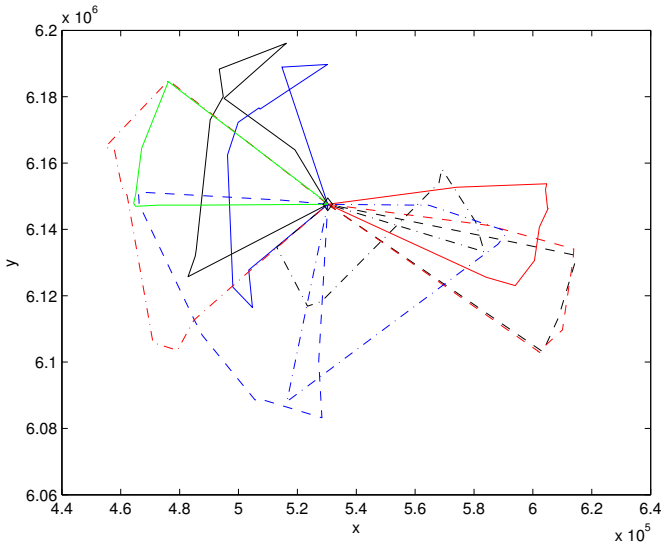


(a) Routes 1-10.

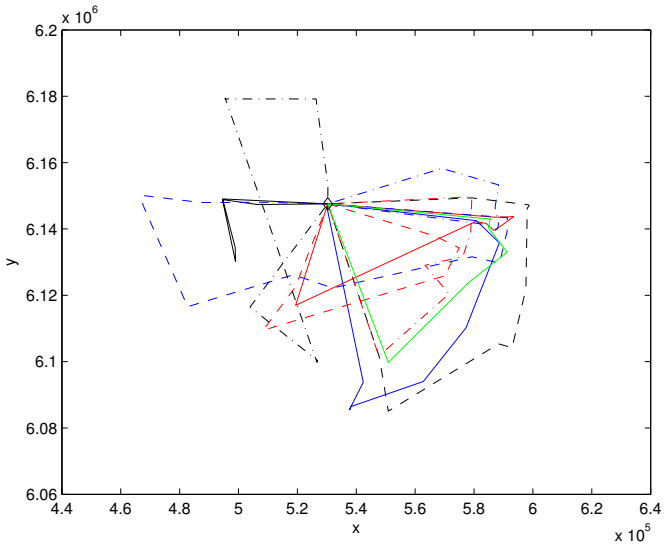


(b) Routes 11-20.

Figure 7.1: Master plans 1-20.

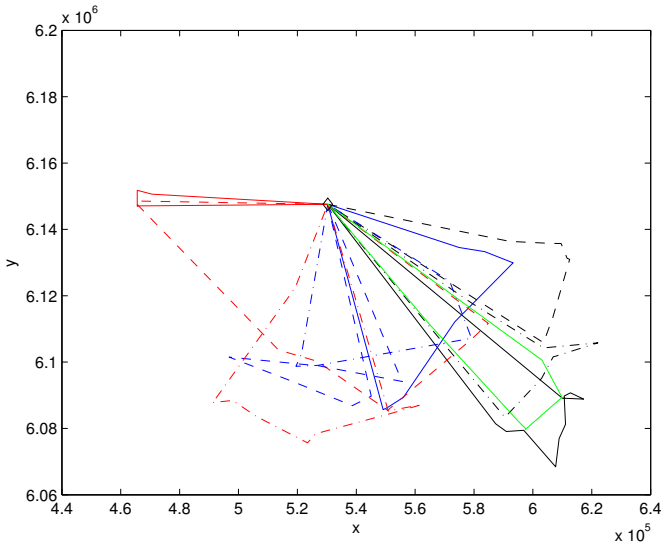


(a) Routes 21-30.

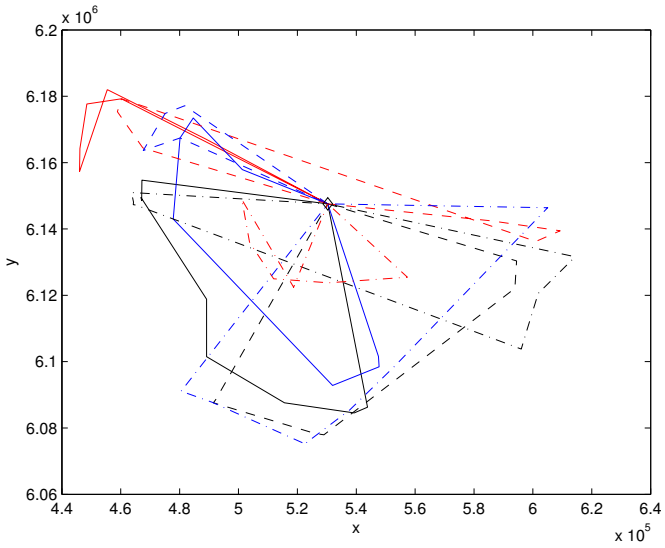


(b) Routes 31-40.

Figure 7.2: Master plans 21-40.



(a) Routes 41-50.



(b) Routes 51-59.

Figure 7.3: Master plans 41-59.

7.2 Demand Variation

The demand changes considered in this project are defined as cancellation of deliveries to some customers. Based on the available information about customer demand, a number of new data sets are generated.

The data sets are generated by randomly choosing a certain number of customers and setting their demand to 0, i.e. cancelling the deliveries to these customers. Several scenarios, i.e. variations in demand of different scale, are considered. The number of customers removed from the original data set in the different scenarios corresponds to 3, 5, 7, 10, 12, 15 and 20% of the total number of customers. For each scenario, 5 different data sets were generated.

In table 7.1 the demand variation is summarized as follows: For each scenario the number of the customers who cancelled their deliveries is shown. The last column of the table indicates the number of master plans affected by demand variation. For example, in the data sets generated for the demand variation of 3%, between 12 and 14 master plans will be affected by the cancelled deliveries.

Demand change %	**No. of cancelled deliveries**	**No. of affected routes**
3	15	12 – 14
5	25	19 – 22
7	35	25 – 28
10	50	32 – 40
12	60	38 – 41
15	75	41 – 46
20	100	48 – 51

Table 7.1: Demand variation overview.

New solutions are then constructed for each data set, and the quality of solutions both in terms of total cost and computational time is discussed. The presentation of the obtained results can be found in chapter 8.

7.3 Test Data Sets for the Lagrangian Heuristic

The problem described the data provided by Transvision A/S proved to be difficult to solve by the developed Lagrangian heuristic. The key issue was

constructing a feasible solution to the problem using the primal heuristic.

For test purposes a number of smaller data sets are constructed based on the original problem. The aim of the test is to find out whether the performance of the primal heuristic is dependent on the size of the problem.

The constructed data sets are divided into 3 classes:

- *micro* – 3 vehicles of each type and 60 customers
- *small* – 5 vehicles of each type and 125 customers
- *medium* – 10 vehicles of each type and 250 customers

The ratio between the number of customers and the number of vehicles in the original problem is maintained in the new data sets. For each class, 3 problem instances are generated by randomly selecting a number of customers from the original data set.

In appendix E the geographical distribution of the customers together with the information about the customers time windows is presented for each generated data set. In the rest of the report the problem instances of different types are denoted in the following way: M1–M3, S1–S3 and Md1–Md3 for the problems of type *micro*, *small* and *medium*, respectively.

Chapter 8

Computational Results

In this chapter the computational experiments conducted during the thesis are presented and discussed. The developed solution approach is programmed in C++, and all the experiments are performed on a Sun Fire 3800 with a 1200 MHz Sun Sparc processor.

8.1 Test of The Lagrangian Heuristic

The Lagrangian heuristic is tested on both the generated data sets described in section 7.3 and on the problem presented in chapter 4.

The following strategies are applied to each problem when executing the route generation phase:

a) Customers are seeded according to decreasing distance from the depot. The route generation phase is executed 3 times.

b) Customers are seeded according to decreasing distance from the depot. At the end of the route generation phase a list of unserviced customers is constructed for each vehicle type. Additional routes are generated for the customers in these lists and added to the pool of routes. It should be noted, that for each vehicle type the constructed routes only include the customers from the corresponding list, and not any other customers. The route generation phase is executed 3 times.

c) The procedure described by a) is performed twice. The second time the customers are seeded according to a different criterion, i.e. the next customer chosen for the insertion is the one with the earliest allowed starting time.

d) The procedure described by b) is performed twice. The second time the customers are seeded according to a criterion where the next customer chosen for the insertion is the one with the earliest allowed starting time.

e) The procedure described under c) is performed first. Afterwards the route generation phase is executed twice for each vehicle type and all the customers. The produced routes are added to the pool of routes.

f) This strategy is similar to the one described by e) except that the procedure c) is replaced by d).

In each execution of the route generation phase the following is performed: the insertion heuristic with embedded re-insertion moves, the post-insertion procedure if necessary and the improvement procedure with both re-insertion and swap moves. Based on the routes obtained from the insertion heuristic with embedded improvement, the distance data of the original problem is permutated as described in section 5.4. The number of pairs of customers P chosen from each route is set to 4. The larger values of P resulted in the generation of sets of routes with some customers left unserviced in the second or third execution of the route generation phase.

For each test problem six different SCPP instances were constructed. The SCPP instances differ by the number of columns in the constraint matrix which depends on the strategy applied to the route generation phase. It seems reasonable to assume that a large number of columns in the SCPP will increase the probability of finding a good solution to the corresponding routing problem. According to this assumption, strategies d), e) and f) which all generate larger number of routes compared to the other strategies are expected to perform best.

Table 8.1 presents a summary of the resulting SCPP instances for each problem type.

Problem type	Strategy RG	No. of rows	No. of columns
M1 – M3	a	63	41, 27, 54
M1 – M3	b	63	86, 68, 102
M1 – M3	c	63	89, 88, 116
M1 – M3	d	63	192, 186, 223
M1 – M3	e	63	120, 112, 139
M1 – M3	f	63	219, 204, 240
S1 – S3	a	128	65, 76, 128
S1 – S3	b	128	160, 175, 167
S1 – S3	c	128	164, 193, 180
S1 – S3	d	128	367, 398, 382
S1 – S3	e	128	223, 244, 240
S1 – S3	f	128	420, 446, 437
Md1 – Md3	a	253	159, 142, 120
Md1 – Md3	b	253	329, 328, 288
Md1 – Md3	c	253	401, 338, 318
Md1 – Md3	d	253	763, 725, 674
Md1 – Md3	e	253	513, 448, 417
Md1 – Md3	f	253	866, 826, 768

Table 8.1: Test problem details. RG stands for route generation phase.

Apart from the three stopping criteria described in section 6.2.3, the Lagrangian heuristic is terminated if no columns are added to the problem after the extra route generation procedure has been performed. This can happen if no routes can be generated for the remaining uncovered customers and the available vehicles, or if the generated routes exist in advance and therefore are not added to the problem. The iteration limit of 100 subgradient iterations is introduced as the last stopping criterion. In the following the results obtained by the Lagrangian heuristic for the different problem types are discussed.

8.1.1 Test results for the problem instances of type *micro*

In table 8.2 the results obtained by the Lagrangian heuristic for the problem instances of type *micro* are presented.

Because of the small size of the problems, it was possible to solve the generated SCPP instances to optimality using COIN. The values of the optimal solutions for both the initial SCPP (IP) and the final SCPP (FP) are displayed in column 2 and 3 of the table. The values of the best solutions found

	Solution value			Max LB Z_{max}	Extra cols. ExtC	No. of subgr. ite.	Time, sec.	Status
	Optimal for IP Z^*_{init}	Optimal for FP Z^*_f	Best found Z_{UB}					
M1.a	2867.14	2867.14	**2867.14**	2867.14	–	1	0.01	
M1.b	2867.14	2867.14	2909.25	2848.07	102	16	3.7	zc
M1.c	2867.14	2867.14	**2867.14**	2867.14	65	12	1.79	
M1.d	2867.14	2856.75	2870.16	2840.71	200	52	12.69	zc
M1.e	2867.14	2867.14	3018.07	2858.49	57	14	3.13	zc
M1.f	2867.14	2799.56	2953.53	2809.15	57	12	2.63	zc
M2.a	3036.3	3036.3	**3036.30**	3036.30	–	1	0.01	
M2.b	3036.3	3015.14	**3015.14**	3015.14	12	2	0.43	
M2.c	3036.30	3009.04	**3009.04**	3009.04	55	11	1.29	
M2.d	3036.30	3036.30	3704.79	2895.48	46	11	2.47	zc
M2.e	3036.30	2997.10	**2997.10**	2880.39	53	22	6.81	zc
M2.f	3036.30	2981.77	3065.47	28116.44	73	16	3.08	zc
M3.a	2873.94	2873.94	**2873.94**	2873.94	–	1	0.02	
M3.b	2873.94	2759.77	**2759.77**	2759.77	110	21	4.03	
M3.c	2873.94	2873.94	**2873.94**	2873.94	–	1	0.03	
M3.d	2873.94	2786.8	*2809.62*	2738.38	227	65	17.26	zc
M3.e	2873.94	2732.88	**2732.88**	2732.88	9	2	0.53	
M3.f	2873.94	2873.94	2944.33	2707.36	122	71	16.5	zc

Table 8.2: Results for the problems of type *micro*. The optimal solutions found by the Lagrangian heuristic are displayed in bold face. Italic style indicates that the obtained solution is better than the optimal solution to the IP, but not optimal for the FP. *zc* in the status column indicates that the Lagrangian heuristic was stopped due to zero added columns.

by the Lagrangian heuristic are shown in column 4 of the table. The number of columns added to the initial problem during the solution process can be seen in column 6. Information about the maximum lower bound, the number of subgradient iterations and the solution time is also included in the table.

Table 8.2 shows that the optimal solution is found for 10 out of 18 SCPP instances. In 5 cases the optimal solution found by the Lagrangian heuristic is better than the optimal solution to the the initial SCPP due to the generation of extra columns. Furthermore, in one case a solution found by the Lagrangian heuristic is better than the optimal solution to the initial problem, though it is not the optimal solution to the final problem, see the solution obtained for M3 by applying strategy d).

For problem M1 the best solution is found by applying strategy a) and c) to the route generation phase. If strategy a) is used the optimal solution is found already in the first iteration of the Lagrangian heuristic without generating

extra columns. When applying strategy c) 12 iterations are performed, and 65 columns are added to the problem before the optimal solution is found. Thus, the assumption about the expected performance of different strategies does not hold in this case.

For M2 the best solution value found is 1.29% better than the value of the optimal solution to the initial SCPP. The number of subgradient iterations and the number of added columns are 22 and 53, respectively. The solution is found by applying strategy e) to the route generation phase. Strategies b) and c) also produce the optimal solutions better than the optimal solution to the IP.

For problem M3 the best solution is found by applying strategy e), and the solution value differs from the optimal solution value to the initial problem by 4.91%. 2 subgradient iterations are performed and 9 new columns are added to the problem. A solution better than the optimal solution to the IP is also found by applying strategy b).

Based on the obtained results, it can be concluded that for the problem instances of type *micro* the Lagrangian heuristic is able to find the optimal solutions to the problems in reasonable time, i.e. within 20 seconds. It is difficult to select a single strategy which is the best for all three problems, though applying strategy e) gives the best results for problem M2 and M3. In the worst case, the solution found by the Lagrangian heuristic is 5%, 18% and 2.39% worse than the optimal solution for problem M1, M2 and M3, respectively.

8.1.2 Test results for the problem instances of type *small*

In table 8.3 the results obtained by the Lagrangian heuristic for the problem instances of type *small* are presented.

According to the table the Lagrangian heuristic found the optimal solution to 7 out of the 18 SCPP instances. It is quite obvious that a longer time is required to solve the problems of type *small* than of type *micro*, though with a longest solution time of just above 3 minutes.

For problem S1 the best solution is found by applying strategy f). The solution is obtained after 63 subgradient iterations, and 320 extra columns have been added to the problem in the process. The value of this solution is 9.34% better than the optimal solution to the initial problem. The worst

solution obtained for S1 lies within 10.03% of the value of the optimal solution to the problem.

For problem S2 the best solution is 7.99% better than the optimal solution to the IP and is obtained after 30 subradient iterations. The number of columns added to the problem is 281, and strategy d) is applied to the route generation phase. As it can be seen in the table, no solution has been found by applying strategy e). The Lagrangian heuristic has terminated because no columns were added to the problem in iteration 2, and no feasible solution was found in the first iteration.

For problem S3 the best solution is found by applying strategy f). The number of subradient iterations performed and the number of added columns are 26 and 147 respectively. The solution value is 3.42% better than the optimal solution to the initial problem. In the worst case the Lagrangian heuristic is not able to find a feasible solution to the problem in 100 iterations, and the algorithm is terminated due to the iteration limit, see the results when applying strategy d).

As for the problem instances of type *micro*, no singe strategy can be pointed out as the best performing one for the problem instances of type *small*. However, the best solutions to problems S1 and S3 are found by applying strategy f) to the route generation phase, which is consistent with the assumption about the best preforming strategies.

8.1.3 Test results for the problem instances of type *medium*

For the problem instances of type *medium* the limit on the number of subgradient iterations was set to 50 instead of 100. This was necessary due to the long computation times, e.g. performing 100 iterations for problem Md1.b took about 25 minutes. At the same time it was noticed that the best solution returned by the Lagrangian heuristic was found in the early iterations and was not improved since.

It should also be noted that due to the problem size, computing the optimal solution for the initial and the final problem by COIN was very time consuming and in some cases impossible. Thus, the objective values for the optimal solutions are not available for comparison with the solutions found by the Lagrangian heuristic. Instead, the value of the best feasible solution found during the route generation phase by the insertion procedure with embedded improvement is used as a reference value.

	Solution value			Max LB	Extra cols.	No. of subgr.	Time,	
	Optimal for IP Z^*_{init}	Optimal for FP Z^*_f	Best found Z_{UB}	Z_{max}	ExtC	ite.	sec.	Status
S1.a	5781.56	5781.56	5884.13	5679.66	297	45	24.99	zc
S1.b	5781.56	5135.15	**5135.15**	5135.15	9	1	0.39	
S1.c	5570.92	5570.92	5731.8	5427.45	671	100	81.36	il
S1.d	5570.92	5570.92	6192.49	5060.35	288	100	144.7	il
S1.e	5570.92	5267.13	*5337.98*	5214.90	42	4	1.57	zc
S1.f	5570.92	5034.13	**5034.13**	4927.53	320	63	135.6	zc
S2.a	5608.47	5608.47	5609.9	5608.47	261	53	35.53	zc
S2.b	5608.47	5515.36	**5515.36**	5432.2	90	13	6.83	zc
S2.c	5608.47	5581.80	**5581.80**	5569.66	272	25	44.66	zc
S2.d	5608.47	5160.66	**5160.66**	5160.66	281	30	58.68	
S2.e	5608.47	5608.47	–	5300.74	0	2	1.04	zc
S2.f	5608.47	5608.47	5610.86	5190.07	313	31	89.31	zc
S3.a	5771.00	5745.54	*5748.52*	5745.54	298	55	81.58	zc
S3.b	5771.00	5768.61	**5768.61**	5418.73	507	100	152.9	il
S3.c	5771.00	5771.00	6093.07	5771	204	37	70.89	zc
S3.d	5771.00	5771.00	–	5383.61	246	100	192.75	il
S3.e	5771.00	5696.42	*5767.87*	5386.47	641	100	192.57	il
S3.f	5771.00	5573.59	**5573.59**	5232.46	147	26	84.26	zc

Table 8.3: Results for the problems of type *small*. The optimal solutions found by the Lagrangian heuristic are displayed in bold face. Italic style indicates that the obtained solution is better than the optimal solution to the IP, but not optimal for the FP. *zc* in the status column indicates that the Lagrangian heuristic was stopped due to zero added columns. *il* in the status column indicates that the Lagrangian heuristic was stopped due to the iteration limit.

In table 8.4 the results obtained by the Lagrangian heuristic for the problem instances of type *medium* are presented.

As it can be seen from the table the performance of the Lagrangian heuristic can be described as poor for the problems of this type: For 9 problems instances of 18 the solution returned by the Lagrangian heuristic is worse than the best solution found in the route generation phase. It is possible that increasing the iteration limit can give better solutions. However, the time required to perform 50 iterations is already quite large. Thus, increasing the iteration limit will lead to longer computation times without guarantee for finding a better solution.

The optimal solution is only found in one case, i.e. for problem Md2 if strategy c) is applied in the route generation phase. The obtained solution is 1.08% better than the best feasible solution found duering the route generation phase.

	Solution value						
	Best feasible from RG Z^*_{ins}	Best found Z_{UB}	Max LB Z_{max}	Extra cols. ExtC	No. of subgr. ite.	Time, sec.	Status
Md1.a	10486.10	10604.00$^-$	10398.90	699	50	188.35	il
Md1.b	10486.10	10421.80	10216.60	1053	50	507.97	il
Md1.c	10486.10	11116.80$^-$	10398.90	1245	50	503.05	il
Md1.d	10486.10	10775.00$^-$	9729.78	1080	50	1299.76	il
Md1.e	10486.10	10270.30	9761.73	1153	50	866.71	il
Md1.f	10486.10	10460.20	9648.55	1044	50	1034.6	il
Md2.a	11112.30	10944.20	10644.10	717	50	120.28	il
Md2.b	11112.30	10967.20	10352.00	990	50	228.55	il
Md2.c	11112.30	**10992.20**	10992.20	51	1	1.85	
Md2.d	11112.30	11958.30$^-$	10064.10	990	50	899.1	il
Md2.e	11112.30	10773.30	10139.10	573	50	109.53	il
Md2.f	11112.30	12097.70$^-$	9728.31	1135	50	1230.79	il
Md3.a	10205.20	11039.00$^-$	10011.90	975	50	341.01	il
Md3.b	10205.20	10064.80	9487.87	573	50	479.12	il
Md3.c	10205.20	10531.40$^-$	10011.90	1222	50	595.94	il
Md3.d	10205.20	10855.50$^-$	9401.40	748	50	516.05	il
Md3.e	10205.20	10176.70	9589.71	101	2	5.83	zc
Md3.f	10205.20	10218.90$^-$	9172.95	768	50	1002.22	il

Table 8.4: Results for the problems of type *medium*. *RG* stands for route generation phase. 10604$^-$ means, that the solution value found by the Lagrangian heuristic is worse than the best feasible solution obtained from the insertion heuristic. *zc* in the status column indicates that the Lagrangian heuristic was stopped due to zero added columns. *il* in the status column indicates that the Lagrangian heuristic was stopped due to the iteration limit.

Though there is a general tendency to longer computation times for large problem instances, the performance of the Lagrangian heuristic is obviously problem specific: The optimal solution is found in 1.85 seconds for the problem instance with 253 rows and 338 columns (Md2.c), whereas almost 6 minutes are used to perform 50 iterations for the problem with 235 rows and 120 columns (Md3.a) without finding a good solution to the problem.

The long computation times can be explained by the following two factors: Firstly, the additional route generation was performed a significant number of times during the iteration process. For the problems of type *medium* the extra route generation was activated at least once in every subgradient iteration. Secondly, the routing problem defined by the available vehicles and the remaining unrouted customers after the primal heuristic is stopped is *hard to solve*. This is explained in more details in section 6.3.

8.1.4 Test results for the case study problem

In the following table the results obtained by the Lagrangian heuristic for the problem considered in this thesis will be presented.

Based on the performance of the Lagrangian heuristic on the problem instances of type *medium*, the iteration limit is also set to 50 for the problem considered in this thesis.

In table 8.5 an overview of the size of the SCPP instances obtained by use of different strategies in the route generation phase is given. The same 6 strategies are applied to the route generation phase as for the other test problems. Furthermore, some of the routes constructed based on the nearest neighbour principle described in section 6.2.2 are added to the problem. The added routes are those with lengths of at least 6 customers, 724 routes in all.

Strategy RG	No. of rows	No. of columns
a	503	965
b	503	1330
c	503	1377
d	503	2124
e	503	1586
f	503	2322

Table 8.5: Overview over SCPP instances for the problem considered in this thesis. *RG* stands for route generation phase.

The results obtained by the Lagrangian heuristic for the problem considered in this thesis (OP) are displayed in table 8.6. The status column is not included in the table, as in all cases the Lagrangian heuristic is terminated due to the iteration limit. The computation time is not reported in the table, but performing 50 iterations took at least one hour in all cases.

	Solution value				
	Best feasible from RG Z^*_{ins}	Best found Z_{UB}	Max LB Z_{max}	Extra cols. ExtC	No. of subgr. ite.
OP.a	21402.20	22592.90^-	20499.40	2238	50
OP.b	21402.20	20631.80	18928.80	2324	50
OP.c	21402.20	21217.50	19758.10	1974	50
OP.d	21402.20	20681.20	18440.70	2240	50
OP.e	21402.20	20027.70	18600.10	2212	50
OP.f	21402.20	20929.30	18122.90	2985	50

Table 8.6: Results for the case study problem. *RG* stands for route generation phase. 22592.90^- means that the solution value found by the Lagrangian heuristic is worse than the best feasible solution obtained from the insertion heuristic.

As it can be seen from the table, the best result is obtained by applying strategy e) to the route generation phase. The solution found by the Lagrangian heuristic is 6.4% better than the best solution found by the insertion heuristic. However, the computational effort needed to find that solution is quite high: 2212 columns are added to the initial problem, and the computation time is about 70 minutes. A more detailed analysis of the iteration process showed that, in spite of the extra column generation, the primal heuristic was unable to find a feasible solution in the first k iterations of the Lagrangian heuristic. For the different strategies the value of k varied from 4 and up to 15.

It is difficult to point out a single reason for the bad performance of the Lagrangian heuristic on the problems of large size. A more detalied discussion of the issues, which need to be investigated further, is given in chapter 9.

Based on the reported results, it can be concluded that the current implementation of the Lagrangian heuristic cannot always find acceptable solutions for real life problems in reasonable time.

8.2 Experiments with Master Plans

As mentioned in chapter 1 the current practice in the daily distribution of products is that master plans are executed without taking the changes in customer demand into account. It is reasonable to assume that adjusting the master plans on the day of operation according to the changes in customer demand would result in a better solution in terms of total costs. In the following the question of how beneficial such a revision of master plans can

be will be discussed.

The master plans for the problem considered in this thesis are described in section 7.1. According to the master plans solution, 59 vehicles are used to service 500 customers. The total cost of this solution is 21245.2 minutes.

In the following, different strategies for adjusting the master plans according to the demand changes are discussed. The easiest option will be to remove the customers which have cancelled their deliveries from the master plans. This strategy requires almost no computation time, and the master plans are changed by the least possible amount. However, it is reasonable to assume that the savings in the total cost obtained by this method will also be minimal.

There is a trade-off between the amount of changes in the master plans and the savings in the total cost of the solution. The main objective of the routing problem is to minimize the total cost. However, a significant change in the master plans is not desirable due to reasons such as driver satisfaction. Frequent changes in the master plans may not be welcomed by drivers who are used to execute the same routes every day. On the other hand, adjusting the master plans can also imply shorter working days for the drivers due to the improved routes.

After the customers with cancelled deliveries have been removed from the master plans, the total cost of the resulting routes can potentially be reduced even further by applying the improvement moves described in section 5.3. As both the re-insertion and the swap move are aimed at changing the routes, a third type of improvement move is introduced. The *within route re-insertion* move works as follows: Each customer of a route is considered for removal and insertion at another position in the same route. The move is accepted only if it leads to a reduction in the total cost. As this approach is designed to preserve the structure of the routes as much as possible, only the routes affected by the changes in customer demand are considered for improvement.

The last option to take the changes in demand into account is to construct a new set of routes from scratch every day based on the information about customer demand. As concluded in the previous section, the performance of the Lagrangian heuristic on the problem considered in this thesis was not satisfactory. Thus, only the first phase of the solution approach will be applied in the following to construct the new routes. Before performing the insertion heuristic, the customers are seeded according to decreasing distance from the depot. During the implementation and tests of the solution approach it was noticed that this seed criterion produces better results than seeding the customers according to the earliest allowed starting time. Due to the time

complexity of the improvement moves, the best strategy is to use the re-insertion moves in the embedded improvement. After the insertion heuristic has finished, the improvement procedure based on the swap moves is applied. The performance of the improvement moves described in section 5.3 has been evaluated, and the results can be found in appendix D. According to the results presented is appendix D, the embedded improvement proved to be beneficial in terms of solution quality.

Before presenting the results, a short summary of the suggested strategies for master plans adjustment is given:

1. simple removal of customers from their respective routes
2. simple removal followed by re-insertion moves
3. simple removal followed by swap moves
4. simple removal followed by within route re-insertion moves
5. construction of the routes from scratch

The first four strategies can be characterized as *modification strategies*, and the last one as a *reconstruction strategy*.

Different scenarios for the changes in customer demand are described in section 7.2.

In table 8.7 the results obtained by applying the modification strategies to adjust the master plans are presented for the case where the customer demand is changed by 3%. The demand change of 3% corresponds to the situation where 15 customers (of 500) have cancelled their deliveries.

	Total route cost							
Set No.	**Simple removal**	**Red. %**	**With re-ins**	**Red. %**	**With swap**	**Red. %**	**With re-insWR**	**Red. %**
1	20939.36	1.44	20517.72	3.42	20542.14	3.31	20927.93	1.49
2	21054.44	0.90	20662.59	2.74	20704.26	2.55	20993.51	1.18
3	21024.08	1.04	20614.59	2.97	20845.70	1.88	21035.98	0.98
4	21008.06	1.12	20596.11	3.06	20870.28	1.76	20996.88	1.17
5	20997.79	1.16	20606.67	3.01	20842.69	1.89	20973.51	1.28
	21004.74	1.13	20599.53	3.04	20761.01	2.28	20985.56	1.22

Table 8.7: New solutions obtained for a demand changes of 3%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

As it can be seen from the table, simple removal of customers from the routes leads to a reduction in total cost of only 1.13% on average. Applying the within route re-insertion moves yields an average reduction of the total cost of 1.22%. The best results are achieved by the re-insertion moves with the average savings of 3.04% in total cost. Similar results are obtained for the other scenarios in the demand change. The corresponding result tables can be seen in appendix F.

In figure 8.1 the average improvement rate of the solution value is shown for the four modification strategies and for each scenario in demand variation. As it can be seen from the figure, the best performing method for all scenarios is removing the customers with cancelled deliveries from their respective routes and then applying re-insertion moves to the resulting routes. The solutions obtained by the first four strategies in all scenarios include the same number of vehicles as in the master plans, i.e. 59 vehicles are used to serve the customers.

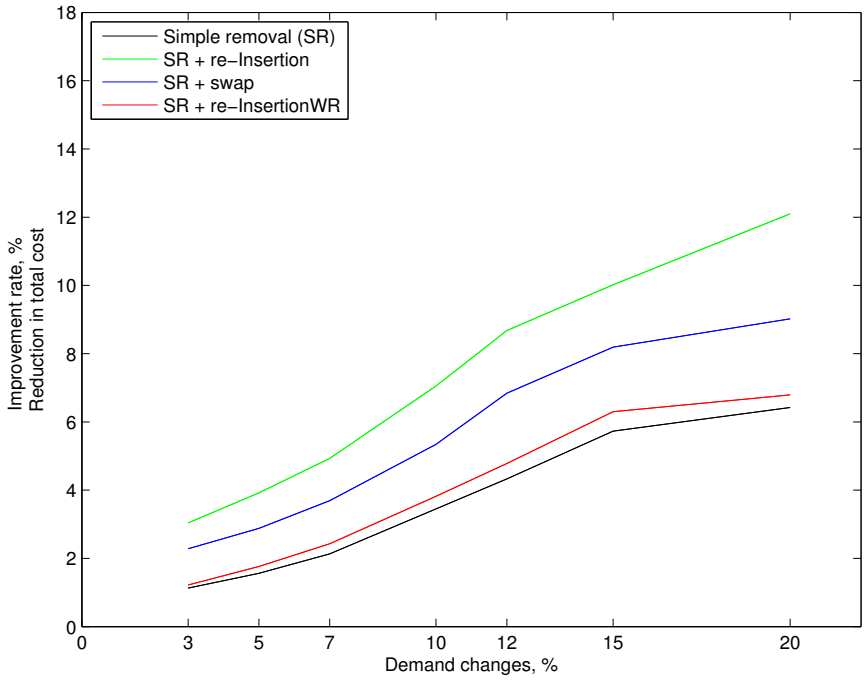


Figure 8.1: Solution improvement for the four different modification strategies.

In table 8.8 the computational effort needed to produce the solutions by applying the modification strategies is illustrated.

Demand change %	Time required by the modification strategies, sec.			
	SR	**SR+riWR**	**SR+rI**	**SR+S**
3	0.010	0.064	0.393	30.103
5	0.016	0.106	0.392	28.644
7	0.022	0.128	0.387	27.697
10	0.028	0.170	0.394	26.212
12	0.030	0.178	0.398	24.932
15	0.036	0.192	0.379	22.829
20	0.040	0.200	0.377	21.361

Table 8.8: The average computation time for the modification strategies under different demand variation scenarios. SR stands for simple removal, $SR + riWR$: simple removal followed by within route re-insertion, $SR + rI$: simple removal followed by re-insertion and $SR + S$: simple removal followed by swap.

As it can be seen from the table, the computation time needed to perform the simple removal is almost negligible. Obviously, the larger change in demand, the more time it takes to perform the first two modification procedures. This is due to the fact that their computation time depends directly on the number of cancelled deliveries and the number of affected routes. The computation time needed when applying the third strategy is almost the same for the different demand variation scenarios. For the last procedure based on simple removal followed by swap improvement, the computation time is decreasing as the percentage of demand changes increases. This can be explained by the fact that when more customers are removed from the routes, less time is needed to perform the swap procedure.

Combining the results displayed in figure 8.1 and table 8.8, the $SR+rI$ strategy can be declared to be the best performing one among the modification strategies. It gives the largest reduction in the total cost of the solution, and it requires at most 0.4 seconds to perform.

In table 8.9 the results obtained by the reconstruction strategy are presented for a demand change of 3%. The result tables for the rest of demand variation scenarios can be found in appendix F.

The table shows that a reduction in the total cost is 2.38% on average, though a slightly worse solution is produced in a single case. In 4 of 5 cases the number of vehicles used to serve the customers is smaller compared to the master plan solution.

The average computation time needed when applying the reconstruction procedure, the average reduction in the total cost and the average number of vehicles used in the solution are presented in table 8.10.

Set No.	No. of vehicles	Total Cost	Red. %	Time sec.
1	57	21358.38	-0.53	31.65
2	59	20619.94	2.94	32.69
3	58	20026.21	5.74	32.02
4	57	20864.54	1.79	31.68
5	58	20823.92	1.98	31.37
	57	20738.60	2.38	31.88

Table 8.9: New solutions obtained by the reconstruction strategy for a demand change of 3%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Demand change %	Time sec.	Red. %	No. of veh
3	31.88	2.38	57
5	31.29	5.33	56
7	29.47	5.83	55
10	27.92	8.93	54
12	26.34	10.61	52
15	24.76	12.52	51
20	21.44	17.70	48

Table 8.10: The average computation time for the reconstruction strategy under the different demand variation scenarios.

The best modification strategy is compared to the reconstruction strategy in figure 8.2.

The figure shows that for small demand variations, i.e. less than 3%, the modification strategy performs best. This seems reasonable, as it makes little sense to build a new set of routes if only 5-15 customers of 500 have cancelled their deliveries. For the changes in demand bigger than 3%, a larger reduction in the total cost can be obtained by applying the reconstruction strategy.

The decision as to which strategy should be used in real life is a question of priorities. Constructing the routes from scratch takes longer time than just modifying the existing master plans. However, the average computation time for the reconstruction strategy is only about 30 seconds for the demand variation of 3% and it is even less for the larger variations, see table 8.10. Thus, if the main objective is to minimize the total cost and the change in customer demand is greater than 3%, the reconstruction strategy should be chosen.

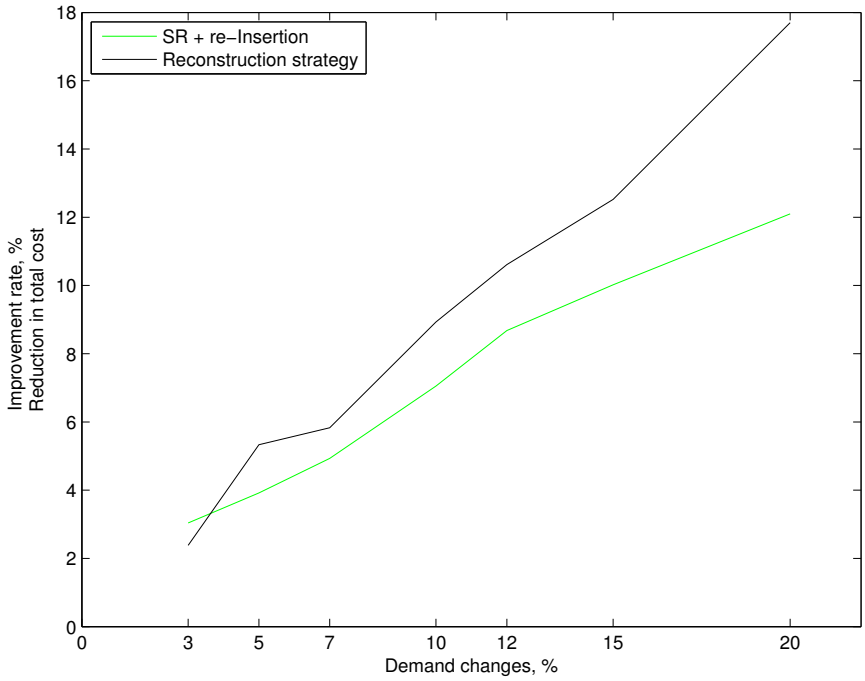


Figure 8.2: Solution improvement – modification versus reconstruction.

The solutions produced by the reconstruction procedure tend to involve fewer vehicles than suggested by the master plans. Whether these solutions can be accepted depends on several factors such as the lease agreements for the vehicles and the contracts with the drivers. In the case where it is desirable to use all the available vehicles, and if the time issue is of high importance, the modification strategy based on re-insertion improvement is to be preferred.

Chapter 9

Discussion and Future Work

In this chapter the solution approach which has been developed in this thesis is discussed based on the results presented in the previous chapter. Furthermore, some areas for future work are suggested.

9.1 Performance of the Lagrangian Heuristic

Before discussing the performance of the developed solution method, a short summary of the problem considered in this thesis is given. As mentioned in chapter 1, the daily distribution of products of the company, which has been considered in this thesis, is based on the concept of master plans. The master plans are executed without taking into account that changes in customer demand will occur, and they are revised only once or twice per year.

Based on the results presented in section 8.2, it can be concluded that revising the master plans on a daily basis can be beneficial for the company, especially in case of large variations in demand. Several strategies for adjusting master plans have been proposed in section 8.2. Depending on the magnitude of the demand variation, different strategies can be chosen. For changes in demand larger than 3%, the reconstruction strategy provides the best results, i.e. the largest reduction in the total cost suggested by the master plans.

Obviously, applying the reconstruction strategy requires development of a solution method that provides good solutions to the routing problem within a reasonable time.

The solution method developed in this thesis is a two-phase approach, where a large set of good quality routes are constructed in the first phase, and a Lagrangian heuristic is used to solve the resulting SCPP in the second phase.

As mentioned in chapter 8 the performance of the Lagrangian heuristic on the problem considered in this thesis is not satisfying. One of the issues for the current implementation of the Lagrangian heuristic is long computation times. As the problem considered in this thesis is a real-life problem, the time issue is important. If all the changes in demand are known prior to the day of operation, a longer time can be spent on searching for a solution that is close to being the optimal solution. However, if the master plans are adjusted just before the final route plans are given to the drivers, the solution has to be produced in a short time.

Furthermore, the performance of the Lagrangian heuristic is not stable is terms of solution quality. This means that in some cases the Lagrangian heuristic produces solutions worse than the best feasible solution obtained from the route generation phase. Based on these two issues, it can be concluded that the current implementation of the Lagrangian heuristic is not suitable for use in real life.

A key issue of the Lagrangian heuristic is the step where a feasible solution to the problem is constructed, i.e. the primal heuristic. The following attempts have been made to improve the performance of the primal heuristic: different selection rules for the next solution column, controlling the overlap, and generating columns 'on the fly'. These attempts seem to give results for the problems of type *micro* and *small*. However, no plausible explanation has been found of the fact that the problem instances of type *medium* and the present case study problem are so hard to solve for the primal heuristic. Several problematic areas for further research on this subject are outlined in the following section.

9.2 Future Research

For the route generation phase a further investigation of the route updating procedure is needed for cases where a customer is removed. As mentioned in chapter 5, the time complexity of the removal and subsequent updating procedure is $O(n^2)$ in the current implementation. For the traditional VRPTW, updating of a route after removal of a customer can be performed in linear time. However, the shift time limit constraints make the corresponding updating procedure for the VRPTWSTL more complicated. This is due to the interaction between the values of the earliest departure time from the depot and the earliest completion time of the route, and the latest departure time from the depot and the latest completion time of the route. In some situations the removal of a customer will imply that the route can be completed

earlier. Due to the shift time limit, the earliest departure time from the depot can also be changed. In this case, the e-values of the other customers on the route have to be re-evaluated, eventually leading to a change of the earliest completion time of the route again. The aim of the investigation should be to establish the rules for the updating procedure and to find out whether it can be performed in linear time. Reducing the time complexity of the updating procedure after removal of a customer will have a positive impact on the time complexity of the post-insertion and the improvement procedures.

To improve the overall performance of the Lagrangian heuristic a thorough investigation of the following areas is needed:

- *Route similarity*: The computational experiments showed that extra routes were generated at least once in almost every iteration of the Lagrangian heuristic. For the large problem instances, the route generation procedure was activated 2 or 3 times in a single iteration. This indicates that the columns of the matrix are still too similar. Therefore, at some point no more columns with a zero overlap can be found, and more routes have to be generated. Thus, some attention should be given to the route generation phase with focus on developing methods for producing distinctive routes.

- *Column selection*: During the experiments it was noticed that columns with a small number of covered rows were included in the solution. This would not be an issue if the number of available vehicles was unlimited. However, in the presence of the set packing constraints, including such columns in the solution corresponds to an unnecessary use of vehicle capacity. One way to solve this problem is to exclude the columns covering less than 2 or 3 rows from the constraint matrix. On the other hand, it seems reasonable to assume that there exist good feasible solutions with columns covering a small number of rows. Such solutions have been produced during the route generation phase. Thus, excluding these columns from the constraint matrix would lead to a reduction of the search space for the heuristic and potentially to a less optimal solution. Instead, further investigation of the column selection rules and the cost structure of the problem is needed to find out why such columns become more 'attractive' than the columns with a large number of covered rows.

- *Problem reduction*: In the current implementation, only one type of problem reduction method is performed, i.e. fixing some variables to

zero. This procedure means that some columns can be deleted from the problem, as they can never be present in the optimal solution. Another problem reduction method is to fix some variables to 1. This would enable the algorithm to delete not only a single column from the problem, but also the rows covered by that column. This problem reduction can potentially improve the performance of the Lagrangian heuristic in terms of computation time.

Finally, in this thesis it has been assumed that the only changes in demand are the cancellations of deliveries to some customers. Another type of demand variation can also be considered in the future, namely variations in the volumes to be delivered to each individual customer.

Chapter 10

Conclusion

The aim of this project is to design an efficient solution method for solving the Vehicle Routing Problem with Time Windows and Shift Time Limits (VRPTWSTL).

The implemented solution approach consists of two phases. In the first phase an insertion heuristic with embedded improvement is used to construct the initial set of routes. To handle the unrouted customers a post-insertion procedure has been developed based on the ejection chain approach. The improvement procedure is applied to the routes at the end of the route generation phase. The aim of the first phase is to produce a large number of good quality routes, which is accomplished by executing it several times. In order to produce distinctive routes, the problem data is permutated after each execution run of the route generation phase.

In the second phase a mixed Set Covering/Packing Problem instance (SCPP) is constructed based on the generated routes. The packing constraints are necessary in this case, as there is limited number of vehicles in the problem. Furthermore, the available vehicles are of different types, so packing constraint has to be introduced for each vehicle type. The resulting SCPP problem is then solved by the Lagrangian heuristic.

During the implementation of the solution approach the greatest challenge in the first phase has been incorporating the shift time limit constraints into the insertion heuristic. This issue has been successfully solved. In the second phase of the solution approach the key issue has been to find a feasible solution to the SCPP. The primal heuristic developed for this purpose failed in spite of producing a large number of routes in the route generation phase. Thus, the initial implementation of the Lagrangian heuristic has been extended to be able to produce additional routes 'on the fly'.

The final version of the solution approach has been tested on a number of problems of different sizes. Based on the results of the test the conclusion is that the Lagrangian heuristic is able to produce optimal solutions to the problems of small sizes. However, as the problem size increases, the performance of the Lagrangian heuristic becomes worse both in terms of solution quality and computation time.

Due to the long computation time, the current implementation of the Lagrangian heuristic cannot be applied in real life. Thus, only the first phase of the solution approach has been used to construct the solutions to the VRPTWSTL considered in this thesis.

The secondary objective of this thesis is to answer a question faced by companies whose daily distribution is based on the concept of master plans. The master plans do not take the demand changes into account. Therefore, it would be reasonable to assume that a reduction in total cost of the routes can be achieved by adjusting the master plans according to changes in the customer demand. The question is how large the variation in demand should be before such a revision is beneficial for the company.

Based on the experimental results it can be concluded that for changes in demand less than 3%, it can hardly be beneficial to reconstruct the routes. Instead, the master plans are modified in the following way: The customers with cancelled deliveries are removed from the routes and the re-insertion improvement procedure is applied to the resulting set of routes. For the case study problem considered in this thesis, such a revision resulted in a cost reduction of 3.04% on average compared to the master plan solution. This may not seem as a large saving on a single day, but in the long run substantial savings in distribution costs are accumulated. The main advantage of this modification method is an almost negligible computation time. For the changes in demand greater 5% the largest savings are achieved by building a new set of routes based on the updated information about customer demand. For the changes in demand of 5 to 15% the reduction in total cost lies within 5.33% – 12.52%. The computation time needed to produce the new solutions is about 30 seconds for the problem solved in this thesis. Obviously, the reconstruction procedure takes longer time than a simple modification, but the required time is still reasonable for real life settings.

Finally, as only the first phase of the solution approach has been used in the reconstruction procedure, it can be assumed that even large savings can be obtained by applying the whole approach. However, this requires further investigation and extensions of the current implementation of the Lagrangian heuristic.

References

[1] P. Badeau, F. Guertin, M. Gendreau, J.-Y. Potvin and E.D. Taillard: *A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows.* Transporation Research Part C: Emerging Technologies, 1997, Vol. 5, No. 2, pp. 109-122.

[2] J. E. Beasley: *An Algorithm for Set-Covering Problems.* European Journal of Operational Research, 1987, Vol. 31, No. 1, pp. 85-93.

[3] J. E. Beasley: *A Lagrangian Heuristic for Set-Covering Problems.* Naval Reserach Logistics, 1990, Vol. 37, No. 1.

[4] J.E. Beasley: *Chapter 6: Lagrangian Relaxation.*

[5] O. Bräysy, G. Hasle, W. Dullaert: *A Multi-start Local Search Algorithm for the Vehicle Routing Problem with Time Windows.* European Journal of Operational Research, 2004, Vol. 159, No. 3, pp. 586-605.

[6] O. Bräysy and M. Gendreau: *Vehicle Routing Problem with Time Windows, Part I: Route Construction and Local Search Algorithms.* Transportation Science, 2005, Vol. 39, No. 1, pp. 104-118.

[7] O. Bräysy and M. Gendreau: *Vehicle Routing Problem with Time Windows, Part II: Metaheuristics.* Transportation Science, 2005, Vol. 39, No. 1, pp. 119-139.

[8] O. Bräysy: *Fast Local Searches for the Vehicle Routing Problem with Time Windows* INFOR Journal, 2003, Vo. 40, No. 4.

[9] A.M. Campbell and M. Savelsbergh: *Efficient Insertion Heuristics for Vehicle Routing and Sceduling Problems.* Transportation Science, 2004, vol. 38, no. 3, s. 369-378.

[10] A. Carpara M. Fischetti and P. Toth: *A Heuristic Method for Set Covering Problem.* Operations Research, 1999, vol. 47., No. 5, pp.730-743.

[11] S. Ceria, P. Nobili and A.SassNO: *A Lagrangian-based Heuristic for Large-scale Set Covering Problems.* Mathematical Programming, 1998, Vol. 81, pp. 215-228.

[12] W.C. Chiang and R.A. Russel: *A Reactive Tabu Search Metaheuristic for the Vehicle Routing Problem with Time Windows.* INFORMS Journal on Computing, 1997, Vol. 9, No. 4.

[13] Z.J. Czech and P. Czarnas: *Parallel Simulated Annealing for the Vehicle Routing Problem with Time Windows* Parallel, Distributed and Network-based Processing, 2002, pp. 376-383.

[14] M. Desrochers, J. Desrochers and M. Solomon: *A New Optimization Algtorithm for the Vehicle Routing Problem with Time Windows.* Operations Research, 1992, Vol. 40, No. 2.

[15] M.L. Fisher, K.O. Jornsten and O.B.G. Madsen : *Vehicle Routing with Time Windows: Two Optimization Algorithms.* Operations Research, 1997, vol. 45., No. 3.

[16] M.L. Fisher and P. Kedia: *Optimal Solution of Set Covering/Partitioning Problems Using Dual Heuristics.* Management Science, 1990, Vol. 36, No. 6.

[17] C. Foisy and J.-Y. Potvin: *Implementing an insertion heuristic for vehicle routing on parallel hardware* Computers and Operations Research, 1993, Vol. 20., No. 7.

[18] L.M. Gambardella, E.D. Taillard ans G. Agazzi: *MACS-VRPTW: A Multiple Colony Ant System for Vehicle Routing Problems with Time Window Constraints* New Ideas in Optimization. McGraw-Hill, London, pp. 63-76.

[19] B.-L. Garcia, J.-Y. Potvin, J.M. Rousseau: *A Parallel Implementation of the Tabu Search Heuristic for Vehicle Routing Problems with Time Window Constraints* Computers and Operations Research, 1994, Vol. 21, No. 9.

[20] S. Haddadi: *Simple Lagrangian Heuristic for the Set Covering Problem* European Journal of Operational Research, 1997, Vol. 97, pp. 200-204.

[21] A. Hamacher and C.Moll *A new heuristic for vehicle routing with narrow time windows* Operations Research Proceedings, 1996, pp. 301-310.

[22] G.Ioannou, M. Kritikos, G. Prastacos: *A Greedy Look-Ahead Heuristic for the Vehicle Routing Problem with Time Windows* The Journal of the Operational Research Society, 2001, Vol.52, No. 5.

[23] B. Kallehauge, J.Larsen and O.B.G. Maden: *Lagrangian Duality Applied to the Vehicle Routing Problem with Time Windows.* Computers and Operations Research, 2006, Vol. 33, No. 5.

[24] J.P. Kelly and J. Xu: *A Set-Partitioning Heuristic for the Vehicle Routing Problem.* INFORMS Journal on Computing, 1999, Vol. 11, No. 2.

[25] G. Kontoravdis and J. Bard: *A GRASpfor the Vehicle Routing Problem with Time Windows.* ORSA Journal on Computing, 1995, Vol. 7, No. 1.

[26] G. Laporte, M. Gendreau, J.-Y. Potvin and F. Semet: *Classical and Modern Heuristics for the Vehicle Routing Problem.* Operational Research, 2000, vol. 7, s. 285-300.

[27] A. Olsen and A.N. Nielsen: *COIN: A simple guide for getting started with the OsiSolverInterface.* IMM, DTU, September 3, 2005.

[28] J.-Y. Potvin and J.-M. Rousseau: *A Parallel Route Building Algorithm for the Vehicle Routing and Scheduling Problem with Time Windows.* European Journal of Operational Research, 1993, No. 66, pp. 331-340.

[29] J.-Y. Potvin and J.-M. Rousseau: *An Exchange Heuristic for Routing Problems with Time Windows.* The Journal of the Operational Research Society, 1995, Vol.46, No. 12.

[30] Y. Rochat and E.D. Taillard: *Probabilistic Diversification and Intensification in Local Serach for Vehicle Routing.* Journal of Heuristics 1, 1995, pp. 147-167.

[31] R.A. Russel: *Hybrid Heuristics for the Vehicle Routing Problem with Time Windows.* Transportation Science, 1995, Vol. 29, No. 2.

[32] G. Schrimpf, J. Schneider, H. Stamm-Wilbrandt and G. Dueck: *Record Breaking Optimization Results Using the Ruin and Recreate Principle.* Journal of Computational Physics, 2000, Vol. 159, pp. 139-171.

[33] P. Shaw: *Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems.* Principles and Practice of Constraint Programming – CP98, Lecture Notes in Computer Science. Springer-Verlag, New York, pp. 417-433.

[34] M. Solomon: *Algorithms for the Vehicle Routing and Scheduling Problems with Time Windows Constraints.* Operations Research, 1987, vol. 35., No. 2.

[35] E.D. Taillard, P. Badeau, M. Gendreau, F. Guertin and J.-Y. Potvin: *A Tabu Search Heuristic for the Vehicle Routing Problem with Soft Time Windows.* Transporation Science, 1997, Vol. 31, No. 2.

[36] K.C. Tan, L.H. Li, Q.L. Zhu, K. Ou: *Heuristic Methods for the Vehicle Routing Problem with Time Windows.* Artificial Intelligence in Engineering, 2001, Vol. 15, pp. 281-285.

[37] S.R. Thangiah, K.E. Nygård and P.L. Juell: *GIDEON: A Genetic Algorithm System for Vehicle Routing with Time Windows.* Artificial Intelligence for Applications, 1991, Vol.i, 322-328.

[38] P. Toth and D. Vigo: *The Vehicle Routing Problem* SIAM Society for Industrial ans Applied mathematics, Philadelphia, 2002, p. 157.

Appendix A

Proof of Statement 4.3

Statement 4.3. *Assume that insertion of customer j causes the latest departure time to decrease, and that the latest completion time is also changed due to the shift time limit, $l_{n+1} = l_0 + ST$. Assume that l_{i+1} is changed in updating process. The value of l_j will remain unchanged, as otherwise the insertion would have been infeasible with respect to shift time limit.*

Proof of the statement 5.3. Assume that insertion of customer j between customers i and $i+1$ is feasible. Then one of the conditions that have to hold is

$$a_0 \leq ST \tag{A.1}$$

After insertion the new value of l_0 can be computed based on the value of l_j:

$$l_0 = l_j - t_{ji} - (a_0 - a_i) \tag{A.2}$$

and l_{n+1} is re-calculated to:

$$l_{n+1} = l_0 + ST \tag{A.3}$$

Due to the updating procedure l-value of customer $i+1$ is changed to

$$l_{i+1}^{new} = l_{n+1} - a_{i+1} \tag{A.4}$$

The only way for l_j to change is if condition (A.5) holds.

$$e_{i+1}^{new} - t_{ji+1} < l_j \qquad \Leftrightarrow \tag{A.5}$$
$$l_{n+1} - a_{i+1} - t_{ji+1} < l_j \qquad \Leftrightarrow \tag{A.6}$$
$$l_{n+1} - a_{i+1} - t_{ji+1} < l_0 + t_{ji} + (a_0 - a_i) \tag{A.7}$$

By substitution of l_{n+1} in (A.7) with the right hand side of equation (A.3), the following is deduced:

$$l_0 + ST - a_{i+1} - t_{ji+1} < l_0 + t_{ji} + (a_0 - a_i) \quad \Leftrightarrow \quad \text{(A.8)}$$

$$ST + (a_i - a_{i+1}) - t_{ji+1} - t_{ji} < a_0 \quad \Leftrightarrow \quad \text{(A.9)}$$

$$ST + t_{ij} + t_{ji+1} - t_{ji+1} - t_{ji} < a_0 \quad \Leftrightarrow \quad \text{(A.10)}$$

$$ST < a_0 \quad \text{(A.11)}$$

The last equation contradicts the assumption that insertion of customer j was feasible – see equation (A.1).

□

Appendix B

Insertion Procedure – Small Example

In the following the insertion algorithm described in chapter 5 is applied to the small problem instance with 8 customers and 3 vehicles. The problem is displayed graphically in figure B.1 below. The distances between the

Customers

i	s_i	q_i
1	15	40
2	20	50
3	30	70
4	30	70
5	30	60
6	15	30
7	20	50
8	20	40

Vehicles

VT	Q_k	A_k	B_k	ST_k
1	150	600	900	180
2	200	420	780	240
3	150	600	960	300

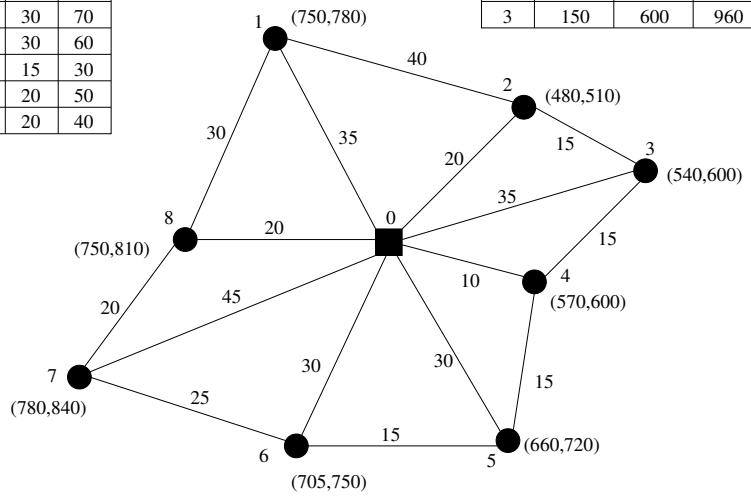


Figure B.1: Small example of the insertion procedure.

customers and the depot are shown in table B.1.

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
0	-	35	20	35	10	30	30	45	20
1	35	-	40	60	50	80	60	60	30
2	20	40	-	15	25	40	45	90	45
3	35	60	15	-	15	30	40	80	60
4	10	50	25	15	-	15	20	30	35
5	30	80	40	30	15	-	15	40	35
6	30	60	45	40	20	15	-	25	25
7	45	60	90	80	30	40	25	-	20
8	20	30	45	60	35	35	25	20	-

Table B.1: Small example – Distance table.

Before starting the insertion procedure, customers are seeded according to the two criteria mentioned in chapter 5:

- decreasing distance from the depot: {7, 1, 3, 5, 6, 2, 8, 4 }
- increasing earliest allowed starting time: {2, 3, 4, 5, 6, 1, 8, 7}

Three routes, one for each vehicle, are initialized as follows:
$r1$: [(600,900),0] – (600,900)
$r2$: [(420,420),0] – (420,780)
$r3$: [(600,960),0] – (600,960)
Customers are inserted then one at a time in the specified seeding order. In the following the first seeding criterion is applied.

Insertion procedure with customers seeded according to decreasing distance from the depot

Iteration 1 – Insert Customer 7

Insertion of customer 7 in route 2 is infeasible due to the customer's time windows, i.e. $e_j > l_j$:

$e_7 = max(780, 420 + 45) = \mathbf{780}$
$l_7 = min(840, 780 - 45 - 20) = \mathbf{715}$

For routes 1 and 3 insertion is feasible – see the table below:

	r1	**r3**
$\mathbf{e_j}$	max(780,600+45)=**780**	max(780,600+45)=**780**
$\mathbf{l_j}$	min(840,900-45-20)=**835**	min(840,960-45-20)=**840**
$\mathbf{e_0}$	max(600,845-180)=**665**	max(600,845-300)=**600**
$\mathbf{l_0}$	min(900,835-45)=**790**	min(960,835-45)=**790**
$\mathbf{e_{n+1}}$	max(600,780+20+45)=**845**	max(600,780+20+45)=**845**
$\mathbf{l_{n+1}}$	min(900,790+180)=**870**	min(960,790+300)=**960**
$\mathbf{a_0}$	45+20+45=**110**	45+20+45=**110**
$\mathbf{D_r}$	**50**	**50**
$\mathbf{c_r}$	45+20+45=**110**	45+20+45=**110**

Customer 7 is inserted in route $r1$ in position 1:

$r1$: [(665,790),110] – 7[(780,835),65] – (845,900)
$r2$: [(420,780),0] – (420,780)
$r3$: [(600,960),0] – (600,960)

Iteration 2 – Insert Customer 1

For route 1 insertion in only feasible in position 1, as insertion in position 2 will fail on condition $e_j \leq l_j$.

r1	**position 1**	**position 2**
$\mathbf{e_j}$	max(750,665+35)=**750**	max(750,780+20+60)=**860**
$\mathbf{l_j}$	min(780,835-15-60)=**760**	min(780,870-15-35)=**820**
$\mathbf{e_0}$	max(665,890-180)=**790**	-
$\mathbf{l_0}$	min(790,760-35)=**725**	-
$\mathbf{e_{n+1}}$	max(845,750+15+60+65)=**890**	-
$\mathbf{l_{n+1}}$	min(900,725+180)=**900**	-
$\mathbf{a_0}$	110+(15+35+60-45)=**175**	-
$\mathbf{D_r}$	**90**	-
$\mathbf{c_r}$	35+60-45=**50**	-

Insertion into route 2 is also infeasible with respect to customer time windows, whereas insertion into route 3 is feasible but more expensive compared to route 1.

	r2	**r3**
$\mathbf{e_j}$	max(750,420+35)=**750**	max(750,600+35)=**750**
$\mathbf{l_j}$	min(780,780-35)=**745**	min(780,960-35)=**780**
$\mathbf{e_0}$	-	max(600,800-300)=600
$\mathbf{l_0}$	-	min(960,780-35)=**745**
$\mathbf{e_{n+1}}$	-	max(600,750+15+35)=**800**
$\mathbf{l_{n+1}}$	-	min(960,745+300)=**960**
$\mathbf{a_0}$	-	35+15+35 = **85**
$\mathbf{D_r}$	-	**40**
$\mathbf{c_r}$	-	35+35=**70**

Customer 1 is inserted in route $r1$ in position 1, and the obtained routes are

shown below. Note, that the earliest delivery time for the customer 7 next in the route is updated due to the insertion , $e_7 = max(780, 750{+}15{+}60) = 825$.

$r1$: [(710,725),175] – 1[(750,760),140] – 7[(825,835),65] – (890,900)
$r2$: [(420,780),0] – (420,780)
$r3$: [(600,960),0] – (600,960)

Iteration 3 – Insert customer 3

Insertion of customer 3 is infeasible with respect to the customer time window for both route 1 and 3: $e_0^{r1} = 710 > L_3 = 600$ and $e_0^{r3} \geq L_3$. For route 2 the insertion is feasible:

	r2
$\mathbf{e_j}$ $\mathbf{l_j}$	max(540,420+35)=**540** min(600,780-30-35)=**600**
$\mathbf{e_0}$ $\mathbf{l_0}$	max(420,605-240)=**790** min(780,600-35)=**565**
$\mathbf{e_{n+1}}$ $\mathbf{l_{n+1}}$	max(420,540+30+35)=**605** min(780,600+240)=**780**
$\mathbf{a_0}$	35+30+35=**95**
$\mathbf{D_r}$	**70**
$\mathbf{c_r}$	35+35=**70**

The resulting routes are:

$r1$: [(710,725),175] – 1[(750,760),140] – 7[(825,835),65] – (890,900)
$r2$: [(420,565),100] – 3[(540,600),65] – (605,780)
$r3$: [(600,960),0] – (600,960)

Iteration 4 – Insert customer 5

According to the tables below, customer 5 can feasibly be inserted into routes 3 and 2.

	r1	**r3**
$\mathbf{e_j}$ $\mathbf{l_j}$	max(660,710+30)=**740** min(720,760-30-80)=**650**	max(660,600+30)=**660** min(720,960-30-30)=**720**
$\mathbf{e_0}$ $\mathbf{l_0}$	- -	max(600,720-300)=**600** min(960,720-30)=**690**
$\mathbf{e_{n+1}}$ $\mathbf{l_{n+1}}$	- -	max(600,660+30+30)=**720** min(960,690+300)=**960**
$\mathbf{a_0}$	-	30+30+30=**90**
$\mathbf{D_r}$	-	**70**
$\mathbf{c_r}$	-	30+30=**60**

r2	**position 1**	**position 2**
$\mathbf{e_j}$	max(660,420+30)=**660**	max(660,540+30+30)=**660**
$\mathbf{l_j}$	min(720,600-30-30)=**540**	min(720,780-30-30)=**720**
$\mathbf{e_0}$	-	max(420,720-240)=**480**
$\mathbf{l_0}$	-	min(565,720-30-30-35)=**565**
$\mathbf{e_{n+1}}$	-	max(605,660+30+30)=**720**
$\mathbf{l_{n+1}}$	-	min(780,565+240)=**780**
$\mathbf{a_0}$	-	100+(30+30+30-35)=**155**
$\mathbf{D_r}$	-	**130**
$\mathbf{c_r}$	-	30+30-35=**25**

The best alternative is to insert customer 5 into route 2 at position 2:
$r1$: [(710,725),175] – 1[(750,760),140] – 7[(825,835),65] – (890,900)
$r2$: [(480,565),155] – 3[(540,600),120] – 5[(660,720),60] – (720,780)
$r3$: [(600,960),0] – (600,960)

Iteration 5 - Insert customer 6

Insertion into route 1 is infeasible at all positions due to time window constraints.

r1	$\mathbf{e_j}$	$\mathbf{l_j}$
position 1	max(705,710+30)=**740**	min(750,760-60-15)=**685**
position 2	max(705,750+15+60)=**825**	min(750,835-20-25)=**750**
position 3	max(705,825+20+25)=**870**	min(750,900-15-30)=**750**

Likewise, insertion into route 2 is infeasible at positions 1 and 2.

r2	**position 1**	**position 2**
$\mathbf{e_j}$	max(705,480+30)=**705**	max(705,540+30+40)=**780**
$\mathbf{l_j}$	min(750,600-40-15)=**545**	min(750,720-15-15)=**690**

Two feasible alternatives exist for insertion of customer 6 – route 2 at position 3 or the empty route 3.

	r2, **position 1**	**r3**
$\mathbf{e_j}$	max(705,660+30+15)=**705**	max(705,600+30)=**705**
$\mathbf{l_j}$	min(750,780-15-30)=**735**	min(750,960-30)=**750**
$\mathbf{e_0}$	max(480,750-240)=**510**	max(600,750-300)=**600**
$\mathbf{l_0}$	min(565,735-15-30-95)=**595**	min(960,750-30)=**720**
$\mathbf{e_{n+1}}$	max(720,705+15+30)=**750**	max(600,705+15+30)=**750**
$\mathbf{l_{n+1}}$	min(780,595+300)=**780**	min(960,720+300)=**960**
$\mathbf{a_0}$	155+15+15+30-30=**185**	30+15+30 = **75**
$\mathbf{D_r}$	**160**	**30**
$\mathbf{c_r}$	15+30-30=**15**	30+30=**60**

Customer 6 is inserted into route 2 at the last position. Note, that the latest delivery time for customer 5 prior in the route is changed during the updating process, i.e. $l_5 = min(720, 735 - 13 - 30) = 690$. Furthermore, the

earliest delivery time for customer 3 is updated due to the change in e_0, i.e. $e_3 = max(540, 510 + 35) = 545$.

$r1$: [(710,725),175] – 1[(750,760),140] – 7[(825,835),65] – (890,900)
$r2$: [(510,565),185] – 3[(545,600),150] – 5[(660,690),90] –
– 6[(705,735),45] –(720,780)
$r3$: [(600,960),0] – (600,960)

Iteration 6 - Insert customer 2

Insertion of customer 2 into routes 1 and 3 is infeasible due to time windows, as $e_0^{r1} = 710 > L_2 = 480$ and $e_0^{r3} = 600 > L_2 = 480$. Insertion into route 2 is also infeasible due to the vehicle capacity constraint, as $D_2 + q_2 = 160 + 50 = 210 > Q_k = 200$.

Thus, customer 2 can not be feasibly inserted into the current routes and is placed into the pool of unserved customers.

Iteration 7 - Insert customer 8

Inserting customer 8 into route 1 at any position is infeasible due to the shift time limit. The cumulative travel time from depot to depot is $a_0 = 175$ and the service time of customer 8 is $s_8 = 40$. Thus, only the sum of these two components without including extra travel time will exceed the shift time limit of 180.

For route 2 the insertion is infeasible at position 2 and 3 with respect to the shift time limit. The values of a_0 if customer 8 is inserted at position 2 and 3 are calculated to 185+60+20+35−30 = 270 and 185+35+20+25−15 = 250, respectively. Both of these values are greater than the shift time limit of 240. For positions 1 and 4 of route 2, the insertion is infeasible with respect to the customer time windows:

r2	**position 1**	**position 4**
$\mathbf{e_j}$	max(750,510+20)=**750**	max(750,660+15+25)=**750**
$\mathbf{l_j}$	min(810,600-60-20)=**520**	min(810,780-20-20)=**740**

The only feasible alternative is the insertion of customer 8 into the empty route 3 – see the table below.

	r3
$\mathbf{e_j}$	max(750,600+20)=**750**
$\mathbf{l_j}$	min(810,960-20-20)=**810**
$\mathbf{e_0}$	max(600,790-300)=**600**
$\mathbf{l_0}$	min(960,810-20)=**790**
$\mathbf{e_{n+1}}$	max(420,750+20+20)=**790**
$\mathbf{l_{n+1}}$	min(960,790+300)=**960**
$\mathbf{a_0}$	20+20+20=**60**
$\mathbf{D_r}$	**40**
$\mathbf{c_r}$	20+20=**40**

The routes after insertion of customer 8 are:

$r1$: [(710,725),175] – 1[(750,760),140] – 7[(825,835),65] – (890,900)
$r2$: [(510,565),185] – 3[(545,600),150] – 5[(660,690),90] –
– 6[(705,735),45] –(720,780)
$r3$: [(600,790),60] – 8[(750,810),40] – (790,960)

Iteration 8 - Insert customer 4

Insertion of customer 4 is infeasible for routes 1 and 2 due to the vehicle capacity constraints, as $D_1 + q_2 = 90 + 70 = 160 > Q_k = 150$ and $D_2 + q_4 = 160 + 70 = 230 > Q_k = 200$ respectively for routes 1 and 2.

For route 3 the insertion is infeasible with respect to the time windows:

r3	**position 1**	**position 4**
$\mathbf{e_j}$	max(570,600+10)=**610**	max(570,750+20+35)=**805**
$\mathbf{l_j}$	min(600,810-35-30)=**600**	min(600,960-10-20)=**600**

Thus, customer 4 can not feasibly be inserted into the current routes and is added to the pool of unserved customers.

Finalizing the solution

To find the final solution, the following approach is applied in this thesis: The vehicle departs from the depot at the latest possible time l_0 and deliveries to the customers are then performed at the earliest feasible times. After a route is finalized, the duration of the route is calculated as the difference between the route completion time and the departure time for the depot. The cost of the route is then set to the route duration.

The final routes can be seen in figure B.2 below.

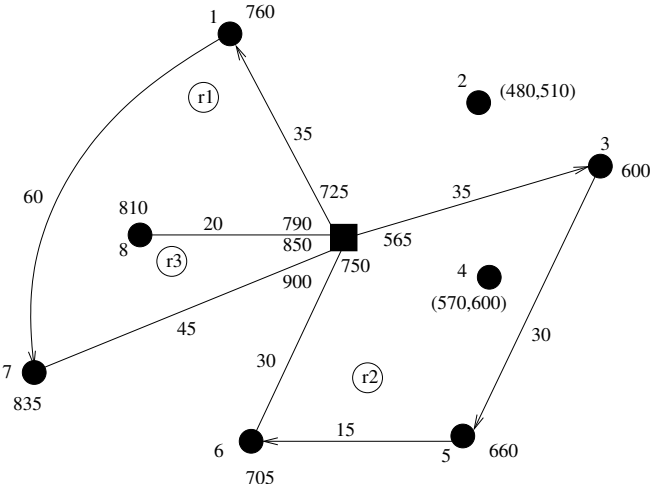


Figure B.2: Small example of insertion procedure – solution obtained with customers seeded according to decreasing distance from the depot. Customers 2 and 4 are still unserved. The numbers at the depot indicate the starting and completion times for each route.

The total cost of the routes is calculated as follows: $(900 - 725) + (750 - 565) + (850 - 790) = 175 + 185 + 60 = 420$.

To demonstrate the dependency of the final solution on the order in which customers are chosen for insertion, the insertion procedure is applied to the same set of customers but with a different seed criterion. In the following the solution produced by the insertion procedure with customers seeded in increasing earliest allowed starting time is presented.

Insertion procedure with customers seeded according to increasing earliest allowed starting time

The order in which the customers are inserted is {2, 3, 4, 5, 6, 1, 8, 7}. The routes obtained in this case after the insertion procedure is finished are:

$r1$: [(610,675),155] – 5[(660,705),125] – 6[(705,750),80] –
– 8[(750,810),40] – (790,855)

$r2$: [(420,490),135] – 2[(480,510),115] – 3[(540,555),85] –
4[(585,600),40] –(620,730)

$r3$: [(600,730),175] – 1[(750,765),140] – 7[(825,840),65] – (890,960)

The routes are displayed graphically in figure B.3 below.

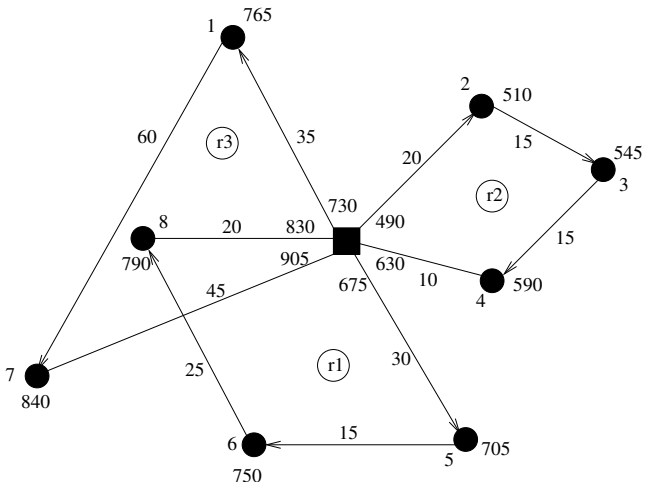


Figure B.3: Small example of the insertion procedure – solution obtained with customers seeded according to increasing e–values. The numbers at the depot indicate the starting and completion times for each route.

As it can be seen from the figure all customers are served in the solution. The total cost of the routes is calculated as follows: $(830 - 675) + (630 - 490) + (905 - 730) = 155 + 140 + 175 = 470$.

Appendix C

Generation of the Initial Lagrangian Multipliers

In the following, two approaches implemented and tested in this thesis to find the initial values of the Lagrangian multipliers are described. As mentioned in chapter 6 the optimal values of Lagrangian multipliers are the same as the optimal solution to the dual LP problem of SCCP, due to the integrality property of SCPP.

The dual problem of the LP relaxation of SCPP is given by:

$$\max \quad \sum_{i \in I^1} u_i + \sum_{i \in I^2} m_l \cdot u_i \tag{C.1}$$

$$\text{s.t.} \quad \sum_{i \in I_r} u_i \leq c_r \qquad \forall r \in R \tag{C.2}$$

$$u_i \geq 0 \qquad \forall i \in I^1 \tag{C.3}$$

$$u_l \leq 0 \qquad \forall i \in I^2 \tag{C.4}$$

Dual greedy heuristic

In the following it is assumed that the columns of the original problem are sorted so that $c_{r+1} \geq c_r$ for all $r \in R$. The constraints of the dual problem are then scanned in the natural order to construct a feasible solution.

For each u_i the following is defined:

$$\Delta_i(\mathbf{u}) = \min_{r \in R_i}(c_r - \sum_{i \in I_r} u_i)$$

and the set $I(\mathbf{u})$ is then defined as $I(\mathbf{u}) = \{i \in I : \Delta_i(\mathbf{u}) > 0\}$.

The algorithm for the greedy heuristic is presented below:

1. Initialize $u_i = 0$, for all $i \in I$.
2. Choose $i^* \in I(\mathbf{u})$ to minimize $|R_i|$. The ties are broken to maximize $\Delta_i(\mathbf{u})$. Set $u_i^* = u_{i^*} + \Delta_{i^*}(\mathbf{u})$.
3. Update $\Delta_i(\mathbf{u})$ for all $i \in I$ and $I(\mathbf{u})$.
4. Stop if $I(\mathbf{u}) = \emptyset$. Otherwise go to step 2.

To illustrate the heuristic consider the small problem from section 6.1 with 8 customers and 3 vehicles. The matrix for the primal problem is shown in the table below. In step 1 of the heuristic all dual values are set to 0. According to the table, four rows are covered by only one column, i.e. the minimal size of R_i. As all four rows have the same Δ-values, the tie can be broken arbitrarily. If row 1 is chosen, u_1 is set to 120 and the Δ-values for rows 2, 3 and 9 are updated, i.e. set to 0. Thus, after first iteration set of available rows to choose from is $I(\mathbf{u}) = \{0, 4, 5, 6, 7, 8, 10\}$.

	Cost, c_r						
Rows, i	120	140	155	175	180	$\Delta_i(\mathbf{u})$	$\lvert R_i \rvert$
0	1			1		120	2
1		1				140	1
2		1				140	1
3		1				140	1
4			1		1	155	2
5			1		1	155	2
6				1	1	175	2
7	1		1			120	2
8			1		1	155	2
9		1				140	1
10	1			1		120	2

In the second iteration row 6 will be chosen, as all the remaining rows are covered by the same number of columns, and $\Delta_6 u$ has the largest value. u_6 is then set to 175, and the Δ-values for rows {0,10,4,5} are updated.

After the dual heuristic is finished, the following solution is obtained: $\mathbf{u} = \{0, 140, 0, 0, 5, 0, 175, 120, 0, 0, 0\}$, where the first 8 entries correspond to the

Lagrangian multipliers λ_i for the set covering constraints for all $i \in I^1$. The last three entries in the solution vector are the multipliers for the set packing constraints, μ_i, for all $i \in I^2$. The value of objective function, hence the lower bound on the solution value of the original problem, is 440.

Optimal solution by COIN

The Common Optimization INterface for Operations Research (COIN-OR) is a collection of Open-source libraries, which can be used to solve different kinds of optimization problems. In this project the Open Solver Interface (OSI) component of COIN is used. OSI is an interface for calling mathematical programming solvers, including the commercial solver CPLEX available at IMM, DTU.

As the problem that needs to be solved is an LP problem, the built-in Open Source COIN-LP solver can be used. One of the advantages of using LP solver in this case is that the program is not dependent on the availability of a commercial solver, e.g. CPLEX.

To solve a problem using COIN, the information about the problem can be read in from an mps file or the elements of the model can be constructed and loaded into the solver interface. The last approach is used in this thesis.

The model components which need to be constructed are the objective coefficients, columns' upper and lower bounds, the constraint matrix and the upper and lower bounds for the rows in the model. The number of variables equals the number of columns in the problem, and for each variable there is a lower and upper bound. Each constraint of the problem corresponds to a row of the COIN model and has a lower and upper bound.

For the small problem described above, the COIN model with 11 columns and 5 rows is constructed, loaded into the LP-solver and solved to optimality yielding the following solution to the dual LP problem: $\mathbf{u} = \{120, 0, 0, 140, 0, 155, 25, 0, 0, 0, 0\}$. The objective value is 440 which is the same as the value obtained by the dual heuristic.

This will not be the case for the problems of large sizes. Obviously, the optimal solution to the dual LP problem obtained by COIN provides a better starting point for the Lagrangian heuristic described in chapter 6 than a heuristic solution.

As a final remark it needs to be said that during the implementation and test of the Lagrangian heuristic, the CPLEX solver is used in the programme. This is done to be able to compute the optimal solutions for the SCPP test

instances. The optimal solution values are then used to evaluate the performance of the Lagrangian heuristic. In the final implementation the built-in LP solver is used to make the programme independent of the commercial solvers.

Appendix D

Test of Improvement Moves

In the following results of experiments with different improvement strategies are presented.

Improvement after the insertion procedure without embedded improvement

First, the improvement moves are tested on the routes constructed by the insertion procedure without the embedded improvement. Customers are sorted according to decreasing distance from the depot and the insertion procedure with subsequent improvement is performed 10 times.

As mentioned in section 5.3 the improvement moves are applied to each route of the route set generated by the insertion procedure. *Re-insertion* and *swap* moves consider two customers on each routes as candidate for the move. These customers are the ones by removal of which from the route the largest savings in travel time can be achieved. For the sake of experiments the third improvement move, *random-swap*, with an element of randomness is introduced: for each route a random number between 0 and n_r is generated. n_r is the number of customers in the route, and the generated number indicates how many of these will be considered as candidates for a swap move. The candidate customers are then selected randomly.

Tables D.1-D.3 show the improvement that can be achieved by applying *re-insertion*, *swap* or *random-swap* moves after the insertion procedure has finished. The entries in the last line of each table are the averages of the values in the corresponding columns.

Run No.	Total cost		Improvement %
	before re-insertion	after re-insertion	
1	25038.1	24150.6	3.55
2	25519.5	24098.7	5.57
3	26397.8	25860.9	2.03
4	26612 2	25282.7	4.995
5	26448.2	26135.8	1.18
6	27475.9	25906.7	5.71
7	27505	26059.2	5.26
8	26952.2	25669.7	4.76
9	27972	26947.5	3.66
10	29149.5	27545.7	5.50
average	26907.04	25765.75	4.22

Table D.1: Solution improvement by the re-insertion move.

By applying the re-insertion improvement moves, the solution obtained from the insertion procedure can be improved by 4.22% on average.

Run No.	Total cost		Improvement %
	before swap	after swap	
1	25038.1	23974.6	4.25
2	25160.7	24044.4	4.44
3	26525.4	25527.9	3.76
4	26094.4	24833.3	4.83
5	25595.5	25233.3	1.42
6	27403.8	25978.1	5.20
7	27457.3	26660.1	2.90
8	27330	26769.5	2.05
9	27864.9	26897.4	3.47
10	27720.3	27207.9	1.85
average	26619.04	25712.65	3.42

Table D.2: Solution improvement by the swap move.

Applying the swap moves to the solution obtained from the insertion heuristic results in an average improvement of 3.42%. Performing the improvement procedure based on swap moves takes considerably longer time than using the re-insertion moves. This is not surprising considering the time complexity of the moves discussed in section 5.3.

Run No.	Total cost		Improvement %
	before r-swap	after r-swap	
1	25038.1	24910.7	0.51
2	25651.3	25471.8	0.699
3	26371.5	26328	0.17
4	25680.5	24997.9	2.66
5	27077.3	26579.6	1.84
6	26425.7	26117	1.17
7	27318.4	27089.6	0.84
8	28081.8	27342.3	2.63
9	27203.2	26769.9	1.59
10	27714.9	27266.8	1.62
average	26656.27	26287.36	1.37

Table D.3: Solution improvement by the random-swap move.

The average solution improvement when applying the random-swap moves is 1.37%, which makes this improvement strategy the worst performing compared to the other two.

The total costs of the obtained routes can be directly compared only for the first iteration, where the three improvement strategies have the same starting point. This is due to the fact that the routes constructed in the next execution of the route generation phase depend on the final routes produced in the previous run. The tables show that the best result in the first run is obtained by applying the swap moves, but at a higher computational cost. On average, applying the swap moves yielded better results than using the re-insertion moves but the difference was only about 0.2%.

Improvement after the insertion procedure with embedded improvement

Due to the long computation time of the improvement procedure based on the swap moves, it was decided not to use this type of moves in the embedded route improvement. The only type of moves applied in the embedded route improvement is re-insertion.

Tables D.4-D.6 contain the results obtained by the insertion procedure with embedded re-insertion followed by different improvement strategies.

It is quite obvious that embedded improvement proves to be very beneficial in

Run No.	Total cost		Improvement %
	with embedded re-ins	after re-ins	
1	21402.2	21377.4	0.12
2	21910.3	21770.7	0.64
3	22281.8	22213.3	0.31
4	23517.6	22928.6	2.51
5	22552.6	22021.5	2.36
6	23112	22540.2	2.47
7	23615.5	22998.9	2.61
8	23232.6	23124.4	0.47
9	23625.9	22907.6	3.04
10	23319.7	23177.4	0.61
average	22817.69	22506	1.51

Table D.4: Solution obtained by the insertion procedure with embedded re-insertion moves and applying re-insertion moves afterwards.

Run No.	Total cost		Improvement %
	with embedded re-ins	after swap	
1	21402.2	21245.2	0.73
2	22492.2	21976.6	2.29
3	22215.9	22093.1	0.55
4	22934.1	22669.3	1.16
5	22486.8	22242.1	1.09
6	22759.2	22635.7	0.54
7	23279.5	22822.6	1.96
8	23076.2	22798	1.21
9	24379.1	24061.4	1.30
10	23513.8	23238.7	1.17
average	22853.9	22578.27	1.20

Table D.5: Solution obtained by the insertion procedure with embedded re-insertion moves and applying swap moves afterwards.

terms of solution quality. The best solution obtained by insertion procedure followed by the improvement procedure was 23974.6 minutes, see table D.2. For the insertion procedure with embedded improvement, all the obtained solutions, except two, are below this value. Further improvements of about 0.8-1.5% on average were achieved by applying the improvement procedure after the insertion procedure had finished. The random-swap improvement

Run No.	Total cost		Improvement %
	with embedded re-ins	after swap	
1	21402.2	21277.2	0.58
2	21841	21663.5	0.81
3	22113.6	22003.8	0.49
4	23088.8	22836.9	1.09
5	24246.9	24060.5	0.77
6	23440.4	23129.1	1.33
7	24280.7	23916.5	1.499
8	24036.3	23860.2	0.73
9	23620.2	23572.4	0.20
10	23735.7	23619.9	0.49
average	23180.58	22994	0.80

Table D.6: Solution obtained by the insertion procedure with embedded re-insertion moves and applying random-swap moves afterwards.

procedure was once again the worst performing one, while the difference in performance between the other two was insignificant.

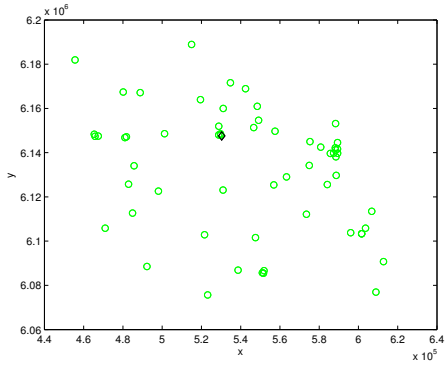
Appendix E

Data for the Generated Problem Instances

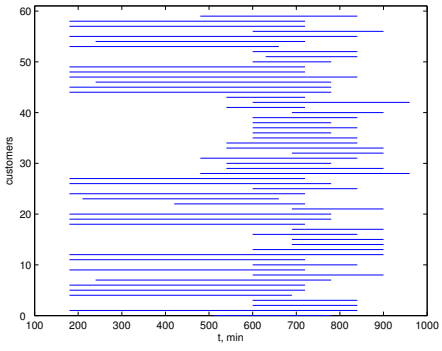
As mentioned in section 7.3 a number of problem instances was constructed to test the performance of the Lagrangian heuristic. In the following the geographical distribution of customers and the customer time windows are illustrated for the problems of each type.

Problem Instances of type *micro*

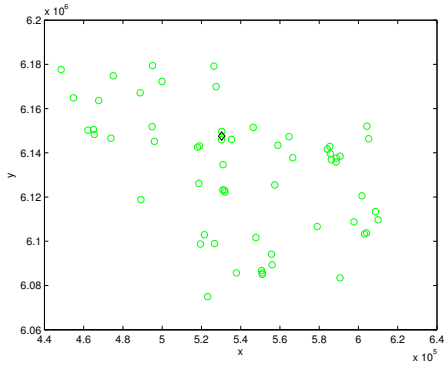
Problem instances of type *micro* contain 60 customers and 3 vehicles of 3 different types, i.e. 9 vehicles in total. The problems are denoted $M1$, $M2$ and $M3$.



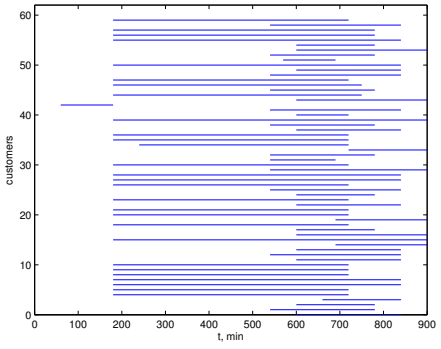
(a) M1, geography.



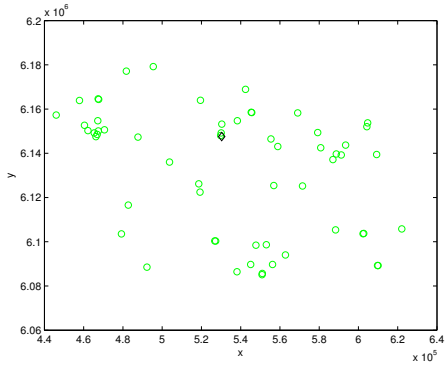
(b) M1, time windows.



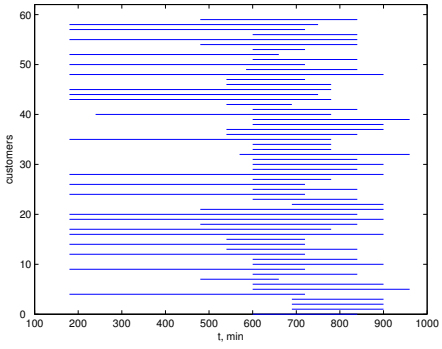
(c) M2, geography.



(d) M2, time windows



(e) M3, geography.

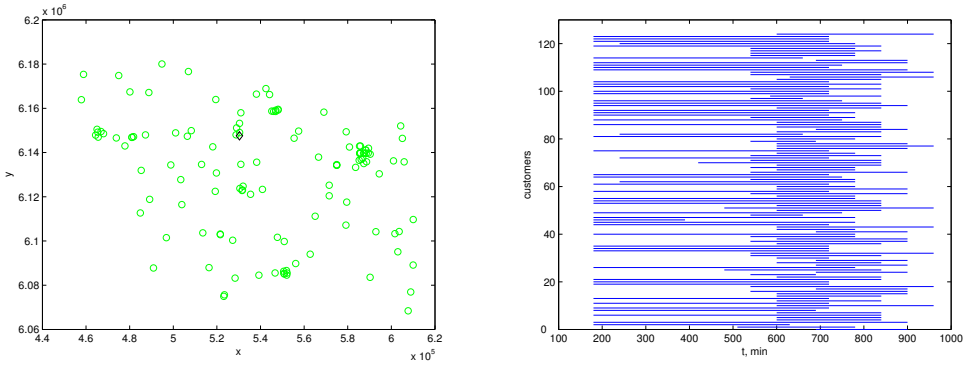


(f) M3, time windows

Figure E.1: Data for the problems of type *micro*.

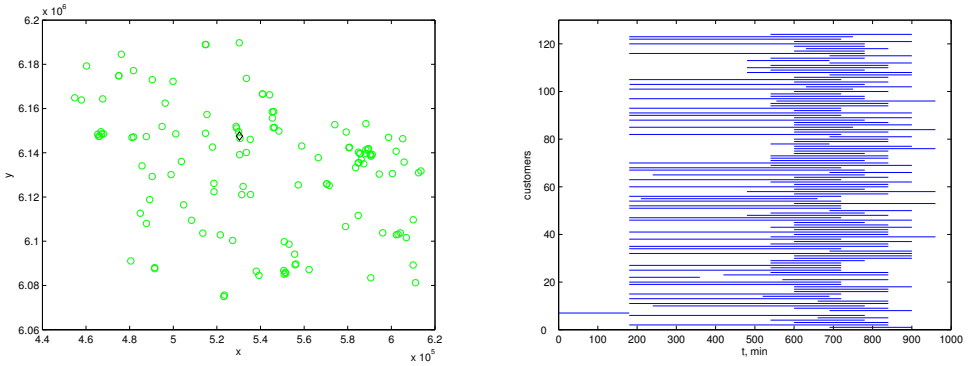
Problem Instances of type *small*

Problem instances of type *small* contain 125 customers and 5 vehicles of 3 different types, i.e. 15 vehicles in total. The problems are denoted $S1$, $S2$ and $S3$.



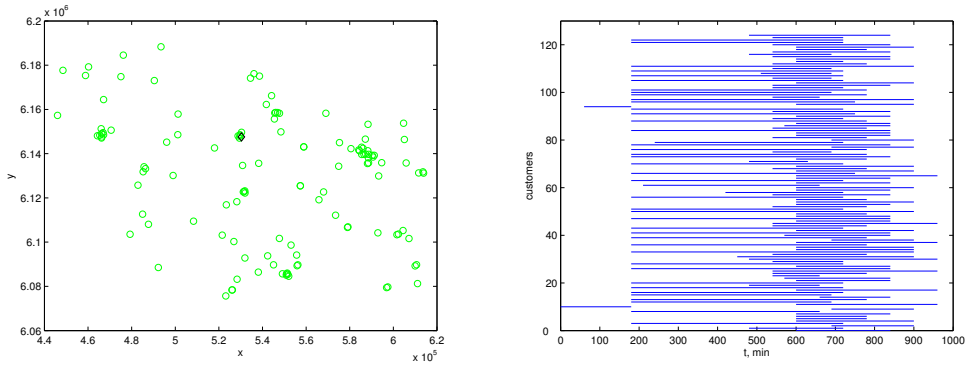
(a) S1, geography. (b) S1, time windows.

Figure E.2: Data for problem S1.



(a) S2, geography. (b) S2, time windows.

Figure E.3: Data for problem S2.



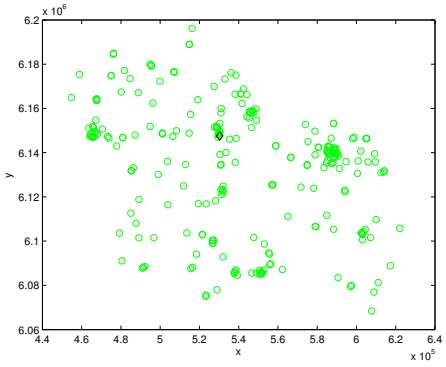
(a) S3, geography.

(b) S3, time windows.

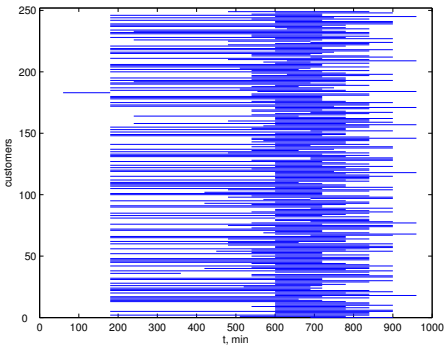
Figure E.4: Data for problem S3.

Problem Instances of type *medium*

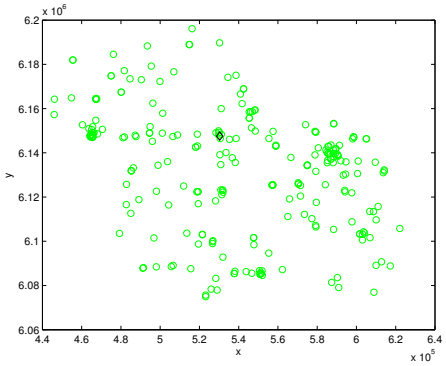
Problem instances of type *medium* contain 125 customers and 10 vehicles of 3 different types, i.e. 30 vehicles in total. The problems are denoted $Md1$, $Md2$ and $Md3$.



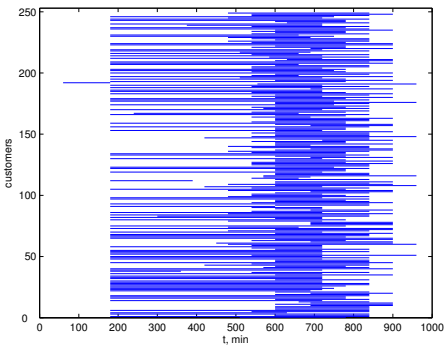
(a) Md1, geography.



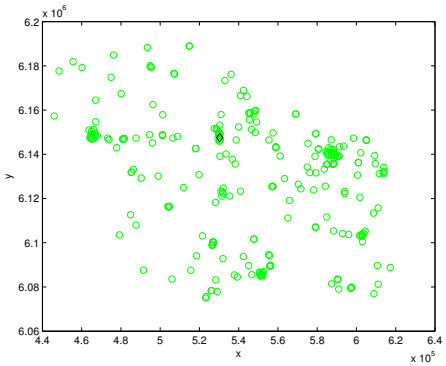
(b) Md1, time windows.



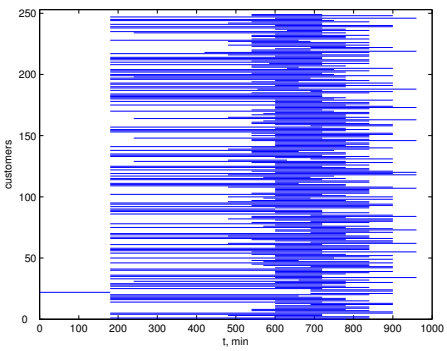
(c) Md2, geography.



(d) Md2, time windows



(e) Md3, geography.



(f) Md3, time windows

Figure E.5: Data for the problems of type *medium*.

Appendix F

Results for Adjusting Master Plans According to Demand Variation

In the following tables the results obtained by adjusting the master plans according to the changes in customer demand are shown. Different strategies for adjusting the master plans are presented in section 8.2 of chapter 8.

Modification strategies

Set No.	Total route cost							
	Simple removal	**Red. %**	**With re-ins**	**Red. %**	**With swap**	**Red. %**	**With re-insWR**	**Red. %**
1	20729.46	2.43	20161.25	5.10	20479.40	3.60	20694.24	2.59
2	20990.08	1.20	20572.41	3.17	20738.80	2.38	20963.25	1.33
3	20912.93	1.56	20372.12	4.11	20500.37	3.51	20879.18	1.72
4	20976.10	1.27	20450.67	3.74	20812.63	2.04	20900.83	1.62
5	20963.20	1.33	20502.80	3.49	20634.86	2.87	20914.82	1.56
	20914.35	1.56	20411.85	3.92	20633.21	2.88	20870.47	1.76

Table F.1: New solutions obtained for a demand change of 5%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	Total route cost							
	Simple removal	**Red. %**	**With re-ins**	**Red. %**	**With swap**	**Red. %**	**With re-insWR**	**Red. %**
1	20799.19	2.10	20061.17	5.57	20373.70	4.10	20745.84	2.35
2	20787.07	2.16	20120.71	5.29	20574.81	3.16	20745.82	2.35
3	20845.92	1.88	20260.24	4.64	20420.14	3.88	20764.40	2.26
4	20752.20	2.32	20229.98	4.78	20441.29	3.78	20658.77	2.76
5	20780.25	2.19	20319.06	4.36	20495.44	3.53	20732.77	2.41
	20792.92	2.13	20198.23	4.93	20461.08	3.69	20729.52	2.43

Table F.2: New solutions obtained for a demand change of 7%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	Total route cost							
	Simple removal	**Red. %**	**With re-ins**	**Red. %**	**With swap**	**Red. %**	**With re-insWR**	**Red. %**
1	20549.31	3.28	19673.74	7.40	20108.66	5.35	20471.91	3.64
2	20613.27	2.97	19884.43	6.41	20427.43	3.85	20546.39	3.29
3	20514.74	3.44	19749.21	7.04	19952.95	6.08	20421.30	3.88
4	20301.11	4.44	19534.08	8.05	19947.37	6.11	20202.56	4.91
5	20581.71	3.12	19892.88	6.37	20114.08	5.32	20524.72	3.39
	20512.03	3.45	19746.87	7.05	20110.10	5.34	20433.38	3.82

Table F.3: New solutions obtained for a demand change of 10%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	Total route cost							
	Simple removal	**Red. %**	**With re-ins**	**Red. %**	**With swap**	**Red. %**	**With re-insWR**	**Red. %**
1	20569.08	3.18	19413.76	8.62	20068.95	5.54	20483.22	3.59
2	20333.33	4.29	19529.31	8.08	19743.43	7.07	20226.88	4.79
3	20472.05	3.64	19390.74	8.73	19922.87	6.22	20333.48	4.29
4	19890.13	6.38	19046.69	10.35	19514.63	8.15	19808.31	6.76
5	20365.99	4.14	19625.70	7.62	19705.54	7.25	20294.29	4.48
	20326.11	4.33	19401.24	8.68	19791.08	6.84	20229.24	4.78

Table F.4: New solutions obtained for a demand change of 12%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	Total route cost							
	Simple removal	**Red. %**	**With re-ins**	**Red. %**	**With swap**	**Red. %**	**With re-insWR**	**Red. %**
1	19810.15	6.75	18869.49	11.18	19182.76	9.71	19702.23	7.26
2	19904.67	6.31	18943.05	10.84	19452.59	8.44	19772.46	6.93
3	19902.66	6.32	19045.82	10.35	19496.74	8.23	19806.03	6.77
4	20200.24	4.92	19255.01	9.37	19602.23	7.73	20024.42	5.75
5	20323.29	4.34	19464.26	8.38	19787.03	6.86	20225.97	4.80
	20028.20	5.73	19115.52	10.02	19504.27	8.19	19906.22	6.30

Table F.5: New solutions obtained for a demand change of 15%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	Total route cost							
	Simple removal	**Red. %**	**With re-ins**	**Red. %**	**With swap**	**Red. %**	**With re-insWR**	**Red. %**
1	19882.82	6.41	18486.42	12.99	19354.10	8.90	19797.41	6.81
2	19721.81	7.17	18686.42	12.04	19292.90	9.19	19663.67	7.44
3	19974.14	5.98	18755.15	11.72	19309.88	9.11	19879.00	6.43
4	19907.26	6.30	18681.70	12.07	19472.04	8.35	19884.13	6.41
5	19925.13	6.21	18766.43	11.67	19210.75	9.58	19791.11	6.84
	19882.23	6.42	18675.22	12.10	19327.94	9.02	19803.06	6.79

Table F.6: New solutions obtained for a demand change of 20%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

For all scenarios in demand variation the best performing method is removing the customers with cancelled deliveries from their respective routes and then applying the re-insertion moves to the routes. The second best approach is to apply the swap moves instead of the re-insertion moves.

Reconstruction strategy

In the following the results obtained by applying the reconstruction strategy are presented in the number of tables, one for each demand variation scenario.

Set No.	**No. of vehicles**	**Total Cost**	**Red. %**	**Time sec.**
1	57	20007.09	5.83	30.62
2	56	20292.18	4.49	31.51
3	57	20065.44	5.55	31.53
4	57	19702.10	7.26	31.83
5	56	20496.13	3.53	30.95
	56	20112.59	5.33	31.29

Table F.7: New solutions obtained by the reconstruction strategy for a demand change of 5%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	**No. of vehicles**	**Total Cost**	**Red. %**	**Time sec.**
1	57	19860.48	6.52	29.77
2	55	20477.91	3.61	29.68
3	56	20091.12	5.43	29.25
4	55	19913.12	6.27	29.18
5	55	19694.02	7.30	29.49
	55	20007.33	5.83	29.47

Table F.8: New solutions obtained by the reconstruction strategy for a demand change of 7%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	No. of vehicles	Total Cost	Red. %	Time sec.
1	54	18904.40	11.02	27.86
2	55	19672.17	7.40	27.77
3	54	18870.40	11.18	28.56
4	54	19596.98	7.76	27.10
5	54	19692.64	7.31	28.33
	54	19347.32	8.93	27.92

Table F.9: New solutions obtained by the reconstruction strategy for a demand change of 10%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	No. of vehicles	Total Cost	Red. %	Time sec.
1	53	18840.02	11.32	25.98
2	53	19328.47	9.02	25.95
3	53	19040.42	10.38	26.35
4	52	18794.43	11.54	26.40
5	53	18952.82	10.79	27.00
	52	18991.23	10.61	26.34

Table F.10: New solutions obtained by the reconstruction strategy for a demand change of 12%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	No. of vehicles	Total Cost	Red. %	Time sec.
1	52	19056.04	10.30	25.53
2	51	18064.03	14.97	24.72
3	50	18153.30	14.55	24.49
4	52	18596.95	12.47	24.28
5	51	19053.81	10.31	24.78
	51	18584.83	12.52	24.76

Table F.11: New solutions obtained by the reconstruction strategy for a demand change of 15%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.

Set No.	**No. of vehicles**	**Total Cost**	**Red. %**	**Time sec.**
1	48	17480.46	17.72	21.44
2	50	18393.21	13.42	21.31
3	48	17049.39	19.75	21.98
4	48	17376.75	18.21	20.97
5	48	17123.90	19.40	21.52
	48	17484.74	17.70	21.44

Table F.12: New solutions obtained by the reconstruction strategy for a demand change of 20%. Reduction in total cost is calculated with respect to the master plan solution value, 21245.2 minutes.