Robust scheduling of the vehicle routing problem with time windows

M.M. Lekkerkerker

Supervised by
dr. ir. D.A. Roozemond and dr. J.A. Hoogeveen

A thesis presented for the degree of
Master in Computer Science

Universiteit Utrecht

QUINTIQ

ICA-3873994
September, 2016

Abstract

In this thesis we extend the Vehicle Routing Problem with Time Windows by adding time-dependent and stochastic travel times. By allowing any probability distribution to represent our stochastic travel times at any point of time, we allow great flexibility in defining these travel times. Feasibility of a route is defined using reliabilities; the probability that the vehicle will arrive on time at the customer.

This problem is solved using a column generation heuristic. To solve the pricing problem, we define four methods; two mixed integer programs, a local search heuristic and a dynamic programming heuristic. We will compare these methods to find out which performs best on our problem.

Taking the new feasibility criterion for routes into account, we will calculate the latest possible departure time from the depot for a given route. This can be done using different methods, which we will explore.

Acknowledgment

This thesis is part of the Data and Algorithms for Integrated Transportation Planning and Execution project, created by Dinalog. The DAIPEX project is a collaboration between Quintiq, TomTom, TU/e and four logistic companies. The goal of this collaboration is to develop information aggregation and optimization algorithms to generate robust and efficient transportation plans for realistic transportation planning problems including uncertain travel and service times.

I would like to thank all the people in Products and Predictive Analytics at Quintiq for their help. In particular I would like to thank Dan Roozemond, Wim Nuijten, Daan Creemers and Edwin de Jong for their help and advice. My supervisors, Dan Roozemond and Han Hoogeveen were a great inspiration to me, they encouraged me to explore new methods and they pointed me in the right directions.

I would like to thank Paul for his moral support and my parents for being supportive.

Contents

Chapter 1

Introduction

The Traveling Salesperson Problem (TSP) consists of minimizing the total travel cost of a salesperson, while traversing a set of cities and returning to the original city. The travel cost can be connected to travel duration, monetary costs, route length, a quantification of comfort on the road, or any other measurement imaginable.

The Truck Dispatching Problem is formulated by Dantzig and Ramser [4]. This would later be known as the first Vehicle Routing Problem (VRP). They proposed a method to solve this problem to near-optimality. It can be solved manually or by using an 'automatic digital computing machine'. The classical VRP can be seen as a TSP with multiple salespersons. All salespersons are originally located in a single city (depot) and every other city should be visited by one of the salespersons. In VRP the cities are described as customers and the salespersons as vehicles. Besides minimizing travel cost, some instances also aim to minimize the number of resources, like drivers, trailers and trucks used. We must note that another definition of the VRP can be found in literature. According to some authors, capacity constraints on the vehicles are also part of the VRP definition. As described by Bektas [1], the capacity constraints are the only difference between VRP and the Multiple Traveling Salesperson Problem (mTSP).

1.1 Variations

The Pickup and Delivery Problem (PDP) is a close sibling of the VRP. It adds precedence constraints to the problem, by specifying that a vehicle should pickup a package at one customer before delivering it to the other. The Dial A Ride Problem (DARP) extends the PDP by allowing only one

package in the vehicle at a given time. This problem is often found in taxi services, with passengers as packages. Many more variations exist, as listed below. This list is certainly not exhaustive, but it contains the most commonly described variations.

CVRP In the Capacitated Vehicle Routing Problem, every customer has a non-negative demand for commodities. Every vehicle has a non-negative capacity of these commodities. The combined demand of all customers in a route may not surpass the capacity of the vehicle that services them.

VRPB The Vehicle Routing Problem with Backhauls allows negative customer demands in the CVRP. This means that vehicles can also pickup goods and deliver these to the depot. Usually vehicles are only allowed to pickup goods after they have delivered all their goods from the depot.

MDVRP The Multi-Depot Vehicle Routing Problem allows for more than one depot. Vehicles can return to any depot, but the number of vehicles returning to a given depot should be equal to the number of vehicles departing from it.

VRPTW In the Vehicle Routing Problem with Time Windows, customers can specify a time window in which the vehicle should arrive. This variation can be subdivided into soft and hard time windows. The difference is that by violating a soft time window a penalty cost is calculated, while hard time windows may not be violated.

DVRP The Dynamic Vehicle Routing Problem is different from the VRP due to the fact that it has to be able to handle changes. During the day new customers arrive which have to be incorporated in the schedule, while minimizing the travel cost. Algorithms that solve changing problems are called on-line, since they are reacting to changes in real-time.

1.2 Formal problem definition

The Time Dependent Stochastic Vehicle Routing Problem with Time Windows consists of a set of customers C, a depot δ, a set of vehicles V and a set of time periods P. Every customer should be serviced exactly once by one of the vehicles in V. To service a customer, a vehicle has to travel to

the customer. When a vehicle arrives at customer $c \in C$, it takes s_c time to service the customer. The service time s_c is independent of the vehicle that services the customer. Every customer $c \in C$ has a time window $[tw_c^o, tw_c^c]$. The vehicle servicing this customer should arrive before tw_c^c and after tw_c^o. If the vehicle arrives at the customer before tw_c^o it has to wait until tw_c^o to service the customer.

The vehicles in V are homogeneous, meaning every vehicle in V is identical. In this thesis we assume that sufficient vehicles are available, so the amount of vehicles will never be the bottleneck. Using a vehicle costs vc, independent of the amount of customers served. The objective of our problem is to create a set of routes, minimizing their combined costs, thereby satisfying all constraints below. A route r is an ordering of customers; r_i is the i^{th} customer visited in the route and as we start and end at the depot, we define $r_0 = r_{|r|+1} = \delta$.

By combining the depot δ and the customers in C we get the set of nodes N. The travel time from node $i \in N$ to node $j \in N$ is dependent on the departure time d from node i and is stochastically defined by the function $T_{ij}(d)$, which returns a random variable. The function $T_{ij}(d)$ uses an interpolation technique described in Section 2.1 to combine the random variables T_{ij}^p, which represent the travel time between node i and j in every time period $p \in P$. Every time period $p \in P$ is defined by its starting time tp_p^o and its ending time tp_p^c. The time periods in P connect to one another. This means that for every $p \in P$ either there is a $q \in P$ such that $tp_p^o = tp_q^c$ or there is no $q \in P$ such that $tp_p^o > tp_q^o$. There are no $p, q \in P$ such that $tp_p^o = tp_q^o$ or $tp_p^c = tp_q^c$.

Traveling from node $i \in N$ to node $j \in N$ costs c_{ij}, which is a combination of the travel distance and the average travel duration. Since we are using stochastic travel times, we are uncertain if a vehicle will be on time at the customer. Therefore for every route r we define the reliability ry_c^r for customer $c \in r$ as the ratio of on time services if the route was driven infinitely many times. Unless stated otherwise, we assume that the vehicle departs from the depot as early as possible. Every customer $c \in C$ has a minimum reliability m_c, with $0 \leq m_c \leq 1$. A route r is feasible if $ry_c^r \geq m_c$ for every customer $c \in r$. Only feasible routes are allowed in our solution.

1.3 Literature review

The VRP is NP-hard, which means that it is not easier than the hardest problem in NP. Therefore it may take a lot of resources (time and memory)

to find an optimal solution. Heuristics and local search algorithms have been deployed to find good solutions. Ten different algorithms to solve the Vehicle Routing Problem are reviewed by Laporte [15]. Four of these algorithms use heuristics to find a good solution, while the other six use dynamic programming, direct tree search and integer linear programming to solve the problem to optimality. Hard time windows were accounted for in the *three-index vehicle flow formulation* by Fisher and Jaikumar [7]. Column generation was used by Sol [18] to solve the pickup and delivery problem with time windows; this solution is then adjusted to allow on-line scheduling.

1.3.1 Stochastics

A deterministic world is often assumed, while most planning decisions in logistics are based on non-deterministic data. Algorithms were proposed by Stewart and Golden [20] to account for stochastic demands in the vehicle routing problem. When a VRP has stochastic demands, the demand of each customer is uncertain before arrival. More than 30 years later, a solution that takes stochastic demands and time windows into account was proposed by Zhang, Lam and Chen [22].

Another stochastic variable in logistics is the travel time, as in our problem. Travel times between two locations vary due to traffic jams, traffic lights, etc. Taking the average travel time to calculate the duration of a route will often result in violated time windows. Using discrete cumulative distribution functions to represent the travel times between customers, Fasting, Firat, Boon and Twist [6] propose a method to find a robust solution for the Vehicle Routing Problem with Hard Time Windows and Stochastic Travel Times. To find an optimal solution for problems with up to 25 customers, the method uses column generation with a dynamic programming algorithm to solve the pricing problem. Their method does not take time dependent travel times into account, which our method will. A local search approach and some heuristics to solve this problem are proposed by Conijn and Nuijten [3].

The time it takes to get from one customer to another is also highly dependent on the time of day. During rush hour it is likely that it will take longer than at midnight. Besides time of day, day of the week can also be an important factor. Traffic is different during the weekends and there is a lot of variety from day to day. Column generation combined with an evolutionary algorithm was used by Koning, van den Akker and Hoogeveen [14] to solve the PDP with Time Windows and Disturbances. Their method

uses normal distributions for stochastic travel times and a separate delay function for time dependency.

Our method incorporates the time dependent nature of travel times within the distributions, allowing great flexibility in the definition of time dependent stochastic travel times. We will adapt multiple approaches for the VRPTW to account for these time dependent stochastic travel times. Finally we explore methods to determine the latest possible departure time from the depot, such that the route is feasible.

This thesis is organized as follows. In Chapter 2 we will explain some principles which will be used throughout the thesis. Chapter 3 presents the method that we will use to solve the problem. In Chapter 4 we explore three methods to compute the latest departure time from the depot for a given route, while taking the minimum reliability constraint into account. In Chapter 5 we present the results of our experiments and we summarize our discoveries in Chapter 6.

Chapter 2

Preliminaries

2.1 Stochastic representation

In our problem we define the travel time between nodes $i, j \in N$ at departure time d using the stochastic variable returned by the function $T_{ij}(d)$. To define this function, we use the stochastic variables T_{ij}^p for every time period $p \in P$. The stochastic variable T_{ij}^p is characterized by the quantile function $Q_{ij}^p(q)$, with q a cumulative probability between 0 and 1. Using this function, we find that for an arbitrary cumulative probability q, the travel time between nodes i and j in time period p is at most $Q_{ij}^p(q)$. In Figure 2.1 an example of a quantile function is shown.

2.1.1 Traversing over time periods

The quantile functions only make sense when both the departure time and the arrival time are within the same time period. For instance, suppose that we have two time periods p_1 and p_2 with $tw_{p_1}^o = 0, tw_{p_1}^c = 4, tw_{p_2}^o = 4, tw_{p_2}^c = 8$. For given nodes i, j and cumulative probability q we find that $Q_{ij}^{p_1}(q) = 2$ and $Q_{ij}^{p_2}(q) = 1$. Assume the earliest departure time from i is at time 3.5, so this is within time period p_1. If we were to depart at time 3.5, we would arrive at 5.5, while if we would wait for time 4 to depart, we would arrive at time 5. This is an undesirable behavior, as this is never the case in reality. In Figure 2.2 this is visualized.

So we need to define a method to combine the quantile functions when the time period of the arrival time is not equal to the time period of the departure time. Our time-dependent travel times should behave according to the first in first out (FIFO) assumption. This means that when two

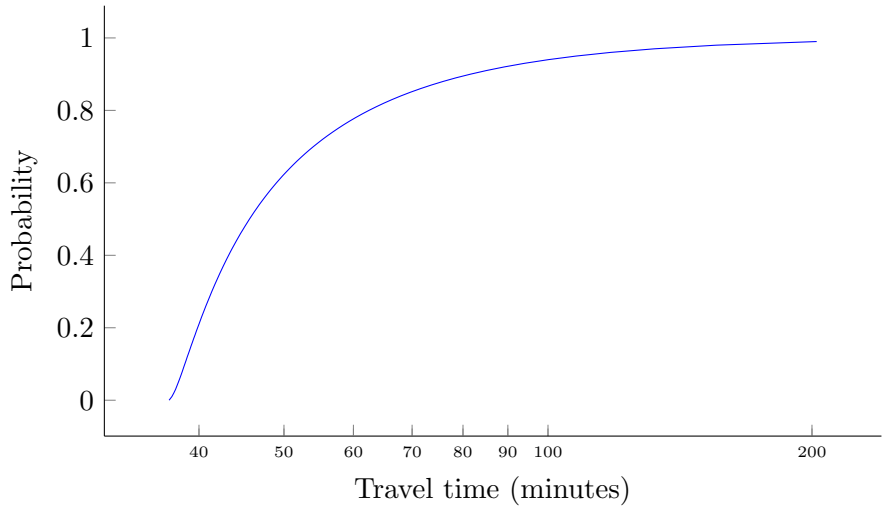

Figure 2.1: The quantile function between Den Bosch and Zeist between 6AM and 10AM plotted on logarithmic x-axis.

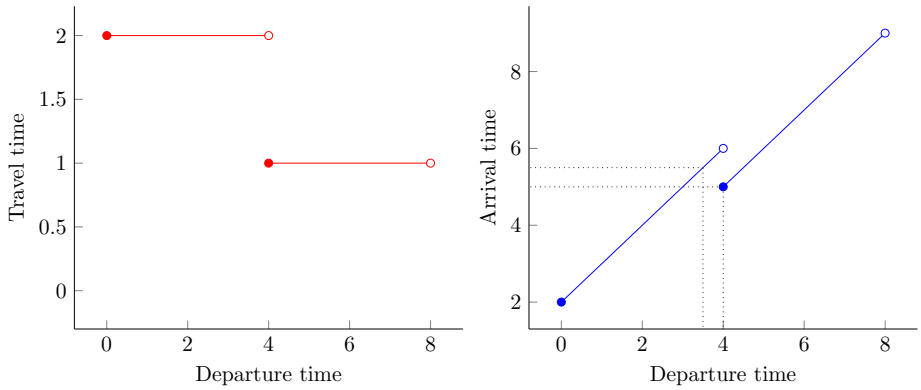

Figure 2.2: Using travel time function (left) and arrival time function (right) for a given probability we sketch the problem in our model that by departing later, we can arrive earlier.

identical trucks drive the same route, the one that departed earlier than the other should arrive first.

Ichoua, Gendreau and Potvin [11] present a speed model to represent time-dependent travel times which behaves according to the FIFO assumption. We will combine their model with our stochastic representation, to create a model with time-dependent stochastic travel times. We use multiple quantile functions to construct the travel time for a given cumulative probability q and departure time d. First we find the time period p with $tp_p^o \leq d < tp_p^c$. In our example $d = 3.5$ and therefore we find time period p_1. Our travel time will be $Q_{ij}^{p_1}(q) = 2$ if $d + Q_{ij}^{p_1}(q) < tp_{p_1}^c$; otherwise we will travel until $tp_{p_1}^c$ in time period p_1 and the remainder of the trip in the next time period. Unfortunately $3.5 + 2 < 4$ does not hold, so we have to continue our calculation. We will travel in time period p_1 until time $tp_{p_1}^c$, therefore the part of the trip we travel in time period p_1 can be calculated using

$$\gamma = \frac{tp_{p_1}^c - d}{Q_{ij}^{p_1}(q)} = \frac{4 - 3.5}{2} = 0.25$$

This means we still have to travel $\beta = 1 - \gamma = 0.75$ part of the trip in the next time period. Now we set the departure time d to $tp_{p_1}^c = 4$ and we perform these steps again, taking into account that we have traveled 0.25 part of the total trip. This results in traveling 0.25 part in time period t_1 and 0.75 part in time period t_2. Our combined travel time is now $0.25Q_{ij}^{p_1}(q) + 0.75Q_{ij}^{p_2}(q) = 0.25 \cdot 2 + 0.75 \cdot 1 = 1.25$. When departing at time 3.5, we arrive at time 4.75. Algorithm 1 shows this method more abstractly. In Figure 2.3 we show the impact this algorithm has on the travel times and arrival times in the example presented in Figure 2.2.

We define $\overrightarrow{t_{ij}}(q, d)$ as the travel time at departure time d between origin-destination pair i, j for cumulative probability q. This can be calculated using Algorithm 1. Similarly we define $\overleftarrow{t_{ij}}(q, a)$ as the travel time given arrival time a between origin-destination pair i, j for cumulative probability q. We can create an algorithm very similar to Algorithm 1, which will calculate the travel time given the arrival time at j. Figure 2.4 shows the three-dimensional plot of the travel time function $\overrightarrow{t_{ij}}(q, d)$ between Den Bosch and Zeist, with the departure time d on the x-axis, the travel time on the y-axis and the cumulative probability q on the z-axis.

2.1.2 Simulations

In our algorithm we will not use the stochastic travel time functions directly, instead we will create simulation worlds in which we realize a value from the

Procedure `CalculateTravelTime`(i,j,q,d)
  $\beta \leftarrow 1$
  $T \leftarrow 0$
  while $\beta > 0$ **do**
    $p \leftarrow$ time period $p \in P$ with $tp_p^o \leq d < tp_p^c$
    $\gamma \leftarrow \min(\beta, \dfrac{tp_p^c - d}{Q_{ij}^p(q)})$
    $T \leftarrow T + \gamma Q_{ij}^p(q)$
    $d \leftarrow tp_p^c$
    $\beta \leftarrow \beta - \gamma$
  end
  return T

Algorithm 1: Calculating the travel time between two nodes, given the departure time d. Variable β stores the part of the trip we still need to do and T stores the combined travel time.

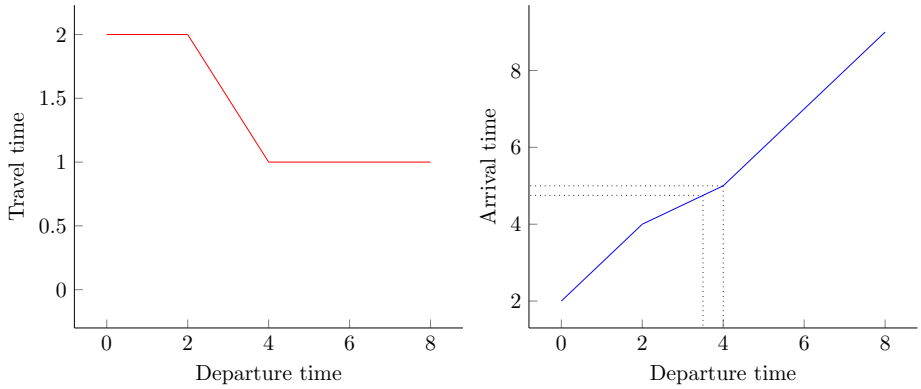

Figure 2.3: Using Algorithm 1 to calculate travel times for a given probability, we obey the FIFO assumption.

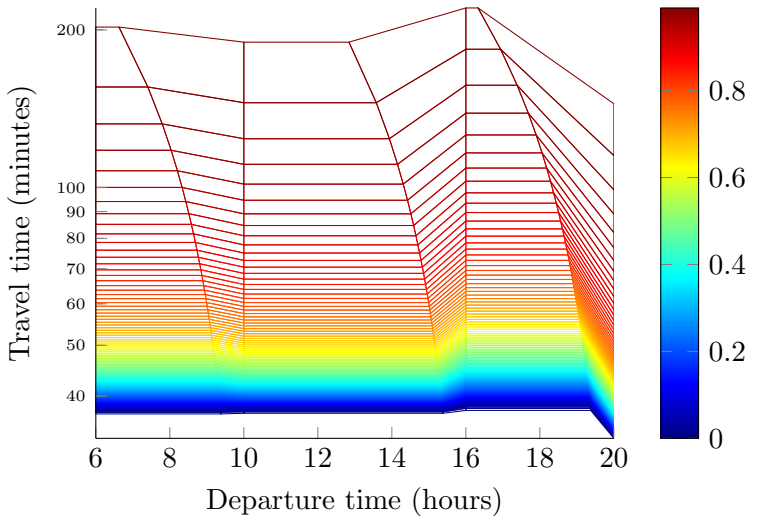

Figure 2.4: Three-dimensional plot of the travel time function between Den Bosch and Zeist for every cumulative probability 0.01, 0.02, etc.

travel time functions. The set S contains the simulation worlds we use. In a simulation world $s \in S$, we take a value q_{ij}^s from $X \sim U(0,1)$ for every node combination $i, j \in N$. The travel time between node i and j in simulation world s at departure time d is $\overrightarrow{t_{ij}^s}(d) = \overrightarrow{t_{ij}}(q_{ij}^s, d)$. Similarly, the travel time between these nodes in simulation world s given the arrival time a at node j is $\overleftarrow{t_{ij}^s}(a) = \overleftarrow{t_{ij}}(q_{ij}^s, a)$.

We approximate the reliability ry_c^r using these simulation worlds. Unless stated otherwise, the departure time from the depot is $-\infty$. We define the arrival time at the customers in a route using the following recursive function.

$$A_{r_i}^{rs} = \begin{cases} -\infty & \text{if } i = 0 \\ \max\{A_{r_{i-1}}^{rs} + s_{r_{i-1}} + \overrightarrow{t_{r_{i-1}r_i}^s}(A_{r_{i-1}}^{rs} + s_{r_{i-1}}), tw_{r_i}^o\} & \text{otherwise} \end{cases}$$

Now we can approximate the reliability of a customer by dividing the number of simulation worlds in which we arrive on time by the total number of simulation worlds. More formally,

$$ry_{r_i}^r \approx \frac{|\{s | s \in S \wedge A_{r_i}^{rs} \leq tw_{r_i}^c\}|}{|S|}$$

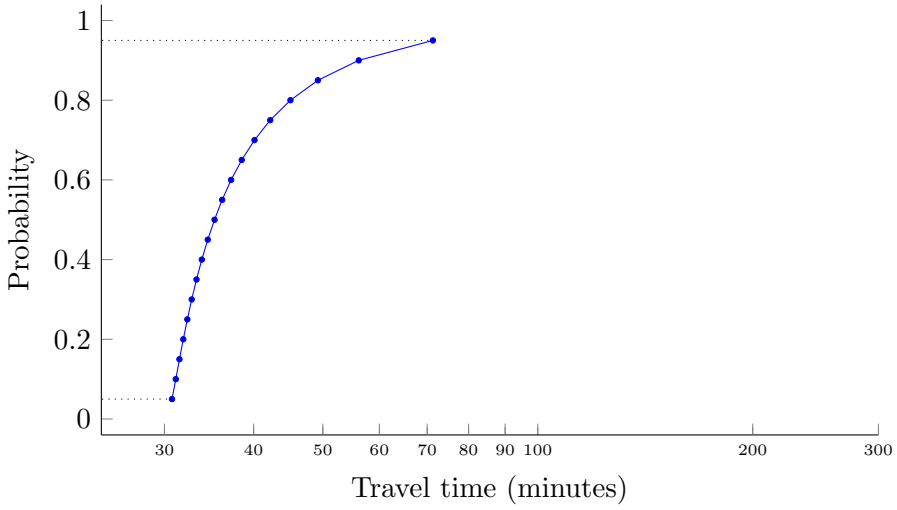

Figure 2.5: The piecewise-linear quantile function between Den Bosch and Zeist between 6AM and 10AM on logarithmic x-axis.

2.2 Data

Besides using distributions like the lognormal and gamma distributions, we can also create a new distribution using measurements. TomTom was kind enough to provide us with such distributions. However, only limited information is provided by them in the form of the average travel time and the travel times for the cumulative probabilities in the set {0.05, 0.1, 0.15, ..., 0.95}. We can use this information to create a piecewise-linear quantile function. In Figure 2.5 we show the piecewise-linear function between Den Bosch and Zeist between 6AM and 10AM.

When we calculate the expected value of the piecewise-linear quantile function we notice that this is not equal to the average value sent by TomTom. First of all this is due to the fact that we only obtained a compressed version of their dataset, but mostly it is because we miss data between cumulative probabilities [0, 0.05] and [0.95,1].

To counter this problem, we will compute values for cumulative probabilities 0 and 1, such that the average obtained by TomTom is equal to the expected value from the distribution. We obtained n data points $(t_i, q_i) \in Q$ from TomTom with cumulative probability q_i and maximum travel time t_i. We know that $q_i < q_{i+1}$ and therefore $t_i \leq t_{i+1}$, as a quantile function can never decrease. The expected value of the piecewise-linear quantile function

is equal to

$$E(Q) = \frac{1}{2}\sum_{i=1}^{n-1}(q_{i+1} - q_i)(t_i + t_{i+1})$$

In our case $q_{i+1} - q_i = 0.05$ for every $1 \leq i < n$.

We can use this formula to define the travel time for the last quantile, such that the average of the raw data is equal to the expected value of the quantile function. First we add two new points to the set; (t_0, q_0) and (t_{n+1}, q_{n+1}) with $q_0 = 0$ and $q_{n+1} = 1$. Then we define t_0 and t_{n+1} using the following formulas.

$$\begin{aligned}
\mu &= E(Q) \\
\mu &= \frac{1}{2}\sum_{i=0}^{n}(q_{i+1} - q_i)(t_i + t_{i+1}) \\
\mu - \frac{1}{2}\sum_{i=0}^{n-1}(q_{i+1} - q_i)(t_i + t_{i+1}) &= \frac{1}{2}(q_{n+1} - q_n)(t_n + t_{n+1}) \\
\frac{2\mu - \sum_{i=0}^{n-1}(q_{i+1} - q_i)(t_i + t_{i+1})}{q_{n+1} - q_n} &= t_n + t_{n+1} \\
\frac{2\mu - \sum_{i=0}^{n-1}(q_{i+1} - q_i)(t_i + t_{i+1})}{q_{n+1} - q_n} - t_n &= t_{n+1} \\
\frac{2\mu - \sum_{i=0}^{n-1}(q_{i+1} - q_i)(t_i + t_{i+1})}{1 - q_n} - t_n &= t_{n+1}
\end{aligned}$$

Similarly we find that the value for t_0

$$t_0 = \frac{2\mu - \sum_{i=1}^{n}(q_{i+1} - q_i)(t_i + t_{i+1})}{q_1} - t_1$$

Now we have a circular calculation for t_0 and t_{n+1}, as we need the value of t_0 to calculate t_{n+1} and vice versa. We choose to set $t_0 = t_1$ to find t_{n+1}, as this results in the function with the lowest variance. When applying this technique to the example of Figure 2.5, we get the piecewise-linear quantile function shown in Figure 2.6.

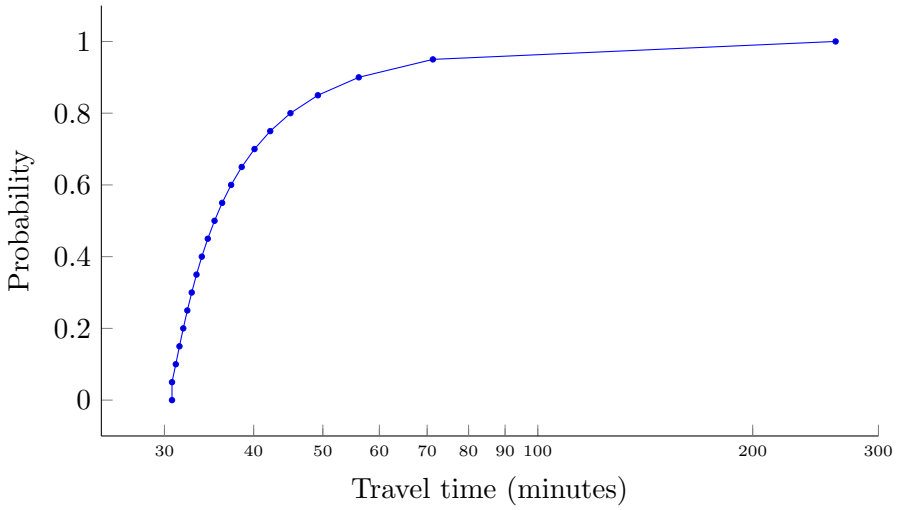

Figure 2.6: The piecewise-linear quantile function between Den Bosch and Zeist between 6AM and 10AM with computed values for cumulative probability 1, assuming $t_0 = t_1$ plotted on logarithmic x-axis.

Chapter 3

Column Generation

Toth and Vigo [21] compare three basic modeling approaches for the VRP, the vehicle flow formulation, the commodity flow formulation and the set-partitioning formulation. While the vehicle flow formulation and the commodity flow formulation use arcs to construct the optimal routes, the set-partitioning formulation takes the optimal routes from the set containing every feasible route. When ranked on the flexibility of the constraints and objective function, the set-partitioning formulation is the clear winner, as it can handle every constraint and objective function that is on the level of a single route. The only constraint our problem has that does not fit this description is that every customer should be visited once. Toth and Vigo show that this constraint can be incorporated in the set-partitioning formulation. The drawback of this formulation is that the number of variables grows exponentially to the number of customers. They propose using column generation to counter this.

Column generation is a commonly used technique for solving problems similar to ours [2, 5, 6, 14, 16, 17, 18]. The method is about generating columns, which are building blocks of the solution. The method divides a problem into two problems; the master problem and the pricing problem. In the master problem we assume that we have generated at least every column that should be in the optimal solution and we select them to create this solution. To make sure our assumption about the master problem holds, we relax the master problem to a linear problem. By solving the relaxed master problem, we obtain valuable information, allowing us to generate columns that should be in the solution. In the pricing problem we check if we have generated every column that should be in the solution. If we have not, we generate at least one and run the relaxed master problem again,

until we are unable to generate any more columns that improve the current solution using the pricing problem. Now we have every column we need to solve the relaxed master problem to optimality, but we may not have every column to solve the master problem to optimality. We can solve the master problem to optimality by using a technique called branch-and-price, which is a variant of branch-and-bound. With this technique we create branches using some branching rule that preferably does not complicate the solution of the pricing problem. By solving the relaxed master problem we can find the lower bound of a solution in a branch. We explore the branch with the lowest lower bound, as this one has the potential to contain the best solution. When we explore a branch, we use the pricing problem to find additional columns, while enforcing the result of the branching rule.

To solve our problem we decide to use column generation without branch-and-price, as the latter is often a performance bottleneck. This means we cannot guarantee to find an optimal solution for our problem and therefore our approach is a heuristic. We use routes as our columns, as they are the top level building blocks of our problem. First we will describe our master problem, then we define four methods to solve the pricing problem and finally we will explain how we solve our problem.

3.1 Master Problem

For every route we know its cost and which customers are visited. These attributes are formally defined as

$$c_r = \text{costs of route } r$$

$$a_{cr} = \begin{cases} 1 & \text{if route } r \text{ visits customer } c \\ 0 & \text{otherwise} \end{cases}$$

Given the set of all feasible routes $\mathcal{R}$, we can solve our problem using the following set partitioning based ILP formulation. Here we use the decision variables

$$x_r = \begin{cases} 1 & \text{if route } r \text{ is in the solution} \\ 0 & \text{otherwise} \end{cases}$$

$$\min \sum_{r \in \mathcal{R}} c_r x_r \tag{3.1a}$$

$$\text{s.t.}$$

$$\sum_{r \in \mathcal{R}} a_{cr} x_r = 1 \qquad \forall c \in C \tag{3.1b}$$

$$x_r \in \{0, 1\} \quad \forall r \in \mathcal{R} \tag{3.1c}$$

In 3.1a the objective function is defined. The goal is to minimize the combined costs of the routes in the solution. We impose that every customer should be visited once in 3.1b. A binary variable is defined for every route in 3.1c.

3.1.1 Number of possible routes

Assuming no feasibility restrictions on the routes, we can obtain every possible route by permuting every combination of customers. The amount of routes with $n = |C| \geq 1$ is equal to

$$\begin{aligned} |\mathcal{R}| &= \sum_{k=1}^{n} k! \binom{n}{k} \\ &= \sum_{k=1}^{n} \frac{n!}{(n-k)!} \\ &= n! \sum_{k=1}^{n} \frac{1}{(n-k)!} \\ &= n! \sum_{i=0}^{n-1} \frac{1}{i!} \end{aligned}$$

We may be able to generate every route for a problem with $n = 10$, which would mean $|\mathcal{R}| = 9864100$, but generating every route for a problem with $n = 15$ results in $|\mathcal{R}| = 3554627472075$. Generating every route for a problem with 15 customers is impractical, as we would need approximately 3.23TB storage space assuming we are able to store every route in a single byte.

$$\lim_{n \to 1} \sum_{i=0}^{n-1} \frac{1}{i!} = 1 \qquad \lim_{n \to \infty} \sum_{i=0}^{n-1} \frac{1}{i!} = e$$

Using these limits, we find the that the number of possible routes is bounded by $n! \leq |\mathcal{R}| \leq en!$. Every new customer might cripple the algorithm, as the j^{th} customer makes the route set grow by at least a factor j.

3.1.2 Relaxations

We can see that a solution only contains a small fraction of the total amount of generated routes. As every customer can only be serviced in one route, we can have at most n routes in a solution. This means an enormous amount of routes is generated that will not be in the solution, which is a waste of computational power. Therefore we should only create promising routes. We store these routes in $R \subseteq \mathcal{R}$. Problem 3.2 is our new Master Problem.

$$\min \sum_{r \in R} c_r x_r \tag{3.2a}$$

s.t.

$$\sum_{r \in R} a_{cr} x_r = 1 \qquad \forall c \in C \tag{3.2b}$$

$$x_r \in \{0, 1\} \quad \forall r \in R \tag{3.2c}$$

In Section 3 we will describe methods to create promising routes. By relaxing Problem 3.1c to $0 \leq x_r \leq 1$, we get a linear program. The linear program is rather restrictive. Assuming that for every route $r \in R$, its cost c_r is positive, we can relax 3.2b and 3.2c to 3.3b and 3.3c respectively. Since we try to minimize 3.2a, and a_{cr} is either 0 or 1, x_r will never be larger than 1. This will get us the final form of our Relaxed Master Problem

$$\min \sum_{r \in R} c_r x_r \tag{3.3a}$$

s.t.

$$\sum_{r \in R} a_{cr} x_r \geq 1 \quad \forall c \in C \tag{3.3b}$$

$$x_r \geq 0 \quad \forall r \in R \tag{3.3c}$$

3.1.3 Initialization

The Restricted Master Problem requires that every customer should be visited at least once. To be able to find a feasible solution, we should make sure that this constraint is met. Route set R is initialized as the set that contains $|C|$ routes; for every customer $c \in C$, there is a route $r \in R$ that services only customer c.

3.1.4 Adding new columns

Domination

When we add a route r to R, we may have another route $r' \in R$ in the master problem with the following properties:

- The costs of r are lower than the costs of r'
- Route r visits at least the same customers as r'

Given these properties, the master problem will always prefer r over r'; therefore r' will never be selected in the solution. We can remove route r', as it is redundant. More formally, we will remove every route $r' \in R$ if there is a route $r \in R$ with $c_{r'} \leq c_r \wedge r' \subseteq r$.

Managing columns

The number of columns may grow very fast and as we have seen in Section 3.1.1, creating every possible route is impractical. Therefore we set a maximum number of columns for the master problem. When we add columns, we first check for the domination criterion defined in Section 3.1.4, then we solve the relaxed master problem. Then we select the columns with a reduced cost larger than 0 which were not created in the initialization step described in Section 3.1.3 and we remove these columns in descending order of their reduced cost until the number of columns is equal to the maximum number of columns. This means we may solve the relaxed master problem with more columns than the maximum number of columns for the master problem, however, we chose to do so to obtain a better solution in the relaxed master problem.

3.2 Pricing problem

The Master Problem is a linear program. When we solve it, we get a dual value for every constraint. The dual values of Constraints 3.3c will always be 0, but the dual values of Constraints 3.3b may not. In Constraints 3.3b we have one constraint for every customer, so we get a dual value of π_c for every customer $c \in C$. We can use the dual values to compute the reduced cost $\overline{c}_r$ of a route r [17].

$$\overline{c}_r = c_r - \sum_{c \in C} a_{cr} \pi_c$$

From LP-theory we know that adding a route to the LP-relaxation can only decrease the current solution value if the reduced cost is negative. This means that we can use this formula as an objective to find routes that are worth adding to R. In the next sections we present four methods that use this formula to find promising routes. Their objective is to find at least one route r with $\bar{c}_r < 0$, to add to the master problem.

3.2.1 Mixed Integer Programming

Fasting et al. [6] formulate a MIP model to solve the VRPTW. In this section we will adapt their model to account for time-dependent stochastic travel times. To estimate the reliabilities we use simulations. We simulate every origin-destination pair a number of times. The set S contains all simulation worlds.

We found that combining both stochasticity and time dependency in the travel times is significantly more difficult than implementing one of these features. Therefore we will first present a model that solves the pricing problem without time dependency, taking the worst travel time from all time periods. Then we show the variables and constraints we need to add to the model to allow time dependency.

Formulation

A solution can be described by the order in which customers are serviced. Therefore we create a variable for every pair of nodes.

$$x_{ij} = \begin{cases} 1 & \text{if we go from customer } i \text{ to customer } j \\ 0 & \text{otherwise} \end{cases}$$

Besides this set of main decision variables, we have some additional decision variables that help create a feasible solution.

$$x_i = \begin{cases} 1 & \text{if we service customer } i \\ 0 & \text{otherwise} \end{cases}$$

$A_i^s =$ the arrival time at customer i in simulation s

$D_i^s =$ the departure time from customer i in simulation s

$$y_i^s = \begin{cases} 1 & \text{if we are on time for customer } i \text{ in simuation } s \\ 0 & \text{otherwise} \end{cases}$$

The benefit of servicing customer $c \in C$ is its dual value π_c. The travel time from node $i \in N$ to node $j \in N$ in simulation world $s \in S$ in time period $p \in P$ is represented in $t_{ij}^{sp} = Q_{ij}^{p}(q_{ij}^{s})$

We will discuss the formal MIP formulation 3.4. We try to find a route that minimizes the reduced cost, using Objective 3.4a. Constraints 3.4b, 3.4c, 3.4d and 3.4e make sure the time windows are satisfied. Together with the flow conservation Constraints 3.4f they ensure that a single route is created. Constraints 3.4c makes sure that the model takes the travel time between two nodes into account. The service time at a customer is ensured by Constraints 3.4d. The opening of the time window is maintained by Constraints 3.4e. Constraint 3.4g guarantees the route visits the depot and that we only create a single route. The variable x_i is set in Constraints 3.4h, so we can use this as a shorthand in other constraints. Constraints 3.4b allow the vehicle in some of the simulation worlds to arrive after the time window has closed, while Constraints 3.4i makes sure the minimum reliability of the customers are respected.

Some constraints are only used if a certain boolean decision variable is equal to 1. These constraints are modeled as indicator constraints. We present them as-is to the solver. We could also model it using the big-M formulation. A constraint $ax \leq b \quad \text{if } y = 1$ can be rewritten as $ax \leq b + (1 - y)M$, with M bigger than the upper bound of ax. The downside of the big-M formulation is that it can cause numerical instability of the model, as described by Klotz and Newman in 2013 [12].

Time dependency

To implement time dependency we create three additional decision variable sets. These variables will work together to implement time-dependent stochastic travel times.

$$u_i^{sp} = \begin{cases} 1 & \text{if the trip to customer } i \text{ in simulation } s \text{ is (partly) in period } p \\ 0 & \text{otherwise} \end{cases}$$

$p_i^{sp} =$ the ratio of the trip to customer i in simulation s that is in period p

$t_i^s =$ the travel time to customer i in simulation s

The travel time between two nodes is calculated using the time periods in which the travel takes place. The set of decision variables p_i^{sp} is the leading force behind the calculation, as it keeps track of how much of the trip takes place in a specific time period. They are aided by the set of decision variables u_i^{sp}, which can be defined as $u_i^{sp} = \lceil p_i^{sp} \rceil$.

$$\min \sum_{i,j \in N} c_{ij} x_{ij} - \sum_{i \in C} \pi_i x_i \tag{3.4a}$$

s.t.

$$D_i^s \leq tw_i^c + s_i \quad \forall i \in C, s \in S \quad \text{if } y_i^s = 1 \tag{3.4b}$$

$$D_i^s + \max_{p \in P} t_{ij}^{sp} \leq A_j^s \quad \forall i, j \in N, s \in S \quad \text{if } x_{ij} = 1 \tag{3.4c}$$

$$A_i^s + s_i x_i \leq D_i^s \quad \forall i \in C, s \in S \tag{3.4d}$$

$$(tw_i^o + s_i) x_i \leq D_i^s \quad \forall i \in C, s \in S \tag{3.4e}$$

$$\sum_{j \in N} x_{ij} = \sum_{j \in N} x_{ji} \quad \forall i \in N \tag{3.4f}$$

$$\sum_{j \in C} x_{\delta j} = 1 \tag{3.4g}$$

$$x_i = \sum_{j \in N} x_{ij} \quad \forall i \in N \tag{3.4h}$$

$$\frac{1}{|S|} \sum_{s \in S} y_i^s \geq m_i x_i \quad \forall i \in C \tag{3.4i}$$

$$x_{ij} \in \{0, 1\} \quad \forall i, j \in N \tag{3.4j}$$

$$A_i^s \geq 0 \quad \forall i \in N, s \in S \tag{3.4k}$$

$$D_i^s \geq 0 \quad \forall i \in N, s \in S \tag{3.4l}$$

$$y_i^s \in \{0, 1\} \quad \forall i \in N, s \in S \tag{3.4m}$$

Constraint 3.4c will be replaced by constraint 3.5a, as the travel time between two nodes is now time-dependent. The other constraints are needed to compute the travel time given the departure time.

$$D_i^s + t_j^s \leq A_j^s \qquad \forall i,j \in N, s \in S \quad \text{if } x_{ij} = 1 \tag{3.5a}$$

$$tp_p^o u_j^{sp} + t_{ij}^{sp} p_j^{sp} \leq A_j^s \qquad \forall i,j \in N, s \in S, p \in P \quad \text{if } x_{ij} = 1 \tag{3.5b}$$

$$D_i^s + t_{ij}^{sp} p_j^{sp} \leq tp_p^c \qquad \forall i,j \in N, s \in S, p \in P \quad \text{if } x_{ij} = 1 \wedge u_j^{sp} = 1 \tag{3.5c}$$

$$\sum_{p \in P} t_{ij}^{sp} p_j^{sp} = t_j^s \qquad \forall i,j \in N, s \in S \quad \text{if } x_{ij} = 1 \tag{3.5d}$$

$$p_i^{sp} \leq u_i^{sp} \qquad \forall i \in N, s \in S, p \in P \tag{3.5e}$$

$$\sum_{p \in P} p_i^{sp} = x_i \qquad \forall i \in N, s \in S \tag{3.5f}$$

$$0 \leq p_i^{sp} \leq 1 \qquad \forall i \in N, s \in S, p \in P \tag{3.5g}$$

$$u_i^{sp} \in \{0,1\} \quad \forall i \in N, s \in S, p \in P \tag{3.5h}$$

$$t_i^s \geq 0 \qquad \forall i \in N, s \in S \tag{3.5i}$$

Constraints 3.5f ensures that if we visit a customer, then we compute the complete travel time to this customer. Using Constraints 3.5e we set the binary decision variable that stores whether we travel in a given time period. The travel time to a node is set using Constraints 3.5d. Constraints 3.5c make sure that the model departs at the correct departure time, so we do not spend more time in the departure time period than possible. Constraints 3.5b guarantee that the arrival time at a node is at least equal to part of the travel time of the time period we arrive in combined with the opening of this time period.

3.2.2 Incremental Mixed Integer Programming

The MIP model shown in the previous section was very complex. Even without the time dependency, the model was quite tricky as the reliabilities had to be calculated using simulation worlds. Another approach we could take is to leave the simulations out of the model, and check the solution the model finds. If the model finds a solution that is not feasible, we change a few parameters of the model and re-execute it. We now present a more

simplistic model than the model in Section 3.2.1. First we define the decision variables.

$$x_{ij} = \begin{cases} 1 & \text{if we go from customer } i \text{ to customer } j \\ 0 & \text{otherwise} \end{cases}$$

$$x_i = \begin{cases} 1 & \text{if we service customer } i \\ 0 & \text{otherwise} \end{cases}$$

$$A_i = \text{the arrival time at customer } i$$

$$D_i = \text{the departure time from customer } i$$

Note that the arrival time decision variables A_i and departure time decision variables D_i do not use simulations, as they did in Section 3.2.1. The arc decision variables x_{ij} and x_i are exactly as in Section 3.2.1. Since we do not use simulations, we need to aggregate the simulated travel times into a single estimated travel time value. We estimate the travel time t_{ij} between nodes $i, j \in N$ using the simulations. We define the estimated travel time t_{ij} equal to the average simulated travel time. More formally, we can compute it using the probability q_{ij}^s in simulation world s as described in Section 2.1.2 in the quantile function $Q_{ij}^p(q)$ as described in Section 2.1. When taking the average travel time in all simulation worlds in all time periods we get

$$t_{ij} = \frac{\sum_{p \in P} \sum_{s \in S} Q_{ij}^p(q_{ij}^s)}{|P||S|}$$

The model is formally represented in MIP 3.6. This model is similar to MIP 3.4, without the time dependency. The major difference is that we aggregate the travel times from the simulation worlds in MIP 3.6, while in MIP 3.4 we use them individually. The constraints in this model are very similar to the constraints of Problem 3.4. We try to find a route that minimizes the reduced cost, using Objective 3.6a. The time windows are satisfied due to Constraints 3.6b, 3.6c, 3.6d and 3.6e. The travel time is taken into account due to Constraints 3.6b. Constraints 3.6c ensure that the service time at a customer is incorporated in the timing mechanism of the model. Constraints 3.6d enforce that the vehicle cannot service customers before their time windows open. Constraints 3.6e make sure the vehicle services the customers before their time window closes. The flow conservation Constraints 3.6f ensure that a single route is created. The route must visit the

$$\min \sum_{i,j \in N} c_{ij} x_{ij} - \sum_{i \in C} \pi_i x_i \tag{3.6a}$$

s.t.

$$D_i + t_{ij} \leq A_j \qquad \forall i, j \in N \quad \text{if } x_{ij} = 1 \tag{3.6b}$$

$$A_i + s_i x_i \leq D_i \qquad \forall i \in C \tag{3.6c}$$

$$(tw_i^o + s_i) x_i \leq D_i \qquad \forall i \in C \tag{3.6d}$$

$$D_i \leq tw_i^c + s_i \quad \forall i \in C \tag{3.6e}$$

$$\sum_{j \in N} x_{ij} = \sum_{j \in N} x_{ji} \qquad \forall i \in N \tag{3.6f}$$

$$\sum_{j \in C} x_{\delta j} = 1 \tag{3.6g}$$

$$x_i = \sum_{j \in N} x_{ij} \qquad \forall i \in N \tag{3.6h}$$

$$x_{ij} \in \{0, 1\} \qquad \forall i, j \in N \tag{3.6i}$$

$$A_i \geq 0 \qquad \forall i \in N \tag{3.6j}$$

$$D_i \geq 0 \qquad \forall i \in N \tag{3.6k}$$

depot, as is specified in Constraint 3.6g. In Constraints 3.6h the variable x_i is set, so we can use this as a shorthand in other constraints.

When we find a route r using this model, we will simulate it. Based on the results of the simulation, we determine the set $V \subseteq C$ containing all customers in r for which the reliability is lower than their minimum reliability. If $V = \emptyset$, we have successfully created a new route to add to the master problem. Otherwise we will add constraints so the model will evaluate r as infeasible. When we re-execute the model, it should find the next best route. One must pay attention when defining these banning constraints, as it should only ban infeasible routes, without the side effect of banning a feasible route. We propose the following methods, which we will test in Chapter 5. Remember we defined r_i as the i^{th} customer visited in route r.

Ban by time window

In stead of increasing the travel duration of an arc, we lower the time window at the violated customer for the given arc. We do so by adding the following constraints on the departure times at the violated customers.

$$D_{r_i} \leq D^*_{r_i} - \epsilon \quad \text{if } x_{r_{i-1}r_i} = 1 \qquad \forall r_i \in V$$

The scalar $D^*_{r_i}$ is the lowest possible departure time from customer r_i in the MIP, given the selected order of customers visited in r. We can calculate $D^*_{r_i}$ using the following recursive definition.

$$D^*_{r_i} = \max(D^*_{r_{i-1}} + t_{r_{i-1}r_i}, tw^o_{r_i}) + s_{r_i}$$

The scalar ϵ is a small time interval; a second would be sufficient. Using this new set of constraints we make all possible solutions with a departure D_{r_i} time equal or larger than $D^*_{r_i}$ infeasible if we go from node r_{i-1} to customer r_i. The impact this new set of constraints has on the solutions that can be found is minor, as it is only enforced if we go from node r_{i-1} to node r_i. This means that we will ban the infeasible solution, but we will most likely not ban any feasible solution.

Ban by route

We can also ban the entire route up until the first violation. To do so we take the first violation $r_m \in V$ and we add the following constraint to the MIP.

$$\sum_{i=0}^{m-1} x_{r_i r_{i+1}} \leq m - 1$$

Using this constraint we will only ban this route and every other route that starts with the same sequence of customers that results in a violation. With this banning method we are guaranteed that we do not ban any feasible routes. However, this comes at the cost of a constraint that uses many binary variables.

3.2.3 Local Search Heuristic

Conijn and Nuijten [3] use a local search approach to find a solution to the stochastic vehicle routing problem with hard time windows. While they use local search to solve the problem, we will use it to solve our pricing problem to create at least one route for the master problem.

We use the proprietary local search heuristic framework called POA, which is developed by Quintiq. POA is an abbreviation for Path Optimization Algorithm. We will first describe the operations we use to improve a given solution and then we will describe how we create the initial solution.

Improving the solution

If the reduced cost of the initial solution is larger than or equal to 0, we start improving the solution with the following operations.

construction The construction operation tries to improve the solution by taking up to 7 unplanned customers and adding them at a random position in the solution.

destruction The destruction operation tries to improve the solution by taking planned customers and removing them from the solution.

single-improvement The single improvement operation tries to improve the solution by taking a random customer and (re)planning it at the best possible position in the solution.

Initial solution

We try to find a good solution in little time. First we perform the I1 heuristic as described by Solomon [19]. This often results in a good starting solution. However, we will try to improve this solution, by using the single-improvement operation on every node as described in Section 3.2.3. If the reduced cost of this solution is lower than 0, we add this solution to the master problem. If not, we continue improving the solution as described in the Section 3.2.3.

3.2.4 Restricted Dynamic Programming

Held and Karp [10] create a dynamic programming algorithm to solve the TSP. As the algorithm enumerates all routes in a clever way, it can prove optimality of the solution it finds. Gromicho et al. [9] transform this algorithm in a heuristic framework for the VRP. Their framework can be adjusted to solve most of the VRP variants. Gromicho uses the Giant-tour representation, as introduced by Funke et al. [8], to represent multiple salespersons in the TSP. We will not need this representation as our objective is to create a single route.

We define the state (S, x) as the cheapest feasible route that visits all customers in S and ends at customer x. All states can be created using the following recursive definition.

$$G(S, x) = \min_{y \in S \backslash \{x\}} (G(S \backslash \{x\}, y) + c_{yx} - \pi_x) \tag{3.7}$$

$$G(\{x\}, x) = c_{\delta x} - \pi_x \tag{3.8}$$

$$H(S) = \min_{x \in S} (G(S, x) + c_{x\delta}) \tag{3.9}$$

In this definition $G(S, x)$ is equal to the feasible route with the lowest cost starting at the depot, visiting all customers in S and ending at customer x. The time window constraints are not modeled in the recursive definition, but implicitly specified by the fact that a state must be the cheapest feasible route; feasibility is checked using simulations when a new state is created. The function $H(S)$ computes the lowest cost of the feasible route that visits all customers in S and starts and ends at the depot. We can compute G using a bottom-up approach. We start the algorithm by setting $G(S, x) = \infty$ for all $S \subseteq C$ and $x \in S$. Then we create a state $(\{c\}, c)$ for every $c \in C$, with cost equal to $c_{\delta x} - \pi_x$ as defined in Formula 3.8. For every iteration after this one we take the states $\mathcal{S}$ we created in the previous iteration. We can now create the states $\{(S \cup \{y\}, y) | (S, x) \in \mathcal{S}, y \in C \backslash S\}$, using Formula 3.7. Only feasible states should be created. To check this we simulate every route we create a fixed number of times. If we exceed the minimum reliability at a customer we deem it infeasible and remove it.

With this approach we will create all routes using $O(n^2 2^n)$ time complexity and $O(n2^n)$ space complexity [10]. The method is too complex, so we will only expand the most promising states. For every iteration that is not the first one, we take the set $X \subseteq \mathcal{S}$ containing the m states with the lowest value in function G. Now we will only create the states $\{(S \cup \{y\}, y) | (S, x) \in X, y \in C \backslash S\}$, using Formula 3.7.

We will have exactly n iterations, as we add one customer to every state we expand in every iteration. In every iteration we expand m states with every customer that we can expand it with. This means we create at most $O(nm)$ states every iteration. Combining this information, we can see that we create at most $O(mn^2)$ states using this algorithm.

3.3 Integer transformation

The decision variables of the Master Problem do not necessarily obtain integer values. If $x_r = 0$ we do not take the route, if $x_r = 1$ we take it, but if $0 < x_r < 1$ it is undefined. To create a solution that can be used, we have to make sure that x_r is either 0 or 1.

Techniques like branch-and-price are developed to find the missing columns for the optimal integer solution. This technique requires a pricing problem method that can prove that we have found every column that we need. The only pricing problem method we defined that can do this is the mixed integer program from Section 3.2.1. As the other pricing problem methods are heuristics, we will not use branch-and-price to find the optimal integer solution; in stead we will heuristically create extra columns for Master Problem 3.2. We will perform these operations when we are unable to find any more routes with negative reduced cost.

3.3.1 Second rank columns

We alter the objective of the pricing problems from $\overline{c}_r < 0$ to $\overline{c}_r < x$, for some given value $x > 0$. This gives us second rank columns; routes with positive reduced cost which we may need to find a better solution to the master problem. We must pay attention to which columns we add, as we could find routes which we found before or create routes that are dominated by routes already in the solution. We will use the dominance criterion defined in Section 3.1.4 to filter out undesirable routes.

We must note that we expect to only find routes with reduced cost non-negative, but if the pricing problem method we have used to populate the master problem with the routes with negative reduced cost is a heuristic, we might find other routes with negative reduced cost using this method, as we have no guarantee that we have found every route with negative reduced cost.

3.3.2 Perturbations

We select the set S containing all routes in R with reduced cost equal to 0. For every route $r \in S$ we create $|r|$ new routes; each leaving out one customer from r. We take the set of new routes and we do this one more time. This means for every route in S we create new routes by removing every possible combination of 2 customers from it. We will use the dominance criterion defined in Section 3.1.4 to filter out undesirable routes.

Chapter 4

Departure time from depot

For a given route r, we would like to find the optimal departure time from the depot. We define D_r as the latest departure time from the depot in route r, such that for every customer $c \in r$ the reliability is equal to or larger than the target reliability m_c^r. Usually m_c^r is equal to the minimum reliability m_c. However, due to characteristics of a route, a reliability of at least m_c can be infeasible, independent of D_r. An example of such a characteristic is a route r that serves the two customers. If difference between the opening of the time window at customer r_1 and the closing of the time window of customer r_2 is small, it is likely that we will not be able to get a high reliability. This is due to the fact that the vehicle cannot depart earlier from customer r_1 than after the opening of its time window combined with its service time, independent of the departure time from the depot. If it is impossible to get a reliability of m_c at customer c in route r, we set m_c^r as the largest reliability of c possible. This value can be calculated by calculating the reliabilities assuming D_r is equal to $-\infty$.

In this chapter we will discuss three unique methods to calculate D_r. First we will introduce a generally applicable heuristic, using binary search. Then we define an exact method to compute the best D_r for the problem sketched. Finally we use a MIP to solve the problem.

4.1 Binary search

By using binary search, we can find D_r. Binary search is a method to find an element in an ordered set. We will use this approach to search the set of all possible departure times. However, binary search will not terminate on an infinite set, so we will add two types of boundaries to this set. The first

boundary we add is a horizontal limit; we need to specify a range to search in. The following range will do, as any earlier departure time will result in more waiting time at the first customer of r and any later departure time will definitely result in a reliability of 0 at every customer in r. Remember that we defined $\overleftarrow{t}^s_{ij}(a)$ in Section 2.1.1 as the travel time from node i to arrive at node j at time a in simulation world s.

$$[tw^o_{r_1} - \max_{s \in S} \overleftarrow{t}^s_{r_0 r_1}(tw^o_{r_1}), \max_{i \in r}\{tw^c_{r_i} - \min_{s \in S} \overleftarrow{t}^s_{r_{i-1} r_i}(tw^c_{r_i})\}]$$

The second boundary we add is a vertical limit; as we can divide time infinitely many times, we specify a precision. We stop our search if we are trying to improve the departure time by less than a second. This means that we can be off by a second, but this is fine as the driver will not likely depart on the exact second we calculate. The binary search algorithm is defined in Algorithm 2, with a and b as the range parameters and p as the precision parameter.

while $b - a > p$ **do**
  $x \leftarrow a + \frac{b-a}{2}$
  approximate reliabilities ry^r_c for all $c \in r$ using depot departure time x as we defined in Section 2.1.2
  if $\forall c \in r : m^r_c \leq ry^r_c$ **then**
    $a \leftarrow x$
  else
    $b \leftarrow x$
  end
end
return a

Algorithm 2: Binary search method to find optimal departure time from the depot, using range $[a, b]$ and precision p.

4.2 Backwards calculation

To calculate D_r, we use D_{rc}; the latest departure time from the depot, such that we have a reliability for every customer c in route r of at least m^r_c.

$$D_r = \min_{c \in r} D_{rc}$$

To find D_{rc} we have to work backwards, from the last customer we visit in r to the first customer. We work with simulations as the reliability at

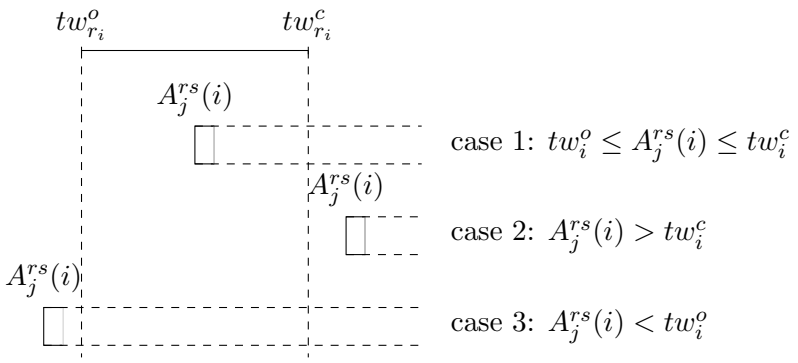

Figure 4.1: Three cases in the departure time calculation

a customer is also calculated using simulations. Let's define $D^{rs}_j(i)$ with $0 \leq i < j$ as the latest time we can depart from node r_i to be on time at customer r_j in simulation world s. Similarly we define $A^{rs}_j(i)$ with $0 \leq i \leq j$ as the latest time we have to arrive at customer r_i to be on time at customer r_j in simulation world s. We first define the relationship between these functions. The easiest case is the latest time we have to arrive at customer r_j to be on time at customer r_j in simulation world s, as this is equal to the end of the time window of customer r_j. The other latest arrival times can be determined by the latest departure times, as the smallest possible difference between the arrival time at customer i and departure time from customer i is the service time s_i. Therefore we can define $A^{rs}_j(i)$ using the following formula.

$$A^{rs}_j(i) = \begin{cases} tw^c_{r_j} & \text{if } i = j \\ D^{rs}_j(i) - s_i & \text{otherwise} \end{cases}$$

In Figure 4.1 we show the three cases that can happen when we calculate from the back. In case 1 the latest arrival time is within the time window. This means the latest departure time from customer r_{i-1} is equal to the latest arrival time at customer r_i minus the travel time from r_{i-1} to r_i. In case 2 the latest arrival time at customer r_i for which we can be on time at customer r_j is larger than the time window. We are optimizing to be on time for customer r_j, so being too late for customer r_i is irrelevant. Therefore we can perform the same step as in case 1; we subtract the travel time from r_{i-1} to r_i from the latest arrival time at r_i. In case 3 the latest arrival time at customer i is smaller than the time window. Since the vehicle will wait until $tw^o_{r_i}$ to serve customer r_i, we will be late for customer c in simulation world s

independent of depot departure time. We mark this simulation as infeasible. We will define this recursively; if the latest arrival time at customer r_i is after its time window opened, the latest departure time from customer r_{i-1} is equal to the arrival time at customer r_i minus the travel time from r_{i-1}. More formally defined we get the following formula for $D_j^{rs}(i)$.

$$D_j^{rs}(i) = \begin{cases} A_j^{rs}(i+1) - \overleftarrow{t^s_{r_i r_{i+1}}}(A_j^{rs}(i+1)) & \text{if } A_j^{rs}(i+1) \geq tw^o_{r_{i+1}} \\ \text{infeasible} & \text{otherwise} \end{cases}$$

Now we can compute D_{rr_j} by creating a list $L^r_{r_j}$ of size $|S|$, containing $D_j^{rs}(0)$ for every $s \in S$. We replace every element 'infeasible' mark by the lowest value in $L^r_{r_j}$. So for instance, if we have obtained the list [9AM, infeasible, 7AM, 8AM, infeasible], we transform it to [9AM, 7AM, 7AM, 8AM, 7AM]. Then D_{rr_j} is equal to the $\lceil m_{r_x} \cdot |S| \rceil^{\text{th}}$ highest value in $L^r_{r_j}$. As we defined at the start of this section, we find our answer using $D_r = \min_{c \in r} D_{rc}$

The method will calculate (part of) the route for every customer, for every simulation world. This means the algorithm will take $O(mn^2)$ with n equal to the amount of customers in the route and m equal to the number of simulation worlds used.

4.3 Mixed Integer Programming

Kok, Hans and Schutten [13] define a method to find the departure time from the depot such that duty time of the driver is minimized, given time-dependent travel times and driving regulations. They present a mixed integer program to solve their problem. Our problem is different from theirs, as we do not take driving regulations into account and our travel times are also stochastic. We will present a MIP that can solve our problem, but can also be easily adapted to find the best depot departure time for other objectives; like the lowest average travel time. First, we define our decision variables.

$$
\begin{aligned}
x &= \text{the departure time from the depot} \\
A_i^s &= \text{the arrival time at customer } i \text{ in simulation } s \\
v_i^s &= \begin{cases} 1 & \text{if we are too late at customer } i \text{ in simuation } s \\ 0 & \text{otherwise} \end{cases} \\
t_i^{sp} &= \text{the ratio of the travel time to customer } i \\
&\quad \text{that is performed in period } p, \text{ in simulation } s \\
u_i^{sp} &= \begin{cases} 1 & \text{if we travel to customer } i \text{ in period } p, \text{ in simuation } s \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

The objective is to maximize the departure time from the depot. Constraints 4.1b, 4.1c, 4.1d and 4.1e model the arrival time at a customer. Constraints 4.1b and 4.1c make sure the arrival time at a customer is later than the departure time of its predecessor, combined with the travel time between the two nodes. In Constraints 4.1b the predecessor is the depot. Constraints 4.1d make sure that the part we travel in a certain time period, is only in that time period. Constraints 4.1e make sure that adjacent time periods are selected, by stating that the arrival at a customer must be after the time period in which we travel. Constraints 4.1f records violations in a given simulation world, by setting v_i^s to 1 if the arrival at i is after the time window has closed. Constraints 4.1g make sure that if u_i^{sp} is non zero, that u_i^{sp} is also. Finally Constraints 4.1i limit the amount of reliability violations we allow, to obtain the target reliability.

$$\max x \tag{4.1a}$$

$$x + \sum_{p \in P} c^{sp}_{\to 1} t^{sp}_1 \leq A^s_1 \qquad \forall s \in S \tag{4.1b}$$

$$A^s_{i-1} + s_{i-1} + \sum_{p \in P} c^{sp}_{\to i} t^{sp}_i \leq A^s_i \qquad \forall i \in r, s \in S \tag{4.1c}$$

$$A^s_{i-1} + s_{i-1} + c^{sp}_{\to i} t^{sp}_i \leq tp^c_p \qquad \forall i \in r, s \in S, p \in P \quad \text{if } u^{sp}_i = 1 \tag{4.1d}$$

$$tp^o_p u^{sp}_i + c^{sp}_{\to i} t^{sp}_i \leq A^s_i \qquad \forall i \in r, s \in S, p \in P \tag{4.1e}$$

$$A^s_i \leq tw^c_{r_i} \qquad \forall i \in r, s \in S \quad \text{if } v^s_i = 0 \tag{4.1f}$$

$$t^{sp}_i \leq u^{sp}_i \qquad \forall i \in r, s \in S, p \in P \tag{4.1g}$$

$$\sum_{p \in P} t^{sp}_i = 1 \qquad \forall i \in r, s \in S \tag{4.1h}$$

$$(1 - m^r_{r_i})|S| \geq \sum_{s \in S} v^s_i \quad \forall i \in r \tag{4.1i}$$

$$x \geq 0 \tag{4.1j}$$

$$A^s_i \geq tw^o_{r_i} \qquad \forall i \in r, s \in S \tag{4.1k}$$

$$t^{sp}_i \geq 0 \qquad \forall i \in r, s \in S, p \in P \tag{4.1l}$$

$$u^{sp}_i \in \{0, 1\} \quad \forall i \in r, s \in S, p \in P \tag{4.1m}$$

$$v^s_i \in \{0, 1\} \quad \forall i \in r, s \in S \tag{4.1n}$$

Chapter 5

Experiments

To analyze our methods, we perform multiple computational experiments. First we will test the banning methods of the incremental mixed pricing problem method. Then we will compare our pricing problem methods to one another. We will use abbreviations for the pricing problem methods, which can be found in Table 5.1. Finally the methods to determine the departure time from the depot will be compared to one another.

To the best of our knowledge, no benchmark for the vehicle routing problem with time windows and time-dependent stochastic travel times is defined in literature. Therefore we test our algorithms on real data obtained from the company Veldhuizen. The data we obtained from Veldhuizen are the costs per distance, duration and vehicle usage and the orders for two days; 18 September 2015 and 12 January 2016. From now on we will use V2015 to refer to the dataset of 18 September 2015 and V2016 for the dataset of 12 January 2016.

To analyze the impact of the problem size on our algorithms, we created 4 subsets, increasing in size, of the customers for both datasets. We select the customers for every subset by computing the optimal solution for the VRPTW, using the state-of-the-art VRPTW solver from Quintiq, and then

Abbreviation	Pricing problem method	Section
MIP	Mixed Integer Programming	3.2.1
IMIP	Incremental Mixed Integer Programming	3.2.2
LS	Local Search Heuristic	3.2.3
DP	Restricted Dynamic Programming	3.2.4

Table 5.1: Pricing problem method abbreviations

Dataset	Tiny (T)	Small (S)	Medium (M)	Large (L)
V2015	12	26	63	102
V2016	10	27	51	108

Table 5.2: Number of customers in Veldhuizen datasets

Name	Start (tw_p^o)	End (tw_p^c)	μ	L
Night	Midnight	7AM	0.84	0.60
Morning rush	7AM	9AM	1.10	0.71
Midday	9AM	4PM	1.03	0.69
Evening rush	4PM	6PM	1.16	0.70
Evening	6PM	Midnight	0.88	0.62

Table 5.3: Time periods used in experiments

selecting routes that were close to one another, so many combinations are possible. Table 5.2 shows the exact number of customers in the different subsets. When we refer to dataset V2015/M, we mean the Medium sized subset of the dataset from 18 September 2015. The default minimum reliability is set to 0.95 for every customer. We use 100 simulation worlds to check feasibility on every route and we use translated lognormal distributions for our stochastic travel times. The distribution between two nodes $i, j \in N$ in time period $p \in P$ can be described using the average travel time A_{ij}, mean μ, the minimum travel time L and the variance σ^2, as $T_{ij}^p \sim A(\ln \mathcal{N}(\mu - L, \sigma^2) + L)$. The variance is a value between 0.5 and 2.1, depending on the following factors

1. The travel duration, as short trips have more variance than large trips.
2. The start and end location, as trips in urban areas like the Randstad in The Netherlands are more prone to traffic jams.
3. Randomness, as other unknown factors also play a role, some degree of randomness is added.

The mean and the minimum travel time depend on the time period. In Table 5.3 we show the 5 time periods we define, accompanied by the mean and minimum travel time values. These values were determined after analyzing time-dependent stochastic travel data from TomTom. For our costs we use 0.5 per kilometer, 50 per average hour traveling and 1000 per vehicle we use. While the costs per kilometer and the costs per hour are somewhat realistic,

the costs per vehicle are not. We chose to use such a large number to also minimize the number of vehicles used.

Using the state-of-the-art VRPTW solver from Quintiq, we solved both V2015/M and V2016/M. As this solver does not take time-dependent stochastic travel times into account, we use the average travel times. This resulted in solutions that cost 7442.27 and 6277.34 respectively, while not meeting the minimum reliability of 95% at every customer.

Every experiment in this chapter has been performed on a computer with an Intel(R) Core(TM) i7-2720QM CPU at 2.20GHz and 16GB RAM. The mathematical programs are solved using CPLEX 12.6.3 from IMB.

5.1 Banning methods

To test the banning methods for the IMIP pricing problem method defined in Section 3.2.2 we use the datasets from Veldhuizen. We use the Tiny, Small and Medium datasets from V2015 and V2016, as we were unable to find a new column in one hour using either method for the Large datasets. We stop our column generation solution if we are unable to find a feasible column within 3 minutes. In this section we will not transform the solution we found to an integer solution. So the data in this section is only about the relaxed master problem.

In Figure 5.1 we can see that banning by time window gives a better result in the experiments than banning by route. First of all, banning by route adds a more difficult constraint, as we combine multiple binary variables, while we only use one binary variable when banning by time window. Furthermore, when banning by time window we may ban too strictly, banning parts of routes that we need in our solution, but this helps us shrink our search space faster. In Table A.1 the results of this experiment are shown.

5.2 Pricing problems

In this section we will test the different methods to solve the pricing problem we defined in Section 3.2. Performing some simple tests we have noticed that both the MIP and IMIP methods perform extremely poor on even the smaller experiments. Therefore we stopped experimenting with these methods. We have not been able to optimize the DP algorithm for speed, therefore it is very slow compared to the LS method. We will use the Medium sized datasets to investigate the effects of different parameters in the problem. Remember that we have the relaxed master problem and the

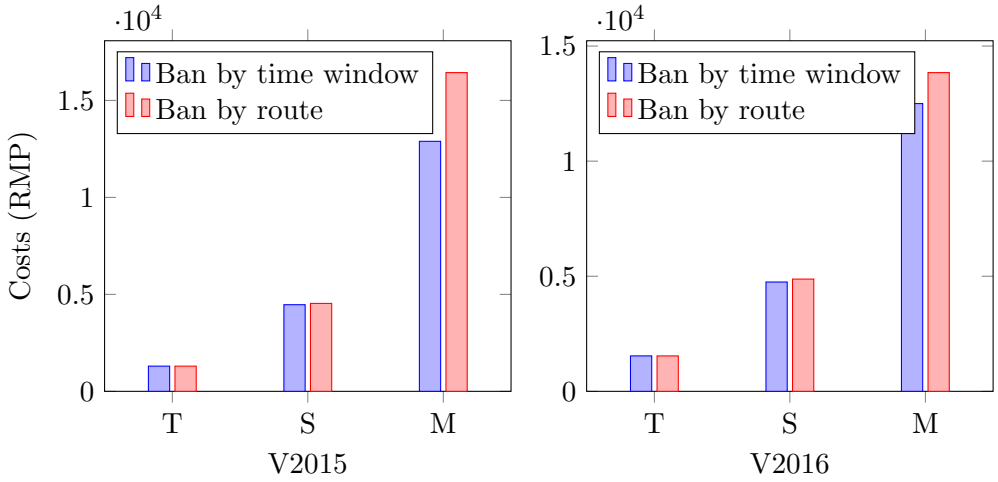

Figure 5.1: Effect of banning methods on solution value.

master problem. As the costs in these problems differ, we will report the costs of the relaxed master problem as Costs (RMP), while the costs of the master problem will be reported as Costs.

5.2.1 Settings

First we will perform some experiments to find optimal settings for our problem. In Section 3.1.4 we defined a column management method that limits the number of columns in the master problem. However, we need to find the best value for the maximum number of columns we allow in the master problem. We test this method using DP, as it generates many routes with negative reduced cost every iteration. The DP pricing problem method is allowed to expand up to 8 routes every DP iteration, but if it is unable to find a route, we double this value. For dataset V2015/M we try the parameter with 10000, 5000 and 2500 columns. In Figure 5.2 we see the results of this experiment; using a maximum of 2500 columns is enough to get the best possible value in 6 hours of DP execution. Therefore in all future experiments we set this value to 2500 columns, meaning that if we have more than 2500 columns, we will remove the columns with highest reduced cost until we have 2500 columns remaining.

As described in the previous experiment, the DP pricing problem method is allowed to double its search space if it cannot find any columns. We found that extending the search space to allow the DP to extend 16384

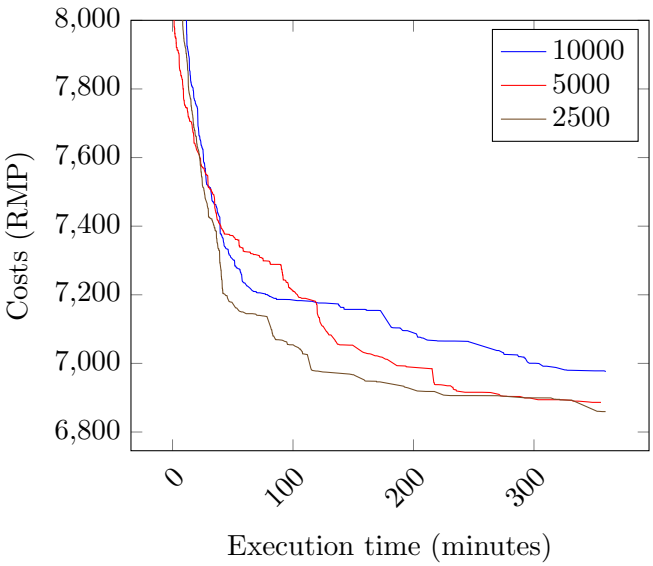

Figure 5.2: Three DP executions with varying maximum number of columns on dataset V2015/M.

partial routes every iteration resulted in memory issues. In Figure 5.3 we cannot detect the convergence of the DP pricing problem method at a given maximum number of routes it is allowed to expand. Investigating the matter further, we can see in Figure 5.4 that the DP pricing problem method is not able to converge with a search space of only 8192. So due to memory issues we are unable to find the best trade-off between the maximum number of routes to expand and the solution value. Therefore we will limit the search space to 8192.

As we limit the DP, we will also limit the time the LS is allowed to find at least one new route. In Figure 5.5 we show the increase in solution value using different time limits. We allow the LS to run up to 15 minutes and we can see in the figure that with a time limit of 3 minutes we are able to find a solution 2% worse than the best solution for both datasets. However, in both cases we were able to find the 2% worse solution in one-third of the time it took to find the best solution. Therefore we set a limit of 3 minutes to the search duration of the LS pricing problem method.

As described in Section 3.3.1, we create second rank routes as they may be useful in the set partitioning step of the master problem. These routes have a positive reduced cost up to a certain threshold. By gradually increas-

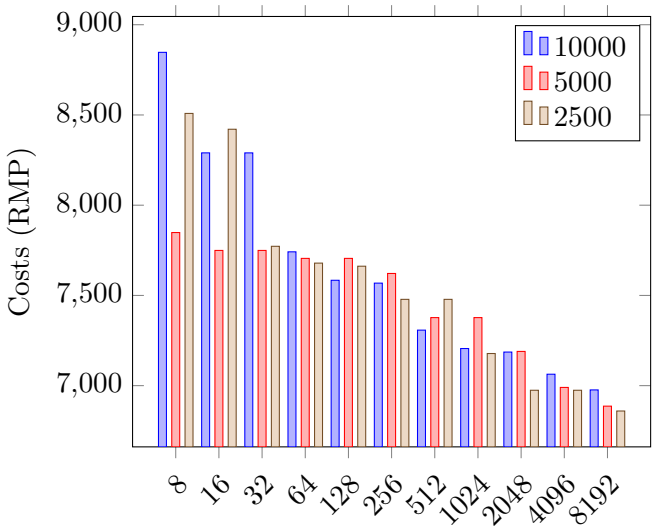

Figure 5.3: The best value we get from the DP execution on dataset V2015/M for different values for the maximum number of routes to expand per DP iteration.

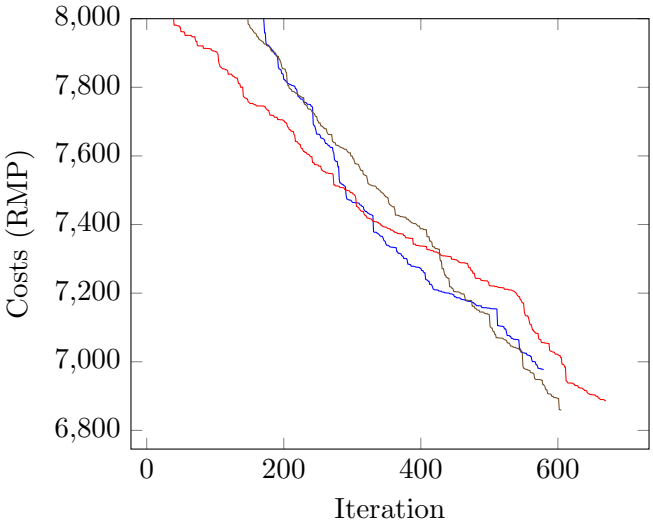

Figure 5.4: No convergence for DP executions on dataset V2015/M with maximum search space 8192.

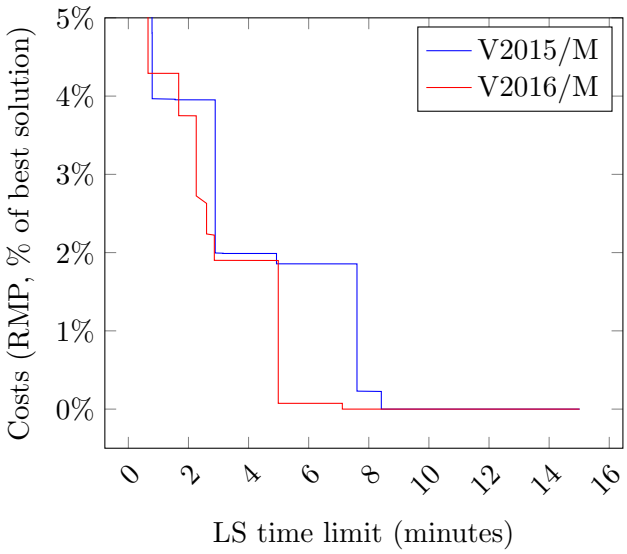

Figure 5.5: Best solution found against maximum duration to find at least one route by LS on Medium datasets.

ing the threshold, we will try to determine the optimal threshold for our problem using the LS method. We start the threshold at 1 and every time we are unable to find a new route within 3 minutes, we increase the threshold by 1. In Figure 5.6 we see that around threshold 40 both datasets are improved. Unfortunately, we are unable to detect a pattern, so we decide to set the threshold to 75, as every improvement we have found is covered when using this value. For the DP method, second rank columns did not improve the solution at all. This is probably due to the large number of columns the DP method creates, as the useful second rank columns will probably already be included in the set of routes.

5.2.2 Results

First we would like to see the impact of the number of simulation worlds on our pricing problem methods. We test the methods with a number of simulation worlds equal to 100, 50 and 20. To make a fair comparison, we first generate 100 simulation worlds and then we extract the first 50 and the first 20 simulation worlds from this instance. In Figure 5.7 we can see that the execution time of both LS and DP are greatly influenced by the number of simulation worlds. Fewer simulation worlds mean faster computation, but

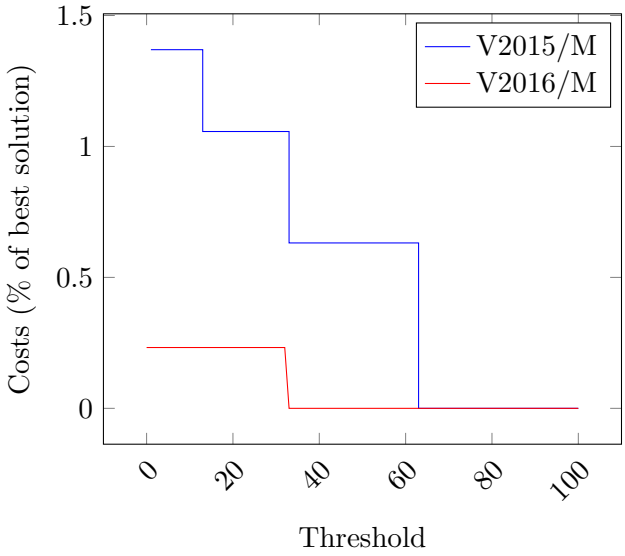

Figure 5.6: Percentage improvement obtained by adding second rank columns using LP method.

it also means lower accuracy of the time-dependent stochastic travel times. This means that by using fewer simulation worlds, there is a higher probability that we create routes which would not meet the minimum reliability criteria if we had used more simulation worlds. In Table A.2 the raw data of this experiment can be found.

Next we will test the impact of the minimum reliability of the customers. We set the minimum reliability to 75%, 90%, 95% and 99% for every customer. We were unable to detect a relation between the time it takes to create a new route and the minimum reliability for both methods. Obviously as the minimum reliability increases, we have fewer feasible routes in the problem and by increasing the minimum reliability the costs for the best solution found increases. In Table A.3 the raw data of this experiment can be found.

The sizes of the time windows may also influence the execution time and the final solution value significantly. With average time window sizes of 6 hours and 27 minutes for V2015/M and 7 hours and 17 minutes for V2016/M, it is not very hard to create routes visiting many customers, while meeting the minimum reliability constraint at every customer. To make it a bit harder we shrink the time windows of every customer to sizes of 4 hours, 2 hours, 1 hour, 30 minutes and 10 minutes. In Figure 5.8 we see that the

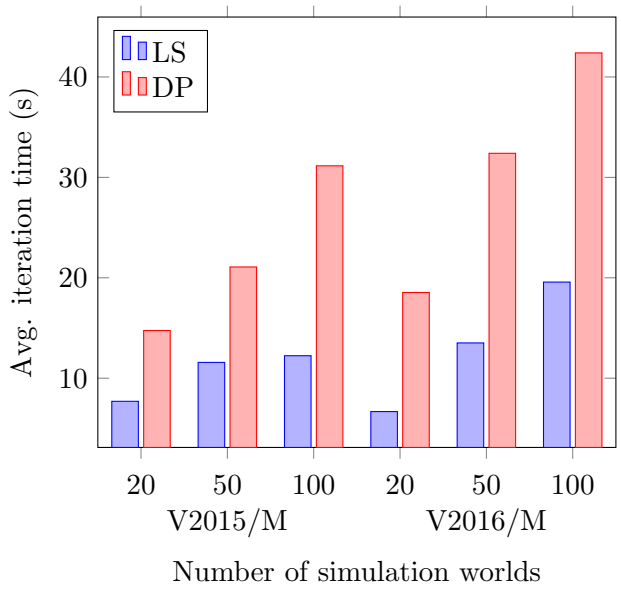

Figure 5.7: Impact of the number of simulation worlds on the execution time of pricing problem methods.

average time window size influences the time it takes to create new routes for the LS method. We do not find such pattern for the DP method. Figure 5.9 shows that the total execution time for both methods is influenced by the average time window size. This can be explained as the number of feasible routes decrease when the average time window size decreases; therefore the search space is smaller and the necessary search time shorter. In Table A.4 the raw data of this experiment can be found.

Next we experiment with the number of customers in our problem. We use the datasets as described in Table 5.2. In Figure 5.10 we see that the time it takes both methods to find their solution increases as the number of customers grows. This can be explained as the search space grows significantly by adding extra customers, as explained in Section 3.1.1. In Table A.5 the raw data of this experiment can be found.

Using the previously described experiments, we found that the perturbation operation to create additional columns as described in Section 3.3.2 improved the solution value on average by 0.9% for the LS method and 1.0% for the DP method.

Comparing the LS to the DP method we see that LS is the clear winner. In almost every experiment it was able to find the best solution in the

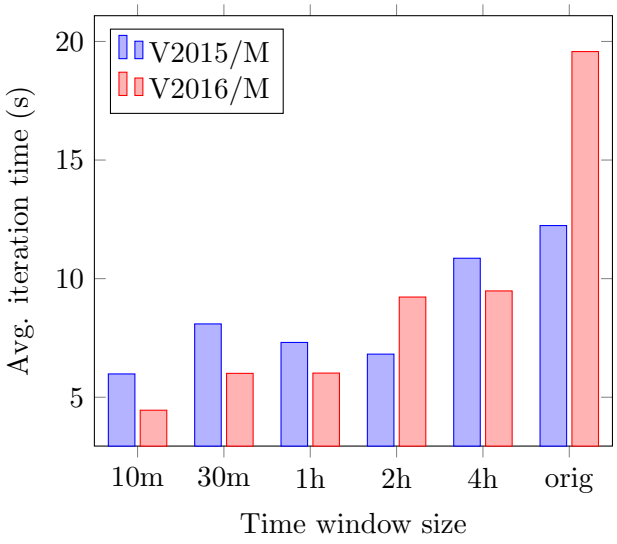

Figure 5.8: The effect of the time window size on the average time it takes LS to create a route.

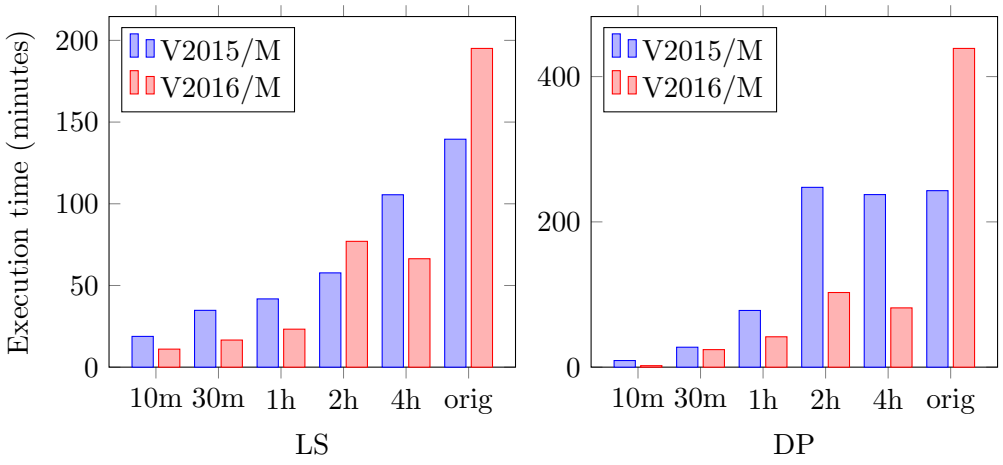

Figure 5.9: The effect of the time window size on the time it takes to find the solution.

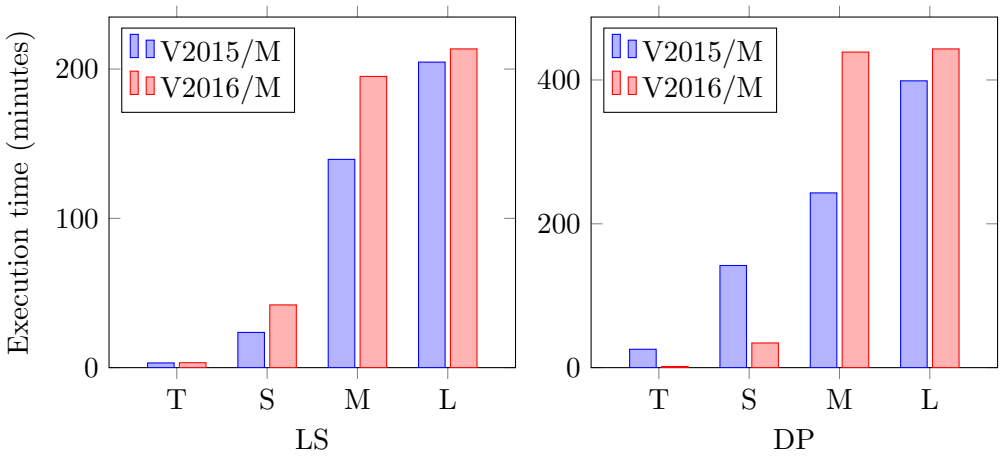

Figure 5.10: The effect of the number of customers on the time it takes to find the solution.

shortest time. This can be explained as the DP method can only improve a partial route by appending a customer, while the LS method improves a route by inserting and deleting customers at any position. While the LS method can exit a local optimum by restarting or exploring unfavorable neighborhoods, the DP method cannot and therefore it is beaten by the LS method. Checking the feasibility of a route is the main bottleneck of the DP method, as this requires many arrival time calculations per route and makes is slow.

To determine the quality of our method, we have to compare our results to another method. To the best of our knowledge no other method has been created to solve our problem. Therefore we will compare it to the state-of-the-art VRPTW solver by Quintiq to solve our problem as a VRPTW. Doing so, we lose the time-dependent stochastic nature of our travel times. To compensate for this, we add a bit of slack to the travel times; for example, in stead of using the average travel time, we use 1.1 times the average travel time. In Figure 5.11 we see the results of this experiment. Adding slack to a VRPTW solver results in a more reliable schedule. However, our method is able to find solutions with a guaranteed minimum reliability at every customer, while not sacrificing a lot of quality.

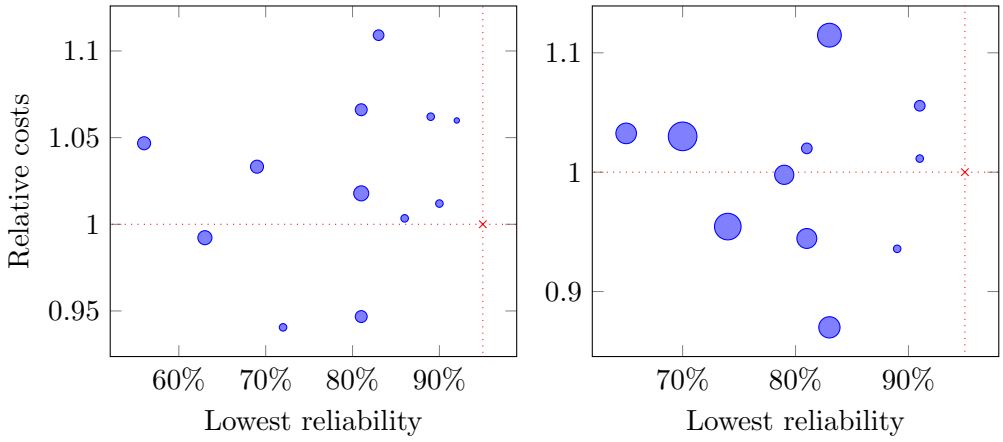

Figure 5.11: Comparing our column generation method to the obvious workaround of adding slack to the VRPTW solver. Our solution is marked by the red cross and the blue dots represent the solutions of the workaround. The size of a dot represents the number of customers we are unreliable at in the solution. Left we see V2015/M and V2016/M and right we see V2015/M and V2016/M with time window sizes of 2 hours.

5.3 Departure time from depot

In this section, we will test the methods to compute the best departure time from the depot we defined in Chapter 4. As these methods are searching for the optimal departure time, we are only interested in the time it takes them to find it. First we will explain our setup and then we will show our results.

We use 5 different routes to test the methods we defined in Chapter 4. In Table 5.4 the number of customers, the average travel duration and the travel distance of these routes are shown. These routes are created using the datasets from Veldhuizen. The customers are selected such that the best possible reliability is equal to 1 for every customer that is visited. For every route we will also create similar routes, with different time windows; we take the average arrival time at a customer and we change its time window around this time. We create routes with 24 hour time windows for every customer, but also 10 and 30 minute time windows. Finally we also take a random number between 0 and 3 hours. For instance, if we have an average arrival time at 2PM, we create time windows of [2AM, 2AM (next day)], [1:55PM, 2:05PM], [1:45PM,2:15PM] and something between [2PM, 2PM] and [12:30PM, 3:30PM]. In Figure A.1 the best possible reliabilities for the

name	# customers	avg. duration (hours)	distance (km)
XS	8	8.06	444.842
S	15	8.5	414.348
M	21	6.34	256.216
L	27	3.86	130.733
XL	34	5.68	300.366

Table 5.4: Properties of the experimentation routes

customers in these routes are shown.

In Table A.6 the results from the binary search method are shown. We can see that the number of simulation worlds plays a role in the execution time of this method. The method is less affected by the problem size, as we see that some smaller problems take longer to solve than larger ones. In Table A.7 the results from the backwards calculation method are shown. Both the problem size and the number of simulation worlds affect the execution time of this method. The actual makeup of reliability and time windows do not affect this method at all. In Table A.8 the results from the mixed integer method are shown. First of all, some results could not be found in 1 minute using this method. The method struggles with smaller minimum reliability values. This is due to the implementation of time-dependent travel times in the MIP.

Comparing all three methods, we see that the backwards calculation method is the clear winner. It scores best on every test instance. However, the method is quite inflexible, as it is hard to change its objective or add other constraints, like driving regulations. Both the binary search method and the mixed integer program are more versatile in that respect. While we can express most constraints and objectives in the mixed integer program, we may be unable to do so in the binary search method, as this method requires that the search space is convex; given two departure times $d_i > d_j$, there should not be fewer time window violations for d_i than for d_j. However, this method can be used as a heuristic if the search space is non-convex.

Chapter 6

Conclusion

In conclusion, we created an algorithm that solves the vehicle routing problem with stochastic and time dependent travel times. The algorithm uses column generation and we explored multiple methods to solve the pricing problem. Comparing them, we found that the MIP methods were not up to the task, while the local search and the dynamic programming heuristic were able to find good solutions. The local search method proves to be superior over the dynamic programming heuristic, as its solutions are better and can be found faster.

We investigated multiple methods to ban routes in a simple MIP method and we found that the one that bans faster is able to obtain better results.

We compared three methods to find the optimal departure time from the depot. We found that the algorithm that would be hardest to extend performed the best and the method that would be easiest to extend performed the worst.

Altogether we found that the MIP formulations that allowed some violations in time windows did not perform well. The solver had problems finding the right combination of violations in the problem, as they are modeled using binary variables.

6.1 Future research

We can make the problem more generally practical by transforming it to a pickup and delivery problem with time-dependent and stochastic travel times. The pricing problem methods defined in this thesis should all work on this problem, except for the restricted dynamic programming one. In the PDP we need both the pickup and delivery action planned and the

restricted dynamic programming method extends partial routes. Therefore the method needs to "guess" whether a delivery can be done, when we plan a pickup.

Another realistic extension to the problem is the enforcement of real driving regulations in the problem, like mandatory breaks in the schedule. Due to the time-dependent nature of the problem it matters at which time a break is scheduled; a rest during rush hour is preferable. The local search method will be best suited for this extension, as we can use a heuristic to find a good time to plan a break. Adding these driving regulations to the depot departure time algorithms is hard, as they will need to find the best time to schedule breaks. While we will be able to add them to the MIP, as is done by Kok, Hans and Schutten in [13], more research needs to be done to be able to add it to the backwards calculation algorithm. The binary search method may be a good heuristic to find a reasonable solution.

In our problem, we do not stick to the triangle inequality. This means that in our problem it can be that traveling from A to B to C takes less time than traveling directly from A to C, even when the departure times are equal. A similar problem is one with two customers A and A' which are close to one another; if we travel from A to B and from A' to B using the same departure time, the arrival times at B may be significantly different. Obviously this is wrong, but this is an artifact of random sampling of the travel time distributions. We may be able to solve this by using a smarter technique to sample from the distributions. Another way to solve this is by using real data; take one historical day and use the traffic data of this day in one simulation world.

Finally, we use simulation worlds in our problem. The biggest disadvantage of this is that there is a correlation between precision and performance; more simulation worlds means more precision, but worse performance. If we would be able to calculate with the time-dependent stochastic travel times more easily, we may be able to abandon the simulation worlds altogether. A promising direction would be to alter the discrete stochastic travel times as defined by Fasting [6], to allow for time-dependency.

Bibliography

[1] T. Bektas. The multiple traveling salesman problem: an overview of formulations and solution procedures. *Omega*, 34(3):209 – 219, 2006.

[2] A. Ceselli, G. Righini, and M. Salani. A column generation algorithm for a vehicle routing problem with economies of scale and additional constraints. *Transportation Science*, 43(1):56–69, 2009.

[3] B.J. Conijn and W.P.M. Nuijten. Modeling and solving the vehicle routing problem with stochastic travel times and hard time windows. *Eindhoven University of Technology, Department of Mathematics and Computer Science*, 2013.

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

[5] I. Dayarian, T.G. Crainic, M. Gendreau, and W. Rei. A column generation approach for a multi-attribute vehicle routing problem. *European Journal of Operational Research*, 241(3):888 – 906, 2015.

[6] L.A. Fasting, M. Firat, M.A.A. Boon, and J. Twist. Branch-and-price approach to the vehicle routing problem with hard time windows and stochastic travel times. Master's thesis, Eindhoven University of Technology, 2014.

[7] M.L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

[8] B. Funke, T. Grünert, and S. Irnich. Local search for vehicle routing and scheduling problems: Review and conceptual integration. *Journal of Heuristics*, 11(4):267–306, 2005.

[9] J. Gromicho, J.J. van Hoorn, A.L. Kok, and J.M.J. Schutten. Restricted dynamic programming: a flexible framework for solving realistic vrps, 2009.

[10] M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962.

[11] S. Ichoua, M. Gendreau, and J.-Y. Potvin. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research*, 144(2):379 – 396, 2003.

[12] E. Klotz and A.M. Newman. Practical guidelines for solving difficult mixed integer linear programs. *Surveys in Operations Research and Management Science*, 18(12):18 – 32, 2013.

[13] A.L. Kok, E.W. Hans, and J.M.J. Schutten. Optimizing departure times in vehicle routes. *European Journal of Operational Research*, 210(3):579 – 587, 2011.

[14] D. Koning, J.M. van den Akker, and J.A. Hoogeveen. Using column generation for the pickup and delivery problem with disturbances. Master's thesis, Utrecht University, June 2011.

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

[16] L. Rousseau, M. Gendreau, and G. Pesant. Solving vrptws with constraint programming based column generation. *Annals OR*, 2004.

[17] D. Simchi-Levi, X. Chen, and J. Bramel. *The Logic of Logistics: Theory, Algorithms, and Applications for Logistics and Supply Chain Management*, chapter 16, pages 275–291. Springer New York, New York, NY, 2005.

[18] M. Sol. *Column Generation Techniques for Pickup and Delivery Problems*. Wibro, 1994.

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

[20] W.R. Stewart and B.L. Golden. Stochastic vehicle routing: A comprehensive approach. *European Journal of Operational Research*, 14(4):371 – 385, 1983.

[21] P. Toth and D. Vigo. *The Vehicle Routing Problem*. Monographs on Discrete Mathematics and Applications. Society for Industrial and Applied Mathematics, 2002.

[22] J. Zhang, W.H.K. Lam, and B.Y. Chen. On-time delivery probabilistic models for the vehicle routing problem with stochastic demands and time windows. *European Journal of Operational Research*, 249(1):144 – 154, 2016.

Appendix A

Raw experimental results

Dataset	Ban method	# Iter.	Time (s)	Costs (RMP)
V2015/T	Time window	46	931.34	1299.2
V2015/S	Time window	66	2015.12	4466.61
V2015/M	Time window	191	7126.93	12891.01
V2015/T	Route	27	1035.42	1299.2
V2015/S	Route	120	3818.35	4533.5
V2015/M	Route	94	3045.29	16433.27
V2016/T	Time window	17	224.25	1539.21
V2016/S	Time window	126	4164.62	4748.86
V2016/M	Time window	154	3967.5	12500.07
V2016/T	Route	16	208.74	1539.21
V2016/S	Route	67	1774.38	4876.17
V2016/M	Route	66	1423.34	13845.54

Table A.1: Results of banning method experiments for IMIP

Dataset	Pr. prob.	# Sim. worlds	# Iter.	Time (s)	Costs (RMP)	Costs	# Vehicles
V2015/M	DP	20	651	9601.39	6729.68	7553.66	5
V2015/M	DP	50	670	14121.98	6759.97	7581.92	5
V2015/M	DP	100	468	14578.73	6284.54	7625.05	5
V2015/M	LS	20	644	4957.86	6211.18	7392.71	5
V2015/M	LS	50	745	8622.34	6406.05	7384.62	5
V2015/M	LS	100	684	8370.77	6634.39	7509.89	5
V2016/M	DP	20	459	8508.59	6090.98	6449.42	4
V2016/M	DP	50	493	15972.04	6132.14	6499.84	4
V2016/M	DP	100	621	26327.33	6859.29	7462.23	5
V2016/M	LS	20	587	3919.33	5540.48	6353.51	4
V2016/M	LS	50	746	10080.04	5802.61	6372.23	4
V2016/M	LS	100	598	11704.98	5906.01	6405.38	4

Table A.2: Impact of number of simulation worlds on the pricing problem methods.

Dataset	Pr. prob.	Min. rel.	# Iter.	Time (s)	Costs (RMP)	Costs	# Vehicles
V2015/M	DP	75%	663	20302.59	6484.14	7537.66	5
V2015/M	DP	90%	718	17302.78	6655.97	7474.39	5
V2015/M	DP	95%	468	14578.73	6284.54	7625.05	5
V2015/M	DP	99%	590	24821.82	7385.39	8743.67	6
V2015/M	LS	75%	639	8155.46	6220.81	7513.36	5
V2015/M	LS	90%	562	6346.76	6515.22	7506.01	5
V2015/M	LS	95%	684	8370.77	6634.39	7509.89	5
V2015/M	LS	99%	514	5704.01	7002.54	7565.28	5
V2016/M	DP	75%	322	11407.64	5930.34	6536.63	4
V2016/M	DP	90%	192	6368.1	7081.9	8005.39	5
V2016/M	DP	95%	621	26327.33	6859.29	7462.23	5
V2016/M	DP	99%	356	16142.85	6765.92	7804.62	5
V2016/M	LS	75%	504	4159.78	5609.51	6384.97	4
V2016/M	LS	90%	627	9436.03	5761.24	6348.86	4
V2016/M	LS	95%	598	11704.98	5906.01	6405.38	4
V2016/M	LS	99%	374	4122.17	6417.33	7549.46	5

Table A.3: Impact of minimum reliability on the pricing problem methods.

Dataset	Pr. prob.	Tw. size	# Iter.	Time (s)	Costs (RMP)	Costs	# Vehicles
V2015/M	DP	10m	78	546.47	18880.87	18967.81	14
V2015/M	DP	30m	115	1647.19	12821.47	13989.35	10
V2015/M	DP	1h	184	4682.58	9841.83	11199.56	8
V2015/M	DP	2h	532	14852.78	8036.16	8691.17	6
V2015/M	DP	4h	668	14254.16	7299.58	7476.78	5
V2015/M	DP	orig	468	14578.73	6284.54	7625.05	5
V2015/M	LS	10m	189	1130.69	17676.86	17792.97	13
V2015/M	LS	30m	258	2087.6	11807.14	12667.38	9
V2015/M	LS	1h	343	2507.46	9442.96	9974.71	7
V2015/M	LS	2h	508	3463.16	7874.3	8653.55	6
V2015/M	LS	4h	583	6331.53	7020.1	7528.66	5
V2015/M	LS	orig	684	8370.77	6634.39	7509.89	5
V2016/M	DP	10m	45	122.4	18719.02	18719.02	13
V2016/M	DP	30m	66	1448.35	11887.8	13141.11	9
V2016/M	DP	1h	134	2508.48	8307.41	10360.37	7
V2016/M	DP	2h	647	6165.39	6795.35	7621.26	5
V2016/M	DP	4h	235	4896.16	7089.79	7763.9	5
V2016/M	DP	orig	621	26327.33	6859.29	7462.23	5
V2016/M	LS	10m	149	663.52	18619.59	18619.59	13
V2016/M	LS	30m	166	997.11	10877.3	11876.64	8
V2016/M	LS	1h	232	1396.34	8033.63	9227.29	6
V2016/M	LS	2h	501	4621.27	6690.3	7616.56	5
V2016/M	LS	4h	420	3982.6	6453.73	7499.58	5
V2016/M	LS	orig	598	11704.98	5906.01	6405.38	4

Table A.4: Impact of time window size on the pricing problem methods.

Dataset	Pr. prob.	# Iter.	Time (s)	Costs (RMP)	Costs	# Vehicles
V2015/T	DP	66	1532.88	1299.2	1299.2	1
V2015/S	DP	252	8522.38	3359.57	4307.23	3
V2015/M	DP	468	14578.73	6284.54	7625.05	5
V2015/L	DP	931	23928.1	11112.73	13213.35	9
V2015/T	LS	26	190.84	1323.33	1323.33	1
V2015/S	LS	195	1417.79	3306.42	4441.04	3
V2015/M	LS	684	8370.77	6634.39	7509.89	5
V2015/L	LS	924	12279.54	10890.07	13082.42	9
V2016/T	DP	24	98.3	1539.21	1539.21	1
V2016/S	DP	187	2058.39	3529.25	4600.82	3
V2016/M	DP	621	26327.33	6859.29	7462.23	5
V2016/L	DP	885	26591.32	11444.1	13675.07	9
V2016/T	LS	24	199.95	1539.21	1539.21	1
V2016/S	LS	323	2522.95	3236.11	3236.11	2
V2016/M	LS	598	11704.98	5906.01	6405.38	4
V2016/L	LS	838	12809.56	11165.8	12540.75	8

Table A.5: Impact of number of customers on the pricing problem methods.

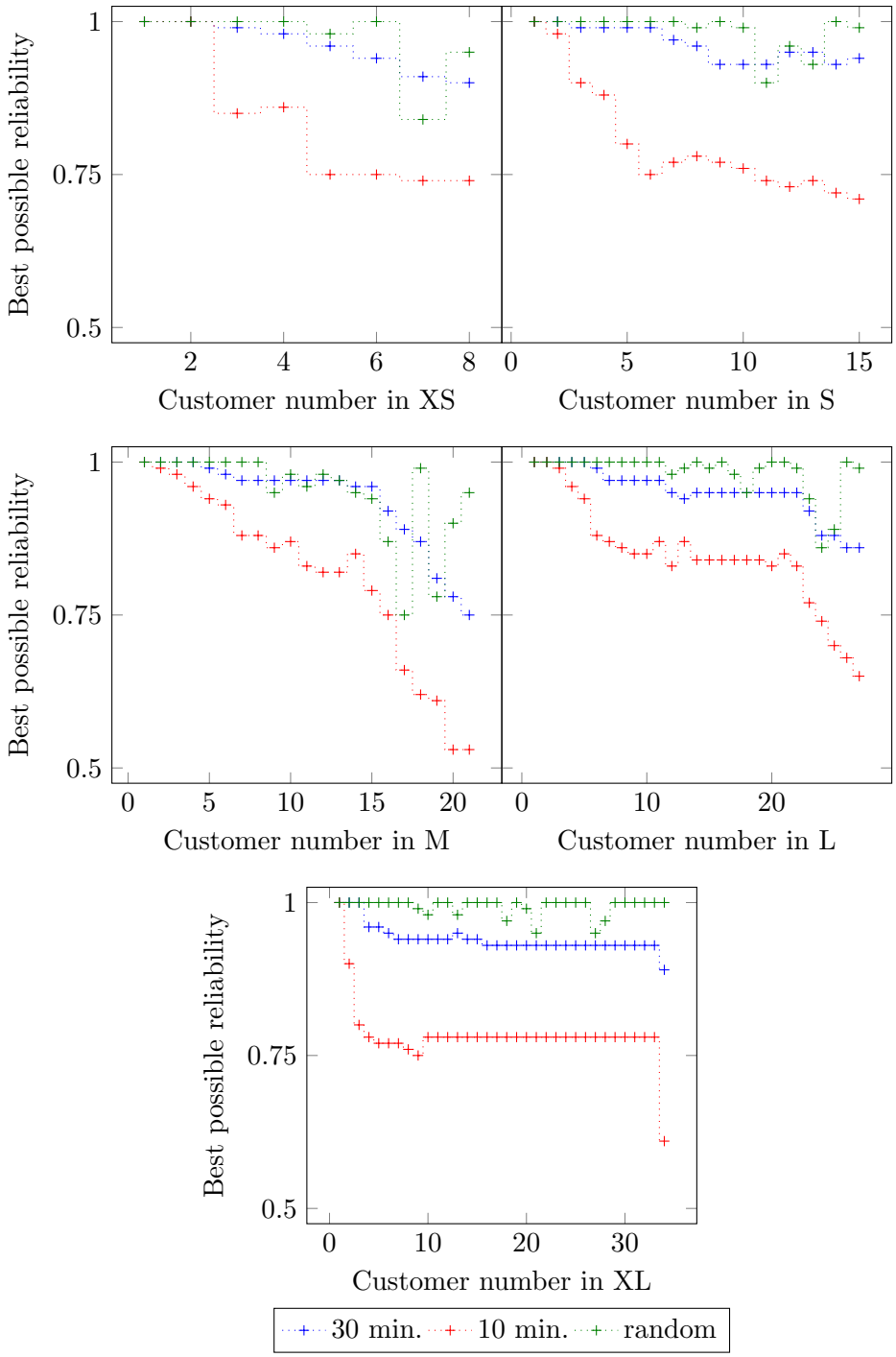

Figure A.1: Best possible reliability for customers in routes defined in Table 5.4.

# Sim. worlds	Time window	Min. rel.	XS	S	M	L	XL
50	orig	0.99	0.58	1.1	1.25	1.51	1.22
50	orig	0.95	0.59	1.04	1.26	1.49	1.57
50	orig	0.75	0.58	1.04	1.25	1.49	1.67
50	orig	rand	0.61	0.98	1.25	1.48	1.62
50	10 min	0.99	0.61	0.76	0.98	1.18	1.45
50	10 min	0.95	0.58	0.77	0.98	1.18	1.46
50	10 min	0.75	0.61	1.02	1.23	1.18	1.47
50	10 min	rand	0.6	0.83	1.23	1.19	1.47
50	30 min	0.99	0.59	0.98	1.23	1.23	1.46
50	30 min	0.95	0.62	0.98	1.24	1.23	1.46
50	30 min	0.75	0.59	0.98	1.32	1.49	1.63
50	30 min	rand	0.59	0.95	1.25	1.22	1.27
50	day	0.99	0.58	1.01	1.24	1.48	1.6
50	day	0.95	0.58	1.0	1.24	1.48	1.6
50	day	0.75	0.58	1.05	1.26	1.48	1.6
50	day	rand	0.58	1.0	1.24	1.47	1.61
50	rand	0.99	0.62	0.97	1.25	1.23	1.5
50	rand	0.95	0.62	0.96	1.25	1.23	1.5
50	rand	0.75	0.63	1.0	1.25	1.6	1.63
50	rand	rand	0.63	1.06	1.25	1.23	1.56
100	orig	0.99	0.72	1.24	1.54	1.86	1.52
100	orig	0.95	0.71	1.27	1.54	1.93	1.94
100	orig	0.75	0.73	1.22	1.56	1.86	1.92
100	orig	rand	0.71	1.22	1.55	1.84	1.97
100	10 min	0.99	0.57	0.98	1.53	1.54	1.75
100	10 min	0.95	0.56	0.98	1.53	1.54	1.76
100	10 min	0.75	0.69	0.98	1.53	1.57	1.82
100	10 min	rand	0.57	0.92	1.53	1.54	1.75
100	30 min	0.99	0.69	1.02	1.55	1.59	1.76
100	30 min	0.95	0.74	1.24	1.54	1.6	1.75
100	30 min	0.75	0.71	1.27	1.54	1.92	1.99
100	30 min	rand	0.73	1.21	1.58	1.56	1.59
100	day	0.99	0.74	1.27	1.57	1.87	1.91
100	day	0.95	0.72	1.26	1.56	1.9	1.98
100	day	0.75	0.72	1.27	1.57	1.87	1.9
100	day	rand	0.72	1.26	1.56	1.86	1.91
100	rand	0.99	0.73	1.21	1.54	1.6	1.81
100	rand	0.95	0.75	1.23	1.55	1.6	1.8
100	rand	0.75	0.76	1.25	1.54	1.94	1.92
100	rand	rand	0.76	1.31	1.54	1.62	1.88

Table A.6: Execution time in seconds of binary search method (Section 4.1) on different routes.

# Sim. worlds	Time window	Min. rel.	XS	S	M	L	XL
50	orig	0.99	0.03	0.1	0.13	0.19	0.35
50	orig	0.95	0.03	0.09	0.13	0.19	0.34
50	orig	0.75	0.03	0.09	0.13	0.19	0.34
50	orig	rand	0.03	0.09	0.13	0.19	0.34
50	10 min	0.99	0.03	0.09	0.13	0.19	0.34
50	10 min	0.95	0.03	0.09	0.12	0.19	0.34
50	10 min	0.75	0.03	0.09	0.12	0.19	0.34
50	10 min	rand	0.03	0.09	0.12	0.19	0.34
50	30 min	0.99	0.03	0.09	0.13	0.19	0.34
50	30 min	0.95	0.03	0.09	0.13	0.19	0.34
50	30 min	0.75	0.03	0.09	0.12	0.19	0.34
50	30 min	rand	0.03	0.09	0.12	0.19	0.34
50	day	0.99	0.03	0.09	0.13	0.19	0.35
50	day	0.95	0.03	0.09	0.13	0.23	0.35
50	day	0.75	0.03	0.09	0.14	0.19	0.35
50	day	rand	0.03	0.09	0.13	0.2	0.34
50	rand	0.99	0.03	0.09	0.13	0.19	0.34
50	rand	0.95	0.03	0.09	0.13	0.2	0.35
50	rand	0.75	0.03	0.09	0.13	0.19	0.34
50	rand	rand	0.03	0.09	0.13	0.19	0.34
100	orig	0.99	0.05	0.16	0.23	0.35	0.61
100	orig	0.95	0.05	0.17	0.23	0.35	0.61
100	orig	0.75	0.05	0.17	0.23	0.35	0.61
100	orig	rand	0.05	0.17	0.23	0.35	0.61
100	10 min	0.99	0.05	0.16	0.22	0.35	0.61
100	10 min	0.95	0.05	0.16	0.23	0.35	0.6
100	10 min	0.75	0.05	0.16	0.22	0.35	0.62
100	10 min	rand	0.05	0.17	0.22	0.34	0.61
100	30 min	0.99	0.05	0.16	0.23	0.34	0.62
100	30 min	0.95	0.05	0.17	0.23	0.35	0.62
100	30 min	0.75	0.05	0.17	0.22	0.34	0.61
100	30 min	rand	0.05	0.17	0.23	0.35	0.62
100	day	0.99	0.05	0.17	0.23	0.35	0.62
100	day	0.95	0.05	0.17	0.23	0.35	0.64
100	day	0.75	0.05	0.17	0.23	0.35	0.62
100	day	rand	0.05	0.17	0.23	0.35	0.62
100	rand	0.99	0.05	0.17	0.22	0.35	0.62
100	rand	0.95	0.05	0.17	0.22	0.35	0.62
100	rand	0.75	0.05	0.16	0.22	0.34	0.62
100	rand	rand	0.05	0.17	0.23	0.35	0.61

Table A.7: Execution time in seconds of backwards calculation algorithm (Section 4.2) on different routes.

# Sim. worlds	Time window	Min. rel.	XS	S	M	L	XL
50	orig	0.99	0.62	0.98	1.3	1.6	1.43
50	orig	0.95	2.52	9.72	24.91	-	3.87
50	orig	0.75	19.21	41.37	-	-	24.31
50	orig	rand	0.54	1.18	-	-	7.72
50	10 min	0.99	0.55	1.18	1.4	1.8	1.58
50	10 min	0.95	0.59	1.28	1.54	1.94	1.69
50	10 min	0.75	0.74	1.33	1.88	2.1	2.14
50	10 min	rand	0.61	1.27	1.69	1.96	1.75
50	30 min	0.99	0.56	1.06	1.35	1.73	1.59
50	30 min	0.95	0.62	1.25	1.65	1.98	1.78
50	30 min	0.75	6.58	55.77	1.91	-	45.42
50	30 min	rand	0.68	1.3	1.96	2.11	1.68
50	day	0.99	1.72	7.45	3.33	4.36	4.05
50	day	0.95	1.63	7.47	11.67	6.99	4.08
50	day	0.75	1.75	7.5	9.65	8.12	4.07
50	day	rand	1.65	7.5	7.94	18.53	4.08
50	rand	0.99	0.5	0.89	1.32	1.47	1.35
50	rand	0.95	0.65	2.11	1.56	2.12	1.74
50	rand	0.75	22.01	27.11	2.07	-	39.96
50	rand	rand	0.64	1.01	1.81	1.62	1.47
100	orig	0.99	9.98	29.67	-	57.42	34.69
100	orig	0.95	11.47	34.92	-	-	18.68
100	orig	0.75	25.8	-	-	-	-
100	orig	rand	11.94	2.4	-	-	29.32
100	10 min	0.99	1.44	2.83	3.29	4.23	3.83
100	10 min	0.95	1.16	2.65	3.42	4.34	3.74
100	10 min	0.75	1.19	2.69	3.51	4.73	3.89
100	10 min	rand	1.18	2.89	3.84	4.35	3.66
100	30 min	0.99	1.28	2.73	3.72	4.69	4.39
100	30 min	0.95	1.29	2.86	3.81	4.41	3.84
100	30 min	0.75	37.72	-	19.22	-	-
100	30 min	rand	1.26	2.83	3.59	4.47	3.66
100	day	0.99	4.83	23.56	12.69	-	11.45
100	day	0.95	4.91	23.19	48.94	55.77	11.53
100	day	0.75	4.9	23.62	-	55.5	11.51
100	day	rand	4.91	23.05	-	-	13.15
100	rand	0.99	2.08	3.57	4.93	7.38	4.85
100	rand	0.95	1.22	2.86	3.77	4.48	4.11
100	rand	0.75	42.51	-	17.56	-	-
100	rand	rand	1.28	2.1	3.57	3.39	7.7

Table A.8: Execution time in seconds of mixed integer method (Section 4.3) on different routes.