Erasmus University Rotterdam
Erasmus School of Economics
Econometrics and Management Science

Master Thesis Operations Research and Quantitative Logistics

October 29, 2022

A p-step Formulation for the Vehicle Routing Problem with Time Windows

Author:
Wouter Vaes

Student number:
443914

Supervisor:
dr. T.A.B. Dollevoet

Second assessor:
dr. R. Spliet

Abstract

In this thesis, we provide a family of formulations for the vehicle routing problem with time windows (VRPTW) by extending the p-step formulation proposed by Dollevoet et al. (2020). The parameter p indicates the length of partial paths which are concatenated to construct routes. The p-step approach is considered a generalization as next to providing new formulations, it unifies traditional arc-based and path-based formulations. We compute the linear programming relaxation of the VRPTW with a column generation algorithm and break down the resulting pricing problem into a polynomial number of well-known problems: elementary shortest path problems with resource constraints (ESPPRC). We solve these with an exact pulse algorithm and, subsequently, provide a dive-and-fix heuristic to generate an integer solution for the VRPTW. Our aim is to provide empirical evidence that using the p-step formulation to solve the VRPTW can gain computational advantage compared to state-of-the-art algorithms. The computational experiments with benchmark instances do not support this statement, but our specifically designed instances do favor intermediate values of p and suggest that computational advantage can be gained with further algorithmic development.

Keywords:
vehicle routing problem with time windows - p-step formulation
pulse algorithm - dive-and-fix heuristic

Contents

1 Introduction

The vehicle routing problem (VRP) is a classical combinatorial optimization problem in which minimum cost routes are designed for a limited number of homogeneous vehicles to cover a set of customers with known demand. Each route starts and ends at the depot, and the cumulative demand of the customers served should not exceed the capacity of a vehicle. Dantzig and Ramser (1959) were the first to use the VRP to model and optimize the physical distribution of gasoline by delivery trucks. Such flow of products through a transportation network has a vital role in the field of logistics. As distribution processes have a significant share in the costs of businesses, the VRP and its extensions have been widely studied over the past decades.

One of these extensions is the so-called vehicle routing problem with time windows (VRPTW), a generalization of the VRP where time windows give a time interval in which the service of every customer should start (Toth and Vigo, 2002). These time windows can either be soft or hard. Soft windows can be violated against additional cost, whereas hard windows are binding. In other words, they do not allow the vehicle to arrive late and make the vehicle wait until the start of the time window when it arrives too early. We consider the latter case in the remainder of this thesis.

Some specific applications where hard time windows are encountered include urban newspaper distribution, postal and supermarket deliveries and security patrol services. Due to significant progress in information technology, business organizations are able to devote more attention to optimization and time-efficiency throughout their supply chain (Kallehauge et al., 2005). Because of this, the practical utilization of the VRPTW has advanced within operation and supply chain designs. In parallel, the visible applicability stimulated research in the direction of both exact and heuristic solving methods (El-Sherbeny, 2010).

Generally, exact algorithms to solve the VRPTW are based on a (mixed) integer linear programming formulation. Two well-known types are the arc-based formulation, where each arc has a corresponding binary decision variable indicating whether it is used, and the path-based formulation, where binary decision variables indicate whether a path or route is used. In his comprehensive review, Kallehauge (2008) states that the main contributions to solving the VRPTW to optimality lay in the area of exact approaches relying on a path-based formulation; the main example is the set partitioning formulation (Balinski and Quandt, 1964). Usually, a column generation algorithm is used to compute the LP bound of such formulations.

Recently, Dollevoet et al. (2020) introduced a new generalized formulation for the capacitated vehicle routing problem (CVRP), which resembles the VRPTW if time window constraints are neglected. Their p-step formulation provides a family of formulations in which partial paths of length p are concatenated to construct routes. The p-step formulation is considered a generalization as the formulation is arc-based when p is one, path-based when p is larger than the total number of customers, and it gives new formulations for intermediate values of p. They show empirical evidence that their formulation could provide computational advantage compared to previously mentioned traditional formulations.

In this thesis, we build on the framework of Dollevoet et al. (2020) and extend the p-step formulation so that it becomes a formulation for the VRPTW. In other words, we redefine the p-step and incorporate additional constraints to ensure time windows are not violated. Furthermore, we use a column generation algorithm to solve the linear programming (LP) relaxation of the VRPTW. We decompose the pricing problem into an elementary shortest path problem with resource constraints (ESPPRC) and solve it by implementing an adjusted version of the pulse algorithm proposed by Lozano et al. (2016). Eventually, we apply a dive-and-fix heuristic which implements before mentioned algorithms to solve the VRPTW to integrality. The aim of this thesis is not only to provide a new generalization of formulations for the VRPTW, but also to investigate whether these formulations can achieve computational advantages.

In our computational experiments, we first solve selected benchmark instances from the literature with our dive-and-fix heuristic. As these results do not show evident proof of potential computational advantage of using the p-step formulation, we also generate problem instances specifically designed to favor intermediate values of p. Even though the computation times cannot compete with state-of-the-art algorithms, four out of five constructed instances suggest potential computational gains of using the p-step formulation to solve the VRPTW. Note that further development in algorithms designed for intermediate values of p is necessary.

The remainder of this thesis is structured as follows. Section 2 gives an overview of the relevant scientific literature. In Section 3, we present a formal problem description of the VRPTW and introduce the p-step formulation. Subsequently, in Section 4 and Section 5 we present a column generation algorithm and elaborate on the pulse algorithm, which is used to solve the ESPPRCs in the pricing problem. In Section 6, we propose the dive-and-fix heuristic we use for our computational experiments. We discuss the results of these experiments in Section 7. Finally, in Section 8 we conclude our thesis with a summary, our limitations and suggestions for future research.

2 Literature review

In Section 2.1, we give a brief overview of studies on heuristic methods to solve the VRPTW and discuss two general formulations used in exact approaches. Thereafter, in Section 2.2, we elaborate on column generation algorithms and related pricing problems. Section 2.3 covers diving heuristics used to compute integer solutions in a vehicle routing setting.

2.1 The VRPTW: heuristic and exact algorithms

Since Dantzig and Ramser (1959) introduced the VRP, the problem itself and its extensions have been extensively studied for decades. The reasons for this interest are both its widespread applicability and considerable difficulty (Toth and Vigo, 2002). VRPs belong to the class of $\mathcal{NP}$-hard problems. Therefore, the size of instances may cause excessively long run times when using exact algorithms to solve large instances to optimality. As the VRPTW is a generalization of the VRP, it is also considered to be $\mathcal{NP}$-hard (Lenstra and Rinnooy Kan, 1981). As a result, (meta-)heuristics might be used in order to solve VRPTWs within a reasonable time span. Some heuristic algorithms are simulated annealing (Chiang and Russell (1996), Mahmudy (2014)) and tabu search (Bräysy and Gendreau, 2002). Both approaches are single solution based; only one initial candidate solution is considered. Meta-heuristics that are population based are, for example, the evolutionary genetic algorithm (Ombuki et al., 2006) or swarm intelligence algorithms like ant colony optimization (Toklu et al., 2014) and particle swarm optimization (Zhu et al. (2006), Liu et al. (2009)).

Above mentioned algorithms require problem-specific tuning of parameters and may not guarantee a close to optimal solution. When optimality becomes crucial and quick feasible solutions are insufficient, exact algorithms have proven to be successful in solving VRPTWs, too. Scientific literature identifies, amongst others, two main (mixed) integer linear programming formulations on which exact approaches depend: arc-based formulations and path-based formulations. The studies of Kallehauge et al. (2007) and Mak and Ernst (2007) are both examples of first-mentioned formulation in which binary variables are associated with the use of arcs. Contrary, path-based formulations let binary variables indicate whether a path, i.e. route, is used. The best-known example of such formulation is the set partitioning formulation (Balinski and Quandt, 1964), which has been the subject of numerous studies since. The difference between these two formulations lies in the constructed LP bounds and their respective computation times. Generally, the LP relaxation of arc-based formulations is found quickly, but provides relatively weak LP bounds as a consequence. Path-based formulations, on the other hand, do provide strong LP bounds. However, this requires substantial computation times due to the number of variables that grows exponentially with instance size.

2.2 Column generation

To overcome dealing with an excessive amount of variables, it is common to apply a column generation algorithm to compute LP bounds in path-based formulations. Such an approach has exponential worst-case computation time. The use of column generation dates back to Desrochers et al. (1992), who applied a Dantzig-Wolfe decomposition (DWD) framework in which the assignment constraints, which link the vehicles to their route, were considered as the linking constraints. Another approach to split the VRPTW into a master problem and a separate subproblem for each vehicle is Lagrangian Relaxation (LR); Kohl and Madsen (1997) also relax the assignment constraints and end up with a master problem in which optimal Lagrangian multipliers are found by a best of both worlds combination of subgradient and bundle methods. Lagrangian Decomposition (LD) for the VRPTW, based on variable splitting suggested by Jörnsten et al. (1986), is proposed by Halse (1992). He proposes three different splitting methods by introducing additional decision variables and constraints. In this way, the resulting master problem needs to optimize for more multipliers, making it more difficult to solve. In addition, the subproblems do not remain identical for each vehicle. As Kohl (1995) derived that the lower bounds of LD are similar to the bounds computed with either DWD or LR, the last two more evident methods have been extended in later studies. For example, Kohl et al. (1999) introduce the 2-path cut, a strong valid inequality, and incorporate this into their DWD approach. In continuation, Cook and Rich (1999) do not only propose a new implementation of the before mentioned method but also apply parallel processing. A merger of LR and DWD is suggested by Kallehauge et al. (2001), who state that the combination of a DWD with a bundle method can be used to strengthen known LR approaches.

2.2.1 Subproblems: SPPRC and ESPPRC

Most column generation algorithms for the VRPTW decompose the subproblem, i.e. pricing problem, into an elementary shortest path problem with resource constraints (ESPPRC) for each vehicle. Here, an elementary path is one in which each customer can be visited at most once. Furthermore, the resource constraints concern the time window of each customer and the capacity of a vehicle. Since the ESPPRC is $\mathcal{NP}$-hard in the strong-sense (Dror, 1994), it is often relaxed by allowing for cycles; resulting in the shortest path problem with resource constraints (SPPRC). Negative cycles may exist, but infinite cycles cannot occur as time goes forward and capacity accumulates with each visit to a customer. Most approaches to solve the SPPRC rely on dynamic programming, such as the pseudo-polynomial labeling algorithm in the seminal work of Desrochers et al. (1992). However, solving the relaxed SPPRC generates significantly more admissible columns than the ESPRRC would, providing weaker lower bounds and causing possible impractically large branch-and-bound trees. Irnich and Villeneuve (2006) show that k-cycle elimination with $k \geq 3$, a compromise between solving the SPPRC and ESPPRC, significantly improves obtained lower bounds. Another compromise is the so-called ng-route relaxation proposed by Baldacci et al. (2011), which allows routes to be non-elementary but forbids short localized cycles. This restriction is realized by a dynamically computed set associated with each customer containing forbidden following customers. Currently, the ng-route relaxation is the state-of-the-art exact approach to solve the VRPTW.

In their approach to solve the VRPTW, Rousseau et al. (2004) use constraint programming to solve the ESPPRC. This does result in a flexible approach, however, no significant improvement is made in either solution quality or computation times. The studies of Feillet et al. (2004) and Chabrier (2006) extend the labeling algorithm of Desrochers et al. (1992) and embed this in a branch-and-price scheme. Both works show improved lower bounds at each node in the branching tree and, additionally, found optimal solutions to numerous open benchmark instances (Solomon, 1987). Building on this, Feillet et al. (2007) propose refinements to reduce computation times. Their results show the substantial influence of applying limited discrepancy search to dynamic programming and removing labels during the search based on lower bounds. Another algorithm to tackle the ESPPRC is introduced by Righini and Salani (2008); their bidirectional labeling algorithm with decremental state-space relaxation heavily outperforms the earlier mentioned dynamic programming approaches of Feillet et al. (2004) and Chabrier (2006). Moreover, Jepsen et al. (2008a) and Jepsen et al. (2008b) suggest subset-row inequalities, which improve the lower bound at all nodes in the branching tree. However, its usefulness is questioned as the pricing problem becomes significantly more complex. To overcome this, Desaulniers et al. (2008) propose before mentioned subset-row inequalities with a combination of tabu search and the exact algorithm of Righini and Salani (2008). They prove their approach to be successful as they find optimal solutions for five previously unsolved Solomon benchmark instances. A recent contribution is the pulse algorithm by Lozano et al. (2016); an exact solution approach that extends partial paths in a recursive manner. The partial paths either reach the final node and provide a solution to the ESPPRC or are rejected by pruning strategies. The authors show that their implicit enumeration with a novel bounding scheme to limit the solution space performs well compared to current state-of-the-art algorithms.

2.3 Diving heuristics

In most cases, applying and solving column generation results in a fractional solution. To solve the VRPTW to integrality, techniques and extensions of branch-and-price algorithms are widely used. Examples of these are diving heuristics, which apply a depth-first search in the branch-and-price tree. After solving each branching node, a part of the fractional variables is bounded or fixed to promising integral values by greedy or rounding strategies (Berthold, 2006). Subsequently, the LP relaxation is solved in an iterative manner to create a simulation of a possible root-leaf path. In this context, we differ from exact approaches as we neglect to balance the tree and increasing the dual bound. The study of Günlük et al. (2006) is an example of applying a diving heuristic in a vehicle routing setting. Their fix-and-price heuristic applies fractional diving with a threshold value in the optimization component of a decision support system.

3 A p-step formulation

The VRPTW is a well known problem, but to introduce notation and for the sake of completeness we provide a general problem description in Section 3.1. Moreover, in Section 3.2, we present a new binary programming formulation of the VRPTW and define the p-steps that it uses, based on Dollevoet et al. (2020). In Section 3.3, we discuss how to modify the formulation to only include a finite number of variables.

3.1 Problem description VRPTW

To define the problem, we let $N = \{0, 1, \ldots, n+1\}$ be the set of vertices where $C = \{1, 2, \ldots, n\}$ represents the customers, and 0 and $n+1$ are the starting and ending depot, respectively. The set of arcs, $A = \{(i,j) : i,j \in N, i \neq j, i \neq n+1, j \neq 0\}$, includes the direct connections among customers and between the customers and the depot. The VRPTW can now be defined on the directed graph $G = (N, A)$.

Each customer $i \in C$ has a demand $q_i > 0$, a time window $[a_i, b_i]$ in which service should start, and a service time s_i. For the depots, we let $q_0 = q_{n+1} = s_0 = s_{n+1} = 0$ and assume identical time windows $[a_0, b_0]$ to represent the scheduling horizon. To satisfy demand within the given time horizon, an unlimited fleet of homogeneous vehicles with capacity Q is available. These vehicles visit customers by traversing an elementary path from 0 to $n+1$, i.e., no vertex is repeated in the path. We refer to such a path as a route. The capacity of a vehicle cannot be exceeded by the total demand of the customers it visits. Therefore, we assume $q_i \leq Q$ for all $i \in N$. A vehicle should fulfill the complete demand of a customer i as customers are visited exactly once. Furthermore, it must arrive at the customer and start service before b_i. A vehicle may arrive before the time window opens but should always wait until a_i before servicing customer i. Each arc $(i, j) \in A$ has associated cost and travel time denoted by $c_{ij}, t_{ij} \geq 0$, respectively. We assume that $c_{ij} = t_{ij}$ and that they are equal to the Euclidean distance d_{ij} between i and j. As a result, both c_{ij} and t_{ij} satisfy the triangle inequality. The objective of the VRPTW is to determine an optimal set of routes to satisfy all customer demands within respective time windows while total costs should be minimized.

3.2 Binary programming formulation

We present a binary programming formulation of the VRPTW that concatenates so-called p-steps to represent routes, based on Dollevoet et al. (2020). The parameter p corresponds to the length of path P_r associated with p-step r. P_r is an elementary path in G that either visits $p+1$ nodes by traversing exactly p arcs or traverses at most p arcs after starting at the depot. We denote a p-step r by a triplet (P_r, d_r, t_r), where d_r is the prior demand of r, i.e., the sum of the customers' demand on a route before arriving at the first location of P_r. In addition, t_r denotes when the first location of P_r is serviced, which is referred to as the start time of r. The total time of P_r, including travel, service and waiting times, is denoted by $T(r)$. Besides, $D(r)$ denotes the cumulative demand of all customers visited on P_r. Lastly, we denote l_{ri} for each $i \in P_r$ as the time at which service starts at customer i. For a p-step r to be feasible, it should hold that $0 \leq d_r + D(r) \leq Q$ and $0 \leq t_r + T(r) \leq b_0$. Moreover, it should be possible to visit all customers within their respective time windows. This means $l_{ri} + s_i + t_{ij} \leq l_{rj}$ for each $(i, j) \in P_r$ with $l_{ri} \in [a_i, b_i]$ for each $i \in P_r$. Now, if s is the start location and f the end location of path P_r, we define $T(r) = l_{rf} - l_{rs}$. Let R^p be the set of all feasible p-steps.

As said before, p-steps are chained to construct and represent routes. Three conditions should hold when two p-steps r and u are concatenated. First, the last location $f \in C$ of r should be equal to the first location of u. Second, the total demand after leaving r should match the prior demand of u and the demand of f, or in other words, $d_r + D(r) = d_u + q_f$. Lastly, the start time of u should always be later than the ending time of r, that is, $t_r + T(r) \leq t_u$. Note that as we enforce a single visit per customer in our formulation, we establish that any concatenated path from 0 to $n+1$ is elementary and, therefore, produces a feasible route.

Next, we present additional parameters to formulate the VRPTW. We introduce the degree of customer i on path P_r of p-step r, denoted by a_r^i, to ensure single visits for each customer. Let a_r^i

be 2 if p-step r visits customer $i \in C$ once, 1 if i is either the first or the last location on P_r, and 0 otherwise. In addition, to correctly concatenate p-steps, we introduce location degree e_r^i, demand degree q_r^i, and time degree v_r^i. Let e_r^i be -1 if i is the last location on P_r, 1 if i is the first location visited by P_r, and 0 otherwise. Then, let q_r^i be $-d_r - D(r)$ if i is the last location on P_r, $d_r + q_i$ if i is the first location visited by P_r, and 0 otherwise. Finally, let v_r^i be $-t_r - T(r)$ if i is the last location on P_r, t_r if i is the first location visited by P_r, and 0 otherwise.

Now, the VRPTW is formulated by the following binary programming formulation, where we let binary decision variables x_r denote whether p-step r is used:

$$\begin{aligned}
\min \quad & \sum_{r \in R^p} c_r x_r && && (1)\\
\text{s.t.} \quad & \sum_{r \in R^p} a_r^i x_r = 2 && \forall i \in C, && (2)\\
& \sum_{r \in R^p} e_r^i x_r = 0 && \forall i \in C, && (3)\\
& \sum_{r \in R^p} v_r^i x_r \geq 0 && \forall i \in C, && (4)\\
& \sum_{r \in R^p} q_r^i x_r = 0 && \forall i \in C, && (5)\\
& x_r \in \{0, 1\} && \forall r \in R^p. && (6)
\end{aligned}$$

In the remainder of this thesis, we call the above formulation (1)-(6) the p-step formulation. Here, the objective function (1) minimizes the total travel costs. The constraints (2) ensure that every customer is visited exactly once. Subsequently, constraints (3)-(5) ensure the correct concatenation of p-steps: a selected p-step that ends at node i can only be chained with another p-step if it starts at i with sufficient prior demand and an achievable start time. Finally, constraints (6) restrict the domain of the p-step variables x.

3.3 Compact formulation

Both the prior demand d_r and the start time t_r are continuous. As pointed out by Dollevoet et al. (2020), this not only makes the number of p-steps in R^p infinite, but also the number of decision variables x_r. We follow their approach to redefine (1)-(6) into a compact p-step formulation of the VRPTW. Hence, we limit the number of p-steps and only consider p-steps with either minimal or maximum prior demand or start time, such that with a convex combination of the extremes we can construct any other p-step.

We first relax constraints (5) by changing the equalities into inequalities as follows (see Dollevoet et al. (2020) for a detailed explanation):

$$\sum_{r \in R^p} q_r^i x_r \geq 0 \quad \forall i \in C. \tag{7}$$

Next, we introduce an additional parameter b_{ij}^r and binary arc-flow variables θ_{ij}. Let b_{ij}^r be 1 if arc $(i, j) \in A$ is used by p-step r, and 0 otherwise, and let θ_{ij} be 1 if arc $(i, j) \in A$ is used, and 0 otherwise. Now, we imply binary conditions on the arc-flow instead of on the p-step variables, which is not uncommon in a set partitioning formulation for the VRPTW, by replacing constraints (6) with:

$$\sum_{r \in R^p} b_{ij}^r x_r = \theta_{ij} \qquad \forall (i, j) \in A, \tag{8}$$

$$x_r \geq 0 \qquad \forall r \in R^p, \tag{9}$$

$$\theta_{ij} \in \{0, 1\} \qquad \forall (i, j) \in A. \tag{10}$$

If every arc in A is selected binarily, constraints (8) can construct a convex combination of several (extreme) p-steps to represent any route. Therefore, we allow for continuous p-step variables x in constraints (9). Note that constraints (2) and (3) imply the upper bound $x_r \leq 1$ for all $r \in R^p$.

Before we limit the number of p-steps, we define the minimal and maximum values of both d_r and t_r. For the prior demand, we assign $d_r = 0$ as minimal value and calculate the maximum value by taking the capacity of a vehicle and subtract the cumulative demand of all customers on p-step r, i.e., $d_r = Q - D(r)$. For t_r, we cannot simply assign minimal value $t_r = a_s$ and maximum value $t_r = b_f - T(r)$, where s is the start location and f the end location of path P_r, as $T(r)$ depends on t_r. If any waiting occurs between customers, increasing t_r decreases $T(r)$, and such later departure might result in the same time service starts at the last location on P_r. Therefore, we define parameters t_r^{early} and t_r^{late} and determine these as follows when path P_r is known. We traverse the path in opposite direction, and we calculate at each customer the earliest and latest start time possible to reach the next customer within their time window. In this way, we end up with reasonable minimal and maximum values for the start time t_r. See illustrative Example 1 below.

We can now limit the number of p-steps and obtain a compact formulation of the VRPTW with a finite number of variables, being of order n^{p+1}, by applying Proposition 1.

Example 1. Consider a 2-step r with $P_r = \{1,2,3\}$, time windows $[1,10]$, $[4,7]$ and $[5,6]$, respectively, travel times $t_{12} = t_{23} = 1$ and service times $s_i = 1$ for all $i \in P_r$. We start looking at customer 3 and observe $s_2 + t_{23} = 2$. To reach customer 3 as early as possible we set $l_{r2} = 3$, whereas for the late case we set $l_{r2} = 4$. However, the time window of customer 2 does not allow the early case start time. Hence, in both cases we have $l_{r2} = 4$ and, as a consequence, $l_{r3} = 6$. Next, as $s_1 + t_{12} = 2$, the starting times at customer 1 should be $t_r^{early} = t_r^{late} = 2$ such that $T(r) = 4$. Note that, the naive minimal value $t_r = a_1 = 1$ with associated $T(r) = 5$ would violate the earliest reasonable start time $t_r = 2$.

Proposition 1. *The p-step formulation (1)-(6) and model (1)-(4) and (7)-(10) have the same optimal solution value and LP bound when including the following p-steps in R^p only, where s is the start location and f the end location of P_r:*

1. *All p-steps r with $s = 0$, prior demand $d_r = 0$ and start time $t_r = t_r^{early}$.*
2. *All p-steps r with $s \neq 0$, $f \neq n+1$, prior demand $d_r = 0$ and start time $t_r = t_r^{early}$.*
3. *All p-steps r with $s \neq 0$, $f \neq n+1$, prior demand $d_r = 0$ and start time $t_r = t_r^{late}$.*
4. *All p-steps r with $s \neq 0$, $f \neq n+1$, prior demand $d_r = Q - D(r)$ and start time $t_r = t_r^{early}$.*
5. *All p-steps r with $s \neq 0$, $f \neq n+1$, prior demand $d_r = Q - D(r)$ and start time $t_r = t_r^{late}$.*
6. *All p-steps r with $s \neq 0$, $f = n+1$, prior demand $d_r = Q - D(r)$ and start time $t_r = t_r^{late}$.*

Proof. First of all, note that it is standard to apply binary conditions on the arc-flow variables instead of on the route variables. Such replacement has no influence on the optimal solution value or the LP bound. Next, assume p-step r, with start location s and end location f, is used in a solution of the p-step formulation (following part of the proof also holds if r is part of the solution to the LP relaxation). If r starts at the depot with $d_r > 0$ and $t_r > t_r^{early}$, we could as well use p-step $u = (P_r, 0, t_r^{early})$ as this would not affect the objective value nor violate constraints (4) and (7). By the same reasoning, we can use p-step $u = (P_r, Q - D(r), t_r^{late})$ to replace r if it ends at the depot with $d_r < Q - D(r)$ and $t_r < t_r^{late}$. Lastly, if the depot is neither the start or end location of r and it has $0 < d_r < Q - D(r)$ and $t_r^{early} < t_r < t_r^{late}$, we replace it by a convex combination of $u_1 = (P_r, 0, t_r^{early})$, $u_2 = (P_r, 0, t_r^{late})$, $u_3 = (P_r, Q - D(r), t_r^{early})$ and $u_4 = (P_r, Q - D(r), t_r^{late})$. □

4 Column generation algorithm

To solve the VRPTW with a dive-and-fix heuristic (see Section 6), we need an algorithm that solves the LP relaxation of the p-step formulation in each branching node. Even though it might be beneficial to use different algorithms for varying values of p, we choose a single algorithm for all values of p as this thesis aims to investigate the potential computational advantage of the p-step formulation. In this way, our computational experiments can provide reliable comparisons between formulations with different values of p.

The number of variables, being of order n^{p+1}, grows large when the instance size increases and even shows exponential growth in case of increasing p. Therefore, we solve the LP relaxation with a column generation algorithm so that not all variables are considered explicitly. Initially, the so-called restricted master problem (RMP) is solved. This RMP simplifies the problem by only including a limited subset of its variables. Subsequently, a pricing problem is solved in an iterative manner to find variables that could potentially improve the objective value, i.e. negative reduced cost variables. In each iteration, we add at least one found negative reduced cost variable to the subset of variables in the RMP and repeat the procedure. If no such variable is found, we terminate the algorithm, and the solution to the LP relaxation is equal to the optimal solution of the RMP in that iteration.

In Section 4.1, we propose a greedy look-ahead heuristic to find a feasible solution for the VRPTW, which we then decompose into p-steps to include in the initial RMP. Next, in Section 4.2, we show how to model the resulting pricing problem as an ESPPRC. To eliminate redundant pricing problems and reduce the size of the graph in the ESPPRC, we describe some preprocessing steps in Section 4.3.

4.1 Initialization: IMPACT heuristic

The master problem in our column generation algorithm is the compact p-step formulation (1)-(4) and (7)-(10). The initial subset of variables in the RMP includes all arc-flow variables θ and a limited number of p-step variables x, which represent a feasible solution for the VRPTW. To generate a feasible solution which can be decomposed into p-steps, we follow the work of Ioannou et al. (2001) and implement their IMPACT heuristic.

The heuristic follows a generic insertion framework, such as proposed by Solomon (1987), and constructs routes sequentially. We initialize a route with a so-called 'seed' customer and insert non-routed customers until no further customer has a feasible insertion place. Then, we find a new 'seed' customer and repeat the process until each customer is assigned to a vehicle. To obtain a good initial solution for the VRPTW, we utilize the time window relations between customers and do not insert customers based on distance only. In other words, we account for the impact of time windows while determining which non-routed customer to insert at which feasible position in the partial route. To measure and minimize the impact of inserting a customer i, we define three 'look-ahead' criteria: own impact IS_i, external impact IU_i and internal impact IR_i (see Sections 4.1.1-4.1.2). We calculate the customer selection criterion of customer i by combining the three impact measures linearly and denote this with Impact(i):

$$\text{Impact}(i) = \alpha_1 IS_i + \alpha_2 IU_i + \alpha_3 IR_i \tag{4.1}$$

Here, $\alpha_1, \alpha_2, \alpha_3 \geq 0$ and $\alpha_1 + \alpha_2 + \alpha_3 = 1$. These weights let each criterion have its own relative contribution to the overall impact. The complete IMPACT heuristic is presented by Algorithm 1. For a detailed description of the heuristic, we refer to Ioannou et al. (2001).

4.1.1 Own impact

The own impact criterion measures the effect of the inclusion of the time window of customer i. We denote the arrival time at customer i with e_i and let the own impact of inserting i equal the difference between this arrival time and its earliest service time a_i (see Equation (4.2)). The lower this value, the less time is wasted, and the more slack is provided for the insertion of other non-routed customers.

$$IS_i = e_i - a_i \tag{4.2}$$

4.1.2 External impact

To illustrate the concept of external impact, assume we have a customer i selected for insertion. After insertion, fewer non-routed customers can additionally be added to the route due to overlapping time windows. In other words, after inserting customer i we prefer the time windows of the remaining non-routed customers to overlap as little as possible. Therefore, the external impact measures the inclusion of time windows of all non-routed customers after inserting another non-routed customer i. This average is calculated by Equation (4.3), where J is the set of non-routed customers:

$$IU_i = \frac{1}{(|J|-1)} \sum_{j \in J \setminus \{i\}} \{\max\{(b_j - a_i - d_{ij}), (b_i - a_j - d_{ji})\}\} \tag{4.3}$$

4.1.3 Internal impact

The internal criterion calculates the impact of inserting customer i on all customers in the current route. Insertion of customer i in between customers u and v causes a distance increase z_{1i}, a time delay at customer v z_{2i} and has corresponding time window compatibility z_{3i}. These three factors are calculated as follows:

$$z_{1i}(u,v) = d_{ui} + d_{iv} - d_{uv} \tag{4.4}$$

$$z_{2i}(u,v) = (b_v - (e_u + s_u + d_{uv})) - (b_v - (e_i + s_i + d_{iv})) \tag{4.5}$$

$$z_{3i}(u,v) = b_i - (e_u + s_u + d_{ui}) \tag{4.6}$$

A weighted combination of the above factors determines the internal impact, i.e. accessibility, of inserting customer i between customers u and v. It might also be possible to insert customer i at different positions on the current route r. Therefore, we take the average over all feasible insertion places to calculate the overall internal impact of inserting customer i on route r. We define I_r the set of all feasible insertion places and let $\beta_1, \beta_2, \beta_3 \geq 0$ and $\beta_1 + \beta_2 + \beta_3 = 1$ to calculate IR_i with Equation (4.7):

$$IR_i = \frac{1}{|I_r|} \sum_{(u,v)\in I_r} \beta_1 z_{1i} + \beta_2 z_{2i} + \beta_3 z_{3i} \tag{4.7}$$

Algorithm 1 The IMPACT heuristic

```
Step 1: Initialization
Set furthest non-routed customer from depot as 'seed' customer. Start with partial route visiting
'seed' customer only.
Step 2: Selection
Among remaining non-routed customers, select customer $i$ that minimizes Impact($i$), i.e.:
  (a) For all feasible insertions of customer $i$, calculate Impact($i$). Select insertion location with
      minimum impact value.
  (b) Repeat (a) for all remaining non-routed customers.
  (c) Select customer $i$ and corresponding insertion location with minimum Impact($i$).
Step 3: Insertion
Add selected customer $i$ to the partial route at corresponding insertion location. Update route and
remove $i$ from non-routed customers.
If feasible non-routed customers for partial route remain, return to 2.
Step 4: Termination
If all customers are assigned to a route, STOP.
Otherwise, return to 1.
```

4.2 Pricing problem

The aim of the pricing problem is to find a feasible p-step with minimal reduced cost. To calculate the reduced cost corresponding to variable x_r, for each customer $i \in C$ we let dual variables κ_i, λ_i, μ_i and ν_i correspond to constraints (2), (3), (4) and (7), respectively. As these are not defined for the starting and ending depot, we define $\kappa_0 = \kappa_{n+1} = \lambda_0 = \lambda_{n+1} = \mu_0 = \mu_{n+1} = \nu_0 = \nu_{n+1} = 0$. Furthermore, we introduce dual variables ξ associated with constraints (8) for $\forall(i,j) \in A$. If we consider p-step $r = (P_r, d_r, t_r)$, with s the first and f the last node of P_r, the reduced cost $RC(r)$ is as follows:

$$\begin{aligned} RC(r) &= c_r - \sum_{i\in C} \kappa_i a_r^i - \sum_{i\in C} \lambda_i e_r^i - \sum_{i\in C} \mu_i v_r^i - \sum_{i\in C} \nu_i q_r^i - \sum_{(i,j)\in A} \xi_{ij} b_{ij}^r \\ &= \sum_{(i,j)\in P_r} (c_{ij} - \xi_{ij} - 2\kappa_j) - \kappa_s + \kappa_f - \lambda_s + \lambda_f - \mu_s t_r + \mu_f(t_r + T(r)) \\ &- \nu_s(d_r + q_s) + \nu_f(d_r + D(r)). \end{aligned} \tag{4.8}$$

To solve the problem of finding the p-step with minimal reduced cost, we follow the approach of Dollevoet et al. (2020) and decompose the pricing problem. We consider so-called (s,f)-pricing problems in which s and f are the fixed start and final node. In total, we construct $(n+1)n$ pricing problems as we do not consider problems with $s = f$ and problems with 0 as final node or $n+1$ as start node. Next, we model each separate problem as an ESPPRC in G such that the reduced cost of a p-step variable equals the total cost of a corresponding feasible (s,f)-path. Recall that to be feasible, the length of the path is at most p if $s = 0$, and exactly p otherwise. Furthermore, the cumulative demand of all customers on the path should not exceed the capacity, and it should be possible to visit all customers within their respective time windows.

To correctly define the reduced cost of a p-step variable, we compute a constant cost for each (s,f)-pricing problem and assign appropriate weights to each arc $(i,j) \in A$ such that the sum of this constant cost and all traversed arcs in P_r equals the reduced cost. Both constant and arc costs depend on whether the minimal or maximum value is assigned to the prior demand d_r and start time t_r. To determine these values for a p-step that does not start or end at the depot, we retrieve the linear coefficients of d_r and t_r in Equation (4.8): $(\nu_f - \nu_s)$ and $(\mu_f - \mu_s)$, respectively. Hence, we let prior demand d_r be 0 if $\nu_f \geq \nu_s$ and, otherwise, we assign maximum value $Q - D(r)$. Following the same reasoning, if $\mu_f \geq \mu_s$ we let t_r be the minimal starting time t_r^{early} and t_r^{late}, otherwise. We always assign minimal values when $s = 0$ and maximum values when $f = n+1$ (see Proposition 1).

The possible optimal values for d_r and t_r result in four different scenarios to compute costs for each (s, f)-pricing problem. With given values d_r and t_r, we derive the constant cost C_{sf} and the weights c'_{ij} from Equation (4.8). See Appendix A for the derived formulas per scenario. As we have now defined all (s, f)-pricing problems, the pricing problem comes down to finding a feasible elementary path in each problem that minimizes the total cost.

4.3 Preprocessing

Some pricing problems might be redundant, while others might contain unnecessarily many nodes and arcs. To gain a computational advantage in our algorithm, we apply various preprocessing steps to eliminate redundant (s, f)-pricing problems and reduce the size of the (partial) graphs in the ESPPRC.

The first preprocessing step reduces the size of graph G by removing arcs that will never be part of a (sub)optimal solution. For $(i, j) \in A$, we remove arc (i, j) from G if $a_i + s_i + t_{ij} > b_j$. In other words, customer j can never be visited after customer i if, from the earliest service moment possible a_i, the service and travel time exceed the latest starting time b_j. Moreover, we remove arc (i, j) from G if the demand of customers i and j together exceeds the capacity of a vehicle. Hence, we remove arc (i, j) if $q_i + q_j > Q$.

In the next preprocessing phase, we eliminate (s, f)-pricing problems which can never provide a feasible p-step. First, we apply a recursive depth-first search (DFS) algorithm to each (s, f)-pricing problem to check if any feasible elementary path P_r exists (see Algorithm 2). Note that any feasible path should not exceed the capacity Q, should visit all customers within their time windows and have at most length p if $s \neq 0$, and precisely length p, otherwise. In line 10 we check the length of the path and in line 12 we check if the capacity is not exceeded and if time windows are respected. If such feasible path P_r does not exist, we do not consider the respective (s, f)-pricing problem. Secondly, we remove (s, f)-pricing problems if the resulting p-step cannot be concatenated with any other feasible p-step. Generally, if there are no feasible p-steps arriving at customer s, we can remove all pricing problems departing at customer s. Similarly, we remove all (s, f)-pricing problems when no feasible p-steps depart from customer f.

In the third preprocessing step, we assign each remaining (s, f)-pricing problem a copy G_{sf} of the graph G and remove the unnecessary nodes and arcs within. In this way, we can specifically reduce the size of each partial graph G_{sf} and solve the pricing problems on this instead to secure computational advantage. First, we remove the nodes that can never be visited directly after s or directly before f. To do so, we use the remaining capacity of a vehicle function from Dollevoet et al. (2020). We define

Algorithm 2 Depth-first search

```
1: **Input:** Directed graph ($G$), current node ($v_i$), final node ($f$), length parameter ($p$), current partial path ($P$)
2: **Output:** Boolean if feasible path exists

3: Label $v_i$ as discovered and append $P$ with $v_i$

4: if $v_i == f$ then
5:   if $P[0] == 0$ and len($P$) $\leq p + 1$ then
6:     Return True
7:   else if $P[0] \neq 0$ and len($P$) $== p + 1$ then
8:     Return True

9: for $v_j$ in $G$.adjacent_edges[$v_i$] do
10:   if $v_j$ labeled as not discovered and len($P$) $\leq p$ then
11:     if Depth-first search($G$, $v_j$, $f$, $p$, $P$) then
12:       if Resource constraints not violated then
13:         Return True

14: Pop last node in $P$
15: Label $v_i$ as not discovered

16: Return False
```

this remaining capacity after visiting the customers in Y and the y customers with lowest demand as $Q(Y, y) = Q - \min\{\sum_{k \in S} q_k : S \subseteq C \backslash Y, |S| = y\}$, with $Y \subseteq N$ and $y \in \mathbb{N}_{>0}$. Now, we check if $Q(\{s, i, f\}, p - 2) < q_s + q_i + q_f$ for $p \geq 2$ and $s \neq 0$, i and f all different. If this is the case, we can eliminate node i from G_{sf} as not all three s, i, and f can be visited by a feasible p-step. We apply the same reasoning for the removal of arcs in G_{sf} and eliminate arc $(i, j) \in A$ if $p \geq 3$ and for different $s \neq 0$, i, j, and f it holds that $Q(\{s, i, j, f\}, p - 3) < q_s + q_i + q_j + q_f$.

5 Pulse algorithm

As the number of p-step variables in the p-step formulation is of order n^{p+1}, we could identify all feasible p-steps and solve the pricing problem, defined in Section 4.2, in polynomial time. However, this approach would explode in terms of computational time as higher values p cause the number of p-step variables to grow exponentially. Therefore, we introduce the exact pulse algorithm which we use to solve the ESPPRC in each (s, f)-pricing problem (Lozano et al., 2016).

As the name suggests, the pulse algorithm solves the ESPPRC by transmitting a pulse from the start node $s \in N \backslash \{n + 1\}$ to explore partial paths recursively. This exploration continues until a partial path is eliminated by a pruning strategy or reaches the given final node $f \in N \backslash \{0\}$. The algorithm includes the following two stages. The first is a bounding stage to compute lower bounds on the reduced cost that could be attained given a certain amount of time passed after leaving s. We assume here that we leave from s as early as possible and refer to this amount of time passed since leaving s as the time consumption. In the second stage, we actually send a pulse through the graph to recursively find an optimal solution by implicit enumeration of the solution space. Whilst propagating, at each node we store the current partial path P, the cumulative reduced cost $R(P)$, the cumulative demand $D(P)$ and the cumulative time consumption $T(P)$. In this way, every pulse that reaches the final node f contains all information on the solution to the respective (s, f)-pricing problem to provide a feasible p-step (Lozano et al., 2016). Below, we do not only present the general pulse algorithm (see Algorithm 3), but also elaborate on the bounding stage in Section 5.1 and the propagation stage with the different pruning strategies in Section 5.2.

Algorithm 3 General pulse algorithm

```
1: Input: Directed graph ($G_{sf}$), start node ($s$), final node ($f$), bounding time limits ($[\underline{t}, \overline{t}]$), bounding
   step size ($\Delta$), length parameter ($p$), primal bound ($\overline{r}$)
2: Output: Optimal path ($P^*$)

3: $P \leftarrow \{\}$
4: $R(P)$, $D(P)$, $T(P) \leftarrow 0, 0, 0$

5: Bounding scheme($G_{sf}$, $[\underline{t}, \overline{t}]$, $\Delta$, $p$, $\overline{r}$)                ▷ see Algorithm 4
6: Pulse propagation($\infty$, $s$, $P$, $R(P)$, $D(P)$, $T(P)$, $\overline{r}$)            ▷ see Algorithm 5

7: Return $P^*$
```

5.1 Bounding stage

In the bounding stage, we compute a three dimensional lower bound matrix $\mathbf{B} = [p_{val}, [\underline{r}(v_i, T(P))]]$ by finding conditional lower bounds for every value of 1 through p, every node $v_i \in C \cup \{0\}$ and for given discrete values of total time consumed $T(P)$. These lower bounds indicate the minimum reduced cost that a partial path P can achieve with remaining length parameter p_{val} when it arrives at node v_i with $T(P)$ time consumed already. In this bounding phase, we also consider time windows and the capacity of a vehicle. We construct $\mathbf{B}$ as follows (see Algorithm 4 for pseudocode).

First, we let $[\underline{t}, \overline{t}] = [a_s, b_f]$ denote the bounding time limits and let $\Delta > 0$ be a discrete time step. For $p_{val} = 1$, we solve an ESPPRC with a time consumption $T(P) = \overline{t} - \Delta$ for every node $v_i \in C \cup \{0\}$ by using pulse propagation (see Section 5.2). Stated differently, we search for a feasible path from v_i to

f with only Δ time units available with a length of at most p_{val} if $s = 0$, and exactly p_{val}, otherwise. In each case, the optimal solution found gives a lower bound on the minimum reduced cost which P with remaining length p_{val} can achieve when it arrives at v_i with at most $\bar{t} - \Delta$ leftover time to consume. Subsequently, we set the total time consumption $T(P) = \bar{t} - 2\Delta$ and again solve an ESPPRC for every node. We proceed in this backward iterative manner and solve until the time consumption $T(P)$ drops below the bounding time limit $\underline{t}$. If $p \neq 1$, we repeat above procedure for all values 2 through p to eventually return three dimensional lower bound matrix $\mathbf{B} = [p_{val}, [\underline{r}(v_i, T(P))]]$. Note that for the first steps, when given time consumption $T(P)$ is relatively high, the resulting ESPPRCs are overly constrained and the pulse propagation will find an optimal solution rapidly. In later stages, when the ESPPRCs are less constrained, we can use the previously computed lower bounds in $\mathbf{B}$ to prune paths and part of the solution space during propagation. Whilst pruning, we use the lower bound corresponding to the lower closest value to $T(P)$ from $\mathbf{B}$ (see also Section 5.2.2).

Algorithm 4 Bounding scheme

```
1: Input: Directed graph ($G_{sf}$), bounding time limits ($[\underline{t}, \bar{t}]$), bounding step size ($\Delta$), length parameter $p$, primal bound ($\bar{r}$)
2: Output: Lower bound matrix $\mathbf{B} = [p_{val}, [\underline{r}(v_i, \tau)]]$

3: for $p_{val}$ in range(1, $p$) do
4:   $\tau \leftarrow \bar{t}$
5:   while $\tau - \Delta > \underline{t}$ do
6:     $\tau \leftarrow \tau - \Delta$
7:     for $v_i$ in range(1, $n$) do
8:       $P \leftarrow \{\}$
9:       $R(P), D(P), T(P) \leftarrow 0, 0, \tau$

10:      if $v_i \neq v_f$ and $v_i$ in $G_{sf}$.nodes() then
11:        $P^* =$ Pulse propagation($\infty$, $v_i$, $P$, $R(P)$, $D(P)$, $T(P)$, $\bar{r}$)    ▷ see Algorithm 5

12:      if $v_i == f$ then
13:        $[p_{val} - 1, v_i, \tau] = -\infty$
14:      else if $P^* == \{\}$ then
15:        $[p_{val} - 1, v_i, \tau] = \infty$
16:      else
17:        $[p_{val} - 1, v_i, \tau] = R(P^*)$

18: Return B
```

To calculate a lower bound on the true reduced cost of the complete path from s to f, we need to be careful with the cost formulas (see Appendix A) and take into account the unknown partial path before node v_i. In the propagation stage, any path reaching v_i will have cost C_{sf} and costs c'_{ij} for each traversed arc (i, j) incurred. Hence, each lower bound in $\mathbf{B}$ consists of the weights c'_{ij} for all arcs (i, j) traversed from v_i to f. Note that correct (dual) variables are used with respect to start node s, and not v_i. The weight c'_{if} is dependent on t_r^{early} or t_r^{late} with corresponding l_{rf} as either the term $-\mu_s t_r^{early} + \mu_f l_{rf}$ or $-\mu_s t_r^{late} + \mu_f l_{rf}$ is included in the cost formula (see Appendix A). Therefore, to compute a valid lower bound we should take t_r^{early} or t_r^{late} as high as possible and l_{rf} as low as possible as we know that $\mu_s, \mu_f \geq 0$. Hence, we let $t_r^{early} = t_r^{late} = b_s$ and $l_{rf} = a_f$ in each case.

5.2 Propagation stage

As noted before, in this stage, we find the optimal solution in a recursive manner by implicit enumeration to limit the solution space. The pulse starts off at node $s \in C \cup \{0\}$ and explores the graph by trying to propagate throughout the adjacent nodes of each visited node. When a neighbouring node v_j is visited, three pruning strategies (see Sections 5.2.1, 5.2.2 and 5.2.3) check if it is possible and efficient to append the current partial path P with node v_j. If this is not the case, we aggressively limit the search space and stop the pulse from propagating. We backtrack and try to visit another adjacent node to current node v_i. On the other hand, suppose we do include v_j in P. In that case, we update and store all information on the cumulative reduced cost $R(P)$, the cumulative demand $D(P)$ and the cumulative time consumption $T(P)$, which includes all travel, service and waiting times. Hence, any node contains all available information of the feasible path P. The primal bound $\bar{r}$ gives the best solution found at any point in time. If a pulse is invoked on the final node f, we update the primal bound if the reduced cost of P is lower than $\bar{r}$. The current pulse is terminated, and the pulse propagation algorithm backtracks to try improve the optimal solution by exploring other pulses recursively. See Algorithm 5 for the pseudocode.

5.2.1 Feasibility pruning

The first pruning strategy discards infeasible partial paths by using structural constraints. Several conditions need to hold for a path to be feasible. First of all, whenever a partial path visits node v_i, it may not create a cycle. In other words, we check in constant time if node v_i is not already visited by the current partial path P. Next, we check if adding the demand q_i to the current cumulative demand $D(P)$ does not exceed the vehicle's capacity Q. Thirdly, we check if $T(P) \leq b_i$. If this is not the case, we prune the partial path as v_i cannot be visited after its latest possible service time. Finally, we check if the length of the partial path corresponds with length parameter p. We prune if, at any point in time, the length exceeds p. If the final node f is reached, we additionally check if the length

Algorithm 5 Pulse propagation

```
1: Input: Previous node ($v_h$), current node ($v_i$), current partial path ($P$), cumulative reduced cost ($R(P)$), cumulative demand ($D(P)$), cumulative time consumption ($T(P)$), primal bound ($\overline{r}$)
2: Output: Optimal path ($P^*$)

3: if Feasibility pruning($v_i$, $P$, $D(P)$, $T(P)$, $p$) == False then
4:     if Bounds pruning($v_i$, $f$, $R(P)$, $\mathbf{B}$, $\overline{r}$) == False then
5:         if Rollback pruning($v_i$, $P$, $R(P)$, $T(P)$) == False then
6:             $P^{'} \leftarrow P \cup \{v_i\}$
7:             $D(P^{'}) \leftarrow D(P) + q_i$
8:             if $v_h \neq \infty$ then
9:                 $T(P^{'}) \leftarrow \max\{a_i, T(P) + s_h + t_{hi}\}$
10:                $R(P^{'}) \leftarrow R(P) + c^{'}_{ij}$
11:            else
12:                $T(P^{'}) \leftarrow a_i$
13:                $R(P^{'}) \leftarrow C_{sf}$

14:            for $v_j$ in $G$.adjacent_nodes[$v_i$] do
15:                if $v_j$ not in P then
16:                    Pulse propagation($v_i$, $v_j$, $P^{'}$, $R(P^{'})$, $D(P^{'})$, $T(P^{'})$, $\overline{r}$)

17:            if $v_i == f$ and $R(P^{'}) < \overline{r}$ then
18:                $P^* \leftarrow P^{'}$
19:                $\overline{r} \leftarrow R(P^{'})$

20: Return $P^*$
```

is at most p if $s = 0$, and exactly p, otherwise.

5.2.2 Bounds pruning

To prune a partial path by bounds, we use the earlier computed three dimensional lower bound matrix $\mathbf{B} = [p_{val}, [\underline{r}(v_i, T(P))]]$ and the current primal bound $\overline{r}$. We first determine the remaining length $p_{val} = p - \text{len}(P)$ in contrast to parameter p. Next, we simply check in constant time if any partial path with length p_{val} from node v_i with available time $\overline{t} - T(P)$ can achieve a lower reduced cost than the current primal bound: $R(P) + [p_{val}, v_i, T(P)] \leq \overline{r}$. If this inequality does not hold, we prune the partial path. Note that we always use the lower closest value to $T(P)$ from $\mathbf{B}$.

5.2.3 Rollback pruning

As any pulse propagates through the graph in a recursive manner, the algorithm characterizes as a depth-first search. The main drawback of such a search is that unpromising regions of the solution space are searched after a deficient choice in the first stages. To prevent time-consuming backtracking, we use the rollback pruning strategy to reassure the last exploring step made. Note that we can only use this pruning method when $s = 0$, as in this case the length of the path is not fixed to $p + 1$.

Assume we have the following two paths $P_{sj} = \{v_s, v_i, v_k, v_j\}$ and $P'_{sj} = \{v_s, v_i, v_j\}$. See a graphical image in Figure 1. If the first path P_{sj} reaches node v_j during exploration, we evaluate if the choice of visiting v_k as well is efficient. To do so, we use the principle of dominating paths (Feillet et al., 2004). Path P'_{sj} dominates original path P_{sj} if it holds that $P'_{sj} \subseteq P_{sj}$, $R(P'_{sj}) \leq R(P_{sj})$, $D(P'_{sj}) \leq D(P_{sj})$, $T(P'_{sj}) \leq T(P_{sj})$, and if at least one holds strictly. By definition of our problem, we only need to check if $R(P'_{sj}) \leq R(P_{sj})$ and $T(P'_{sj}) \leq T(P_{sj})$ for path P'_{sj} to dominate path P_{sj}. Again, we verify in constant time whether these conditions hold and prune accordingly.

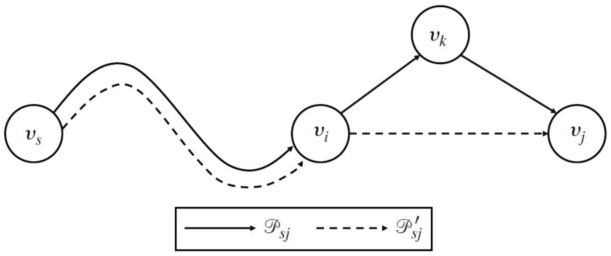

Figure 1: Graphical image of paths P_{sj} and P'_{sj} (Lozano et al., 2016)

6 Dive-and-fix heuristic

In Section 4 and 5, we described how to solve the LP relaxation of the p-step formulation. To solve the VRPTW to integrality in our experiments, we develop a dive-and-fix heuristic in which the proposed column generation approach and the pulse algorithm are slightly modified and combined with a fixed diving method. Below, we describe the initialization of our heuristic, the extended column generation algorithm and the branching procedure.

To initialize the dive-and-fix heuristic, we first apply the IMPACT heuristic to compute a feasible solution for the VRPTW instance (see Section 4.1) and follow the preprocessing procedure described in Section 4.3. After decomposing the solution of the VRPTW into feasible p-steps, we additionally generate p-steps with the DFS algorithm (see Algorithm 2) for each remaining (s, f)-pricing problem after preprocessing. Here, we take the first feasible path that reaches final node f. Following Proposition 1, if the (s, f)-pricing problem starts at the depot and does not end there, we generate p-steps of length 1 through p with both minimal prior demand and start time. If the problem does end at the depot, we do not generate the p-step of length 1 as at least one customer needs to be visited. If the (s, f)-pricing problem does not start or end at the depot, we create four p-steps of length p: one with

$d_r = 0$ and $t_r = t_r^{early}$, one with $d_r = 0$ and $t_r = t_r^{late}$, one with $d_r = Q - D(r)$ and $t_r = t_r^{early}$ and one with $d_r = Q - D(r)$ and $t_r = t_r^{late}$. Lastly, if the problem ends at the depot and did not start there, we create a p-step of length p with both maximum prior demand and start time.

After the initial p-steps are generated, we apply column generation to solve the LP relaxation at the root node in the branching tree. We solve the RMP and, subsequently, try to find negative reduced cost columns by solving each (s, f)-pricing problem individually. Instead of solving them directly with the proposed exact pulse algorithm, we first apply a greedy randomized adaptive search procedure with local search (GRASP). We construct a path with a greedy random insertion procedure. We start with s and construct a path by randomly adding one of its adjacent nodes. We only consider the K adjacent nodes v_j which have the lowest arc weight c'_{sj}. We repeat this process and terminate if f is reached or no feasible path is found. If we do find a feasible path, we try to improve the reduced cost by swapping a routed customer with a non-routed customer, if possible. When $s = 0$, we also try to lower the reduced cost by inserting or removing (non-)routed customers as the length of the p-step may vary in this case. We terminate the GRASP and continue with the column generation algorithm if we find a feasible path with negative reduced cost. If we do not find such path after I_{max} iterations, we proceed and apply the exact pulse algorithm.

We branch at the root node if the column generation algorithm finds a fractional solution for at least one of the arc-flow variables θ_{ij}. In this case, we sort all fractional θ_{ij} variables in descending order and fix the first $\alpha\%$ to value 1. We keep all columns and repeat the above procedure. Note that after each iteration, we can reduce the size of graph G_{sf} in each (s, f)-pricing problem as follows. For each variable θ_{ij} set to 1, we enforce arc $(i, j) \in A$ to be used. This means we can eliminate any other arc $(i, k) \in A$ for $k \neq j$ and any other arc $(h, j) \in A$ for $h \neq i$. The removal of arcs has no further consequences for the pulse algorithm. We continue this diving until either the relaxation is integer feasible or linearly infeasible.

7 Computational results

In this section, we present and discuss the computational results of our experiments with the p-step formulation. In Section 7.1, we introduce benchmark instances for the VRPTW from the literature and provide computation times of solving the LP relaxation in the root node. Thereafter, in Section 7.2, we evaluate the performance of the dive-and-fix heuristic on selected benchmark instances and specifically generated instances.

We implemented all algorithms in Python 3.8.8 and use CPLEX 20.1 as a solver for the RMPs in the column generation algorithm. The results are obtained by using an Intel(R) Core(TM) i7-10510U CPU processor, with 16 GB of RAM. We use the following values for the parameters in our algorithm. For the IMPACT heuristic, we adapt the values from Ioannou et al. (2001) and let $\alpha_1 = 0.1$, $\alpha_2 = 0.2$, $\alpha_3 = 0.7$, and $\beta_1 = \beta_2 = \beta_3 = \frac{1}{3}$. In our dive-and-fix heuristic we use $K = 3$ and $I_{max} = 15$ in the GRASP, and while diving we set $\alpha = 20$. Lastly, we use a time limit of 3600 seconds in all experiments.

7.1 Computation time LP relaxation

First, we present the computation times of solving the LP relaxation in the root node with our column generation algorithm for selected benchmark instances. In the literature, algorithms to solve the VRPTW are frequently tested on Solomon's VRPTW benchmark problems (Solomon, 1987). This classical set contains instances for which customer locations are randomly generated (problem sets R), clustered (problem sets C), or a combination of these two (problem sets RC). The instances differ in their respective time windows. Whereas in some problems the customers have very tight time windows and a short scheduling horizon, in other problems the scheduling horizon is broad and time windows of customers overlap and are hardly constraining.

If we compute the LP relaxation in the root node with our column generation algorithm, we encounter two counteracting effects which influence the computation time. On the one hand, larger values of p cut down the number of (s, f)-pricing problems to be solved from n^2 to one. On the other hand, increasing values of p cause the worst-case number of computations of the pulse algorithm to grow exponentially. As also the number of iterations until convergence influences the speed of the column generation algorithm, we do not necessarily believe that computation times increase with p.

In Table 1, we present the computation times of computing the LP relaxation in the root node of selected Solomon benchmark instances for different values of p. A variety of instances is chosen to evaluate how the density of time windows affects the run time of our column generation algorithm, such that we can specifically select suitable instances for further computational experiments. In the pulse algorithm, we let the bounding step size Δ depend on the scheduling horizon of the VRPTW instance. We let $\Delta = 100$ for all clustered instances with scheduling horizon $[0, 1236]$, and $\Delta = 20$ for the randomly generated and mixed problems with scheduling horizons $[0, 230]$ and $[0, 240]$, respectively. We use these values for the remainder of the experiments in this thesis. We have included the obtained LP bounds in Appendix B.

We observe that not all instances allow to compute the LP relaxation within the time limit with our column generation algorithm. Note that this is indicated by a dash in Table 1. Especially for values of p higher than 1, the clustered and mixed problem instances barely show results. In all cases, most time is spent during the execution of the exact pulse algorithm. As the exact part is determining for the speed of our algorithm, we expect that instances with wide and overlapping time windows are relatively harder to solve as the corresponding graphs are larger. Indeed, if we look into the instances that do solve the LP relaxation for different values of p within reasonable time (R101, R105, R109, RC101 and RC106), we observe that the time windows are tight and hardly overlap. Therefore, we select these five instances for the remaining experiments.

Table 1: Computation times in seconds of LP relaxation in root node (15 customers)

Instance	p :	1	2	4	6	$n+1$
C101		50.74	-	-	-	-
C103		316.40	-	-	-	-
C105		90.75	-	-	-	-
C106		33.93	-	-	-	-
C107		121.26	-	-	-	-
R101		23.64	97.05	238.91	160.12	
R102		239.02	-	-	-	
R105		195.69	392.08	1668.12	1446.69	
R109		299.47	3362.90	-	-	
R110		231.07	-	-	-	
RC101		216.01	1396.01	-	2129.53	2365.84
RC105		179.71	-	-	-	-
RC106		77.23	991.38	-	-	-
RC107		132.44	-	-	-	-
RC108		170.54	-	-	-	-

7.2 Performance of dive-and-fix heuristic

Our aim is to find empirical evidence of the potential computational advantages of using the p-step formulation to solve the VRPTW. Recall that our formulation is a generalization as, apart from intermediate values of p, it resembles arc-based formulations for $p = 1$, and the set partitioning formulation for $p = n + 1$ (Balinski and Quandt, 1964). Therefore, we identify potential advantage in computation time if there are instances for which the dive-and-fix heuristic solves to optimality for intermediate values of p with the lowest computation time.

7.2.1 Solomon benchmark instances

In Table 2, we provide the performance of our dive-and-fix heuristic for the five selected Solomon benchmark instances with different number of customers: $n = 10$, $n = 15$ and $n = 20$. For each instance, we compute the maximum number of customers a route could visit without exceeding the capacity of a vehicle or violating the time windows and denote this with c_{max}. If p is equal or higher than $c_{max}+1$, we obtain the set partitioning bound. Therefore, we only compute the lower and upper bounds for all values p from 1 through $c_{max}+1$. The lower bound is the objective value after the column generation algorithm has terminated. The upper bound is then obtained by applying the dive-and-fix heuristic. If the upper bound has value ∞, a linearly infeasible problem was obtained while diving. Again, a dash indicates that no solution was found before the time limit of 3600 seconds.

We observe that the column generation algorithm already solves all cases of instance R101 to optimality. For instance R105 with $n = 10$, our algorithm provides an optimality gap of 6.2% for all values of p except 1. However, for $n = 15$ and $n = 20$ the bounds converge to (near) optimality when p increases: from a gap of 1.2% to optimality and from 8.1% to 1.1%, respectively. Instances R109 with $n = 10$ and RC101 with $n = 10$ and $n = 15$ show a similar convergence to optimality. Even though we cannot draw meaningful conclusions for the remaining instances (R109 with $n = 15$ and $n = 20$, RC101 with $n = 20$ and all cases for instance RC106) as many cases did not solve within the time limit, we obtain better bounds and convergence to optimality for higher values of p, as expected.

Next, we look at the corresponding computation times of the dive-and-fix heuristic in Table 3. A dash indicates no solution was found within the time limit and the fastest computation times for each instance are displayed in bold. As instance R109 with $n = 20$ only solves for one value of p, we do not recognize a fastest time here. For every other instance, we observe that the fastest computation time corresponds with $p = 1$. In other words, from our experiments with the benchmark instances we cannot conclude a potential computational advantage for using the p-step formulation to solve the VRPTW. However, in general we do find better upper bounds for higher values of p.

Table 2: Lower and upper bounds of dive-and-fix heuristic

Instance	n	p :	1	2	3	4	5	6	7
R101	10	LB	241.50	241.50	241.50	241.50	241.50	241.50	
		UB	241.50	241.50	241.50	241.50	241.50	241.50	
		Gap (%)	0	0	0	0	0	0	
	15	LB	364.86	364.86	364.86	364.86	364.86	364.86	
		UB	364.86	364.86	364.86	364.86	364.86	364.86	
		Gap (%)	0	0	0	0	0	0	
	20	LB	470.94	470.94	470.94	470.94	470.94	470.94	
		UB	470.94	470.94	470.94	470.94	470.94	470.94	
		Gap (%)	0	0	0	0	0	0	
R105	10	LB	227.17	227.38	239.17	239.17	239.17	239.17	
		UB	∞	241.50	241.50	241.50	241.50	241.50	
		Gap (%)	-	6.2	6.2	6.2	6.2	6.2	
	15	LB	327.07	329.70	327.95	330.25	330.94	330.94	
		UB	330.94	330.94	330.94	330.94	330.94	330.94	
		Gap (%)	1.2	0.4	0.9	0.2	0	0	
	20	LB	400.26	400.62	403.84	-	-	-	
		UB	432.80	428.51	408.18	-	-	-	
		Gap (%)	8.1	7	1.1	-	-	-	
R109	10	LB	206.34	210.28	207.01	212.17	212.17	212.17	
		UB	∞	∞	∞	212.17	212.17	212.17	
		Gap (%)	-	-	-	0	0	0	
	15	LB	279.69	284.23	-	-	-	-	
		UB	∞	317.08	-	-	-	-	
		Gap (%)	-	1.2	-	-	-	-	
	20	LB	337.24	-	-	-	-	-	
		UB	∞	-	-	-	-	-	
		Gap (%)	-	-	-	-	-	-	
RC101	10	LB	180.20	180.39	181.47	180.97	180.78	182.83	182.83
		UB	∞	∞	182.83	∞	∞	182.83	182.83
		Gap (%)	-	-	0.7	-	-	0	0
	15	LB	212.01	212.19	213.28	-	212.57	214.60	214.63
		UB	∞	∞	214.63	-	∞	214.63	214.63
		Gap (%)	-	-	0.6	-	-	0	0
	20	LB	330.67	336.93	-	-	-	-	-
		UB	∞	∞	-	-	-	-	-
		Gap (%)	-	-	-	-	-	-	-
RC106	10	LB	154.18	157.27	160.16	-	-	-	-
		UB	∞	175.01	∞	-	-	-	-
		Gap (%)	-	1.5	1.3	-	-	-	-
	15	LB	159.85	162.81	-	-	-	-	-
		UB	∞	∞	-	-	-	-	-
		Gap (%)	-	-	-	-	-	-	-
	20	LB	233.14	264.99	-	-	-	-	-
		UB	∞	∞	-	-	-	-	-
		Gap (%)	-	-	-	-	-	-	-

Table 3: Computation times in seconds of dive-and-fix heuristic

Instance	n	p :	1	2	3	4	5	6	7
R101	10		**15.94**	24.01	47.46	59.14	72.28	52.04	
	15		**23.64**	97.05	144.97	238.91	226.12	160.12	
	20		**38.18**	116.11	117.17	335.08	620.89	1640.82	
R105	10		**41.51**	75.77	122.45	174.90	458.45	185.89	
	15		**195.69**	392.08	1696.38	1668.12	1178.55	1888.64	
	20		**553.92**	1754.57	3723.29	-	-	-	
R109	10		**125.31**	367.66	544.36	1522.25	1564.91	465.65	
	15		**299.47**	2623.91	-	-	-	-	
	20		299.96	-	-	-	-	-	
RC101	10		**64.22**	343.26	833.51	1523.74	384.56	750.99	286.76
	15		**216.01**	1396.01	2960.47	-	3414.72	2129.53	2365.84
	20		**282.31**	1648.51	-	-	-	-	-
RC106	10		**42.13**	662.55	1253.68	-	-	-	-
	15		**77.23**	991.38	-	-	-	-	-
	20		**685.93**	3243.15	-	-	-	-	-

7.2.2 Generated instances

As the Solomon benchmark instances do not provide evidence for the potential advantage of using the p-step formulation to solve the VRPTW, we generated specific instances that should favor intermediate values of p. We construct these instances as follows. We generate W clusters containing Q customers with unit demand each. In this way, vehicles with capacity Q can precisely visit and serve one complete cluster. We refer to these as (W, Q) instances. We assign the depot coordinate $(200, 200)$ and place each cluster such that its center is at distance 100 away from the depot. Then, we assign random coordinates to each customer within a square of dimension 3 around the corresponding cluster's center. Lastly, we set tight time windows: $[0, 240]$ for the depot and $[100, 140]$ for each customer.

In Table 4, we present the computation times of the dive-and-fix heuristic for five generated instances. As Q equals the maximum number of visits for each vehicle, we run each instance for the values of p from 1 through $Q + 1$. As before, a bold computation time indicates the fastest run-time for that instance. We identify four out of five instances which have a fastest computation time for an intermediate value of p: instance $(3, 3)$ with $p = 2$, instance $(3, 5)$ with $p = 5$, instance $(4, 3)$ with $p = 2$ and instance $(4, 4)$ with $p = 2$. The computation times cannot compete with the current state-of-the-art algorithms, but the experiments with generated instances show the potential of the p-step formulation in solving VRPTWs and merits further research to decrease computation times with specifically designed algorithms.

Table 4: Computation times in seconds of dive-and-fix heuristic

W	Q	p :	1	2	3	4	5	6
3	3		32.10	**21.03**	43.94	22.69		
	4		**58.92**	107.21	395.13	448.92	46.88	
	5		151.53	273.67	1864.08	3304.09	**137.43**	218.67
4	3		44.01	**39.65**	62.97	41.62		
	4		155.11	**142.21**	755.12	625.07	176.78	

8 Conclusion and discussion

In this thesis, we have presented a family of formulations for the VRPTW: the p-step formulation. By concatenating partial paths, of which the length is indicated by parameter p, we construct optimal routes with minimum cost to satisfy all customer demands within respective time windows. We consider this formulation to be a generalization as it not only provides new formulations for intermediate values of p, but also unifies traditional arc-based formulations for $p = 1$ and path-based formulations for $p = n + 1$. We proposed a column generation algorithm to solve the LP relaxation of the VRPTW, in which we decomposed the resulting pricing problems into a polynomial number of ESPPRCs. Next, we solved these problems with an exact pulse algorithm. Eventually, the introduced dive-and-fix heuristic solves until the relaxation is integer feasible or linearly infeasible.

The presented computational experiments with the Solomon benchmark instances do not show evident proof of the potential advantage of the p-step formulation. Out of fourteen instances, none showed the lowest computation time for an intermediate value of p. Therefore, we generated specifically designed clustered instances which are expected to favor intermediate values of p. Solving these generated problems with our dive-and-fix heuristic resulted in the lowest computation time for an intermediate value of p in four out of five instances. Even though computation times cannot compete with current state-of-the-art algorithms, we consider this empirical evidence that solving the VRPTW with the p-step formulation could potentially reduce computation time with algorithms designed specifically for this formulation. Note that classic arc-based and path-based formulations have been intensively studied over the past decades. In other words, to construct a competitive p-step formulation that exploits its computational advantage, we strongly recommend further algorithmic development.

A main drawback of our implementation is the relatively high computation time of the exact pulse algorithm. Lozano et al. (2016) show that their algorithm performs well compared to current state-of-the-art algorithms in solving the VRPTW, but our modified bounding stage to fit the p-step formulation implied a larger problem to solve. Instead of only one lower bound matrix, we need to compute multiple matrices for every value of 1 through p. Also, during the pulse propagation we check if the final node is reached after the recursive call to explore the graph further. We could have performed this check earlier to prevent the unnecessary visits to the neighbouring nodes of f.

Another aspect that influences the performance of our algorithm is the choice for the parameter values K, I_{max} and α in the dive-and-fix heuristic. In the GRASP, after some testing, we set relatively low values for K and I_{max} to avoid extreme randomness. Higher parameter values decrease the computation time of finding a negative reduced cost column in each (s, f)-pricing problem as the pulse algorithm is called less frequently. However, this does not decrease the total computation time of the column generation algorithm as more iterations need to be performed. In the choice of parameter value α, we saw in our experiments that a significantly lower value of α did not result in more feasible solutions being found. Therefore, taking the computation time into consideration, we chose a relatively larger value for α and leave the specific tuning of parameters for future research.

To advance the p-step formulation for the VRPTW and pursue its potential, several factors could be investigated in future studies. First of all, we break down the pricing problem into a number of well-known ESPPRCs, whereas other known or new problems could be more suitable. Moreover, in solving the pricing problems other bounding strategies or k-cycle elimination or ng-route relaxation might speed up the algorithm. Lastly, we only use one and the same algorithm for all values of p. It might be beneficial to design algorithms for specific values of p. Also, as such algorithms could potentially solve larger instances, future research might be able to draw more generic conclusions about the computational advantage of the p-step formulation.

Bibliography

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, 59(5):1269–1283.

Balinski, M. L. and Quandt, R. E. (1964). On an integer program for a delivery problem. *Operations Research*, 12(2):300–304.

Berthold, T. (2006). *Primal heuristics for mixed integer programs.* PhD thesis, Zuse Institute Berlin (ZIB).

Bräysy, O. and Gendreau, M. (2002). Tabu search heuristics for the vehicle routing problem with time windows. *TOP*, 10(2):211–237.

Chabrier, A. (2006). Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research*, 33(10):2972–2990.

Chiang, W.-C. and Russell, R. A. (1996). Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63(1):3–27.

Cook, W. and Rich, J. L. (1999). A parallel cutting-plane algorithm for the vehicle routing problem with time windows. Technical report.

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

Desaulniers, G., Lessard, F., and Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, 42(3):387–404.

Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40(2):342–354.

Dollevoet, T., Munari, P., and Spliet, R. (2020). A p-step formulation for the capacitated vehicle routing problem. Technical report.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, 42(5):977–978.

El-Sherbeny, N. A. (2010). Vehicle routing with time windows: An overview of exact, heuristic and metaheuristic methods. *Journal of King Saud University-Science*, 22(3):123–131.

Feillet, D., Dejax, P., Gendreau, M., and Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks: An International Journal*, 44(3):216–229.

Feillet, D., Gendreau, M., and Rousseau, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. *INFOR: Information Systems and Operational Research*, 45(4):239–256.

Günlük, O., Kimbrel, T., Ladanyi, L., Schieber, B., and Sorkin, G. B. (2006). Vehicle routing and staffing for sedan service. *Transportation Science*, 40(3):313–326.

Halse, K. (1992). *Modeling and solving complex vehicle routing problems.* PhD thesis, Technical University of Denmark.

Ioannou, G., Kritikos, M., and Prastacos, G. (2001). A greedy look-ahead heuristic for the vehicle routing problem with time windows. *Journal of the Operational Research Society*, 52(5):523–537.

Irnich, S. and Villeneuve, D. (2006). The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, 18(3):391–406.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008a). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, 56(2):497–511.

Jepsen, M. K., Petersen, B., and Spoorendonk, S. (2008b). A branch-and-cut algorithm for the elementary shortest path problem with a capacity constraint. Technical report.

Jörnsten, K., Madsen, O. B., and Sörensen, B. (1986). Exact solution of the vehicle routing and scheduling problem with time windows by splitting. Technical report.

Kallehauge, B. (2008). Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, 35(7):2307–2330.

Kallehauge, B., Boland, N., and Madsen, O. B. (2007). Path inequalities for the vehicle routing problem with time windows. *Networks: An International Journal*, 49(4):273–293.

Kallehauge, B., Larsen, J., and Madsen, O. B. (2001). Lagrangean duality applied on vehicle routing with time windows-experimental results. Technical report.

Kallehauge, B., Larsen, J., Madsen, O. B., and Solomon, M. M. (2005). Vehicle routing problem with time windows. In *Column generation*, pages 67–98. Springer.

Kohl, N. (1995). *Exact methods for time constrained routing and related scheduling problems.* PhD thesis, Technical University of Denmark.

Kohl, N., Desrosiers, J., Madsen, O. B., Solomon, M. M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, 33(1):101–116.

Kohl, N. and Madsen, O. B. (1997). An optimization algorithm for the vehicle routing problem with time windows based on lagrangian relaxation. *Operations Research*, 45(3):395–406.

Lenstra, J. K. and Rinnooy Kan, A. (1981). Complexity of vehicle routing and scheduling problems. *Networks*, 11(2):221–227.

Liu, X., Jiang, W., and Xie, J. (2009). Vehicle routing problem with time windows: a hybrid particle swarm optimization approach. In *2009 Fifth International Conference on Natural Computation*, volume 4, pages 502–506. IEEE.

Lozano, L., Duque, D., and Medaglia, A. L. (2016). An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, 50(1):348–357.

Mahmudy, W. F. (2014). Improved simulated annealing for optimization of vehicle routing problem with time windows (VRPTW). *Jurnal Ilmiah Kursor*, 7(3).

Mak, V. and Ernst, A. T. (2007). New cutting-planes for the time-and/or precedence-constrained ATSP and directed VRP. *Mathematical Methods of Operations Research*, 66(1):69–98.

Ombuki, B., Ross, B. J., and Hanshar, F. (2006). Multi-objective genetic algorithms for vehicle routing problem with time windows. *Applied Intelligence*, 24(1):17–30.

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, 51(3):155–170.

Rousseau, L.-M., Gendreau, M., Pesant, G., and Focacci, F. (2004). Solving VRPTWs with constraint programming based column generation. *Annals of Operations Research*, 130(1):199–216.

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

Toklu, N. E., Gambardella, L. M., and Montemanni, R. (2014). A multiple ant colony system for a vehicle routing problem with time windows and uncertain travel times. *Journal of Traffic and Logistics Engineering*, 2(1).

Toth, P. and Vigo, D. (2002). *The vehicle routing problem.* Society for Industrial and Applied Mathematics.

Zhu, Q., Qian, L., Li, Y., and Zhu, S. (2006). An improved particle swarm optimization algorithm for vehicle routing problem with time windows. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1386–1390. IEEE.

A Pricing problem: cost formulas

To compute the reduced cost for a p-step variable r in each (s,f)-pricing problem, we need a constant cost C_{sf} and weights c'_{ij} for each arc $(i,j) \in A$. Both computations depend on whether the minimal or maximum value is assigned to the prior demand d_r and start time t_r. Below, all four scenarios with their respective cost formulas are given. Note that we can calculate variables l_{rf}, t_r^{early} and t_r^{late} once we reach end location f.

If both the prior demand and start time are assigned the minimal value, $d_r = 0$ and $t_r = t_r^{early}$, we have:

$$C_{sf} = -\kappa_s + \kappa_f - \lambda_s + \lambda_f + (\nu_f - \nu_s)q_s \tag{A.1}$$

$$c'_{ij} = \begin{cases} c_{ij} - \xi_{ij} - 2\kappa_j - \mu_s t_r^{early} + \mu_f l_{rf} + \nu_f q_j, & \text{if } j = f, \\ c_{ij} - \xi_{ij} - 2\kappa_j + \nu_f q_j, & \text{otherwise.} \end{cases} \tag{A.2}$$

If prior demand is assigned the minimal value and start time the maximum value, $d_r = 0$ and $t_r = t_r^{late}$, we have:

$$C_{sf} = -\kappa_s + \kappa_f - \lambda_s + \lambda_f + (\nu_f - \nu_s)q_s \tag{A.3}$$

$$c'_{ij} = \begin{cases} c_{ij} - \xi_{ij} - 2\kappa_j - \mu_s t_r^{late} + \mu_f l_{rf} + \nu_f q_j, & \text{if } j = f, \\ c_{ij} - \xi_{ij} - 2\kappa_j + \nu_f q_j, & \text{otherwise.} \end{cases} \tag{A.4}$$

If prior demand is assigned the maximum value and start time the minimal value, $d_r = Q - D(r)$ and $t_r = t_r^{early}$, we have:

$$C_{sf} = -\kappa_s + \kappa_f - \lambda_s + \lambda_f + (\nu_f - \nu_s)Q \tag{A.5}$$

$$c'_{ij} = \begin{cases} c_{ij} - \xi_{ij} - 2\kappa_j - \mu_s t_r^{early} + \mu_f l_{rf} - \nu_s q_j, & \text{if } j = f, \\ c_{ij} - \xi_{ij} - 2\kappa_j - \nu_s q_j, & \text{otherwise.} \end{cases} \tag{A.6}$$

If both the prior demand and start time are assigned the maximum value, $d_r = Q - D(r)$ and $t_r = t_r^{late}$, we have:

$$C_{sf} = -\kappa_s + \kappa_f - \lambda_s + \lambda_f + (\nu_f - \nu_s)Q \tag{A.7}$$

$$c_{ij}^{'} = \begin{cases} c_{ij} - \xi_{ij} - 2\kappa_j - \mu_s t_r^{late} + \mu_f l_{rf} - \nu_s q_j, & \text{if } j = f, \\ c_{ij} - \xi_{ij} - 2\kappa_j - \nu_s q_j, & \text{otherwise.} \end{cases} \tag{A.8}$$

B LP bounds for benchmark instances

Table 5: LP bounds in root node (15 customers)

Instance	$p:$	1	2	4	6	$n+1$
C101		141.37	-	-	-	-
C103		102.23	-	-	-	-
C105		140.90	-	-	-	-
C106		141.37	-	-	-	-
C107		140.56	-	-	-	-
R101		364.86	364.86	364.86	364.86	
R102		262.33	-	-	-	
R105		327.07	329.70	330.25	330.94	
R109		279.69	284.23	-	-	
R110		242.60	-	-	-	
RC101		212.01	212.19	-	214.60	214.63
RC105		181.20	-	-	-	-
RC106		159.85	162.81	-	-	-
RC107		146.20	-	-	-	-
RC108		145.19	-	-	-	-