A pulse algorithm for column generation for a generalized vehicle routing formulation

Master Thesis Operations Research and Quantitative Logistics

Julian Rabbie 449730

July 21, 2018

Abstract

In this thesis we describe the use of an exact algorithm to solve a pricing problem arising from applying column generation in the context of the capacitated vehicle routing problem. A generalization of the vehicle flow and set covering formulations is used, known as the p-step formulation. This causes the corresponding pricing problem to be modeled as an elementary resource constrained shortest path problem, where the number of customers on a path is restricted.

To solve this pricing problem, we utilize an exact algorithm, known as the pulse algorithm. In particular, we use it to quickly identify negative reduced cost variables which can be added to the linear relaxation of the p-step formulation. Four pruning strategies are used which are mostly adapted to be specifically applicable to our pricing problem. The performance is tested using known benchmark problem instances in the literature and compared to a labeling algorithm.

The results show that the pulse algorithm is able to partially outperform the labeling algorithm in terms of both bounds and computation times. For one particular instance, a tighter bound can be obtained while requiring 41 times less computation time. However, when the allowed number of customers on a path increases to its maximum, the computation times of the pulse algorithm become intractable in contrast to the labeling algorithm. This suggests that multiple pricing algorithms might be required in order to obtain the overall best performance.

SUPERVISOR: DR. R. SPLIET

SECOND ASSESSOR: DR. T. A. B. DOLLEVOET

Erasmus School of Economics

ERASMUS UNIVERSITY ROTTERDAM

Contents

1 Introduction

Given a set of customers with demands, roads connecting the customers and a fleet of vehicles, the *Vehicle Routing Problem* (VRP) aims to find the optimal allocation of vehicles to sets of customers such that the costs are minimized, while visiting every customer exactly once. These costs can represent money, but also environmental factors like fuel consumption or the emission of greenhouse gases. The VRP is one of the most studied problems in the field of combinatorial optimization, due to its practical relevance and intrinsic difficulty.

Dantzig and Ramser (1959) published a report in which they stated a problem called the truck dispatching problem, concerning the distribution of fuel to gas stations. In the introduction of their paper, the two American mathematicians wrote that the problem they posed could also be referred to as the "Clover Leaf Problem", which is a rather elegant name resulting from the similarities of a typical solution of their problem and a clover leaf. Nevertheless, the problem they formally introduced has since then been referred to as the VRP[1]. This was a generalization of the *Traveling Salesperson Problem* (TSP), which was initially published a few years earlier by Flood (1956), where a single salesperson, instead of a fleet of vehicles, needs to visit the given locations.

In 2014, about 75% of all the transported goods in the European Union (EU) traveled by road, mostly using freight trucks (Eurostat et al., 2016). These trucks carried a staggering amount of 1499 billion tonne-kilometers[2] throughout the EU. Despite the economic crisis of 2008, this type of transport has kept growing every year, as well as transport by rail and water. The cumulative costs that are made in this sector are huge, so therefore even a small improvement of a few percent can make enormous differences in absolute terms. Moreover, with the rapid growth of automated vehicles and its applications like truck platooning (Harper et al., 2018), the need for efficient and fast transport optimization models does not tend to decrease in the (near) future.

Since its origin, a lot of research has been done in improving mathematical formulations for describing and solving the VRP. Every new model aims to either outperform or allow for more restrictions to be incorporated than previous ones. Recently, a new formulation of the VRP, called the p-step formulation, has been introduced by a collaboration of Brazilian and Dutch operations researchers. The exciting feature of this formulation is the fact that it (together with another formulation) outperforms any previous compact formulation in terms of its bounds.

In this thesis, we focus on solving the pricing problem arising from the application of column generation to the p-step formulation. In particular, we implement a pulse algorithm in order to solve this pricing problem, which can be modeled as a *Resource Constrained Elementary Shortest Path Problem* (RCESPP), where the number of customers on the path is restricted.

We will start with a literature review of the VRP in Section 2 and in particular focus on two

[1] Technically, the problem in Dantzig and Ramser (1959) was the *Capacitated Vehicle Routing Problem*.

[2] One tonne-kilometer denotes the transport of one tonne of goods over a kilometer.

classes of extensively researched formulations as well as a brief description of solution methods for the RCESPP. Next, in Section 3, we will elaborate on the mathematical description of the VRP and furthermore we will formally state the p-step formulation. Section 4 covers the methodology with which we intend to solve problem instances by using the p-step formulation. Here we also describe the pricing problem and the pulse algorithm in detail, including the adaptations we apply to the latter in order to make it fitting for our pricing problem. In order to analyze the performance of the pulse algorithm, in Section 5 we use a set of known benchmark instances from scientific literature to compute lower bounds and compare these against those obtained by another exact algorithm. Finally, in Section 6 we draw conclusions from our research and propose some topics which can be investigated in future research.

2 Literature Review

Since its introduction, many research has been done in finding suitable formulations for the VRP, see for example the book by Vigo and Toth (2015). As a vehicle that needs to be routed will always have a certain limited capacity, in general the *Capacitated Vehicle Routing Problem* (CVRP) is considered for practical applications. There is a large number of other extensions for the VRP, like the presence of delivery time windows or customers that need their goods to be picked-up instead of delivered. The VRP and all of its extended forms belong to the class of NP-hard combinatorial optimization problems, which makes them intrinsically interesting for mathematicians to investigate.

In general, there are two main classes of mathematical models for the standard CVRP: *vehicle flow* formulations and *set partitioning* formulations. They both have their advantages and disadvantages in terms of complexity and bounds, which we will elaborate on in this section.

2.1 Vehicle Flow Formulation

One approach of solving the CVRP is to solve the problem step-by-step and determine per arc whether it should be used or not. The formulation arising from this approach is called the vehicle flow formulation (Laporte et al., 1986).

The benefit of this formulation is that it has a polynomial number of variables in terms of the number of customers (and vehicle fleet size). The disadvantage arises from the need for an exponential number of *Generalized Subtour Elimination Constraints* (GSECs), which impose that for each subset $\mathcal{S}$ of customers with cumulative demand $q(\mathcal{S})$ we need at least $\left\lceil \frac{q(\mathcal{S})}{Q} \right\rceil$ vehicles, where Q is the capacity of a single vehicle. This should hold for all subsets of customers, hence the exponential total number of constraints. These constraints can be rewritten into a linear number of constraints using for example the MTZ-formulation (Miller et al., 1960). The resulting formulation is compact, which means that both the number of variables and constraints scale polynomially with the number of customers. This is a convenient property to have for formulations, since problem instances with a large number of customers will not tend the formulation to explode in size. However, the MTZ-formulation requires the introduction of additional variables and, more importantly, results in significantly weaker lower bounds. This illustrates a typical trade-off between the tightness of the bounds and the size of the corresponding formulation.

Another approach in finding bounds for formulations with an exponential number of constraints can also be found by dropping the GSECs altogether, such that the problem reduces to the TSP, which can be used to construct lower bounds (Little et al., 1963). These rely on finding a maximum matching in a modified version of the graph. The bounds for the VRP that are determined by using this approach are generally pretty weak.

Tighter bounds can be found by relaxing the GSECs in a Lagrangian way, after which a sub-

gradient optimization procedure can be used to determine bounds. However, the exponential cardinality of the set of GSECs disallows the explicit inclusion of all of them into the objective function in practice. This brings the requirement of heuristics to determine which constraints to initially include and add hereafter. Even though the bounds that are found by using this approach tend to be much tighter, the computation times also significantly increase.

Extensive research has also been done in the use of branch-and-cut algorithms (Laporte et al., 1985; Ralphs et al., 2003; Lysgaard et al., 2004). This approach uses a branch-and-bound scheme where the GSECs are initially removed and similar suitable restrictions are added sequentially.

Although the improvement of the *Linear Programming* (LP) bound of vehicle flow formulations remains to have practical value for real-life applications, the focus in more recent years has been on (improving) the second type of commonly used VRP formulations.

2.2 Set Partitioning Formulation

Another way of solving the CVRP might be to look at the considered graph as a whole and determine in which way it should be partitioned in order to construct a set of feasible routes that form an optimal solution. This results in the set partitioning formulation, first proposed by Balinski and Quandt (1964).

This formulation has a polynomial number of constraints, since the feasibility of the routes is implicitly required, thus there is no necessity for GSECs. The downside of this formulation is that the number of feasible routes scales exponentially with the input size. This also brings an exponential number of binary decision variables, so the formulation is not compact.

Under certain conditions[3], this formulation can also be rewritten into a set covering formulation, where customers can be visited more than once. Any solution of the set covering formulation can be transformed into a feasible solution of the set partitioning formulation with the same or lower costs (Vigo and Toth, 2015). This can considerably reduce the computation time for finding the optimal solution, but not enough for it to be practical to solve the whole problem at once for a large number of customers. Therefore, column generation approaches are used for solving CVRPs using this formulation, for example in Bramel and Simchi-Levi (2002). When this is further extended by using robust cuts in branch-and-price-and-cut schemes, it is possible to compute tight lower bounds for large problem instances (Fukasawa et al., 2006; Baldacci et al., 2008; Pessoa et al., 2008).

Extensive research is still done in investigating stronger valid inequalities for making the LP bounds of either one of the formulations stronger (Yaman, 2006; Leggieri and Haouari, 2017), as they are seen as rather dissimilar. However, the formulation used in this thesis can be seen as a generalization of both, where the vehicle flow and set partitioning formulation arise in two extreme cases.

[3] In particular, the triangle inequality in terms of costs (see Section 3.1) should hold.

2.3 The Resource Constrained Elementary Shortest Path Problem

In order to solve CVRP instances with this generalized formulation, we will use column generation scheme. The pricing problem arising from applying column generation can be modeled as a RCESPP, which is an NP-hard optimization problem (Dror, 1994). This problem also occurs as pricing problem for column generation in the context of crew scheduling and flight planning problems. A common approach for solving the RCESPP is by using dynamic programming label setting algorithms, with Desrochers et al. (1992) being the first ones to apply such an algorithm to column generation in vehicle routing. Due to the fact that the reduced costs do not necessarily have to be positive, negative cost cycles might exist in such networks, hence the elementarity of the paths. However, for elementary routes, label setting algorithms tend to have an exponential running time of $O\left(2^n Q\right)$, where n is the number of customers, i.e., the input size. The elementarity can be partially relaxed by using for example k-cycle elimination (Fukasawa et al., 2006) or ng-route relaxation (Baldacci et al., 2011). These significantly decrease the running times but simultaneously reduce the tightness of the LP bounds.

More recently, Lozano and Medaglia (2013) introduced a new exact algorithm for solving shortest path problems, referred to as the pulse algorithm. This algorithm is quite similar to branch-and-bound, as all paths are implicitly enumerated, while infeasible or uninteresting ones are pruned as soon and effectively as possible. Lozano et al. (2015) applied the pulse algorithm to column generation for the vehicle routing problem with time windows, using a set covering formulation. Besides the RCESPP, the pulse algorithm is also applicable to a wide range of shortest path problems, such as the biobjective shortest path problem (Duque et al., 2015a) and the orienteering problem with time windows (Duque et al., 2015b).

The main additional difficulty of the RCESPP considered in this thesis is the fact that the length of the path is either bounded from above or fixed beforehand. We will thus investigate whether the pulse algorithm is also suitable for this form of the shortest path problem.

3 Problem Description

In this section, we will mathematically describe the CVRP in more detail, and in particular focus on the recently developed p-step formulation.

3.1 The Capacitated Vehicle Routing Problem

The CVRP is the problem of finding a set of routes that visit a set of customers exactly once, using a fleet of vehicles which should all start and end at the depot, under the restriction that the cumulative demand of customers in a route may not exceed the vehicle capacity. Let $\mathcal{G} = (\mathcal{N}, \mathcal{E})$ be a complete graph, where $\mathcal{N} = \{0, \ldots, n+1\}$ is the set of nodes and $\mathcal{E} = \{(i,j) : i,j \in \mathcal{N}, i \neq j\}$ is the set of undirected edges which connect the nodes. Here, the set $\mathcal{C} = \mathcal{N} \setminus \{0, n+1\}$ are the n customers and 0 and $n+1$ represent the starting and ending depot respectively. We assume that there is a single depot in all of our problem instances and therefore 0 and $n+1$ implicitly represent the same node. Each customer $i \in \mathcal{C}$ has demand $q_i \geq 0$ and for convenience we let $q_0 = q_{n+1} = 0$. The cost of using edge (i,j) is given by c_{ij}. Since we use an undirected graph, the costs are symmetric, i.e., $c_{ij} = c_{ji}$. This form of the CVRP is commonly used in benchmark instances that we can use to generate our results and allows us to compare them to other methods. Furthermore, we assume that the triangle inequality in terms of costs holds, which implies that $c_{hi} + c_{ij} \geq c_{hj}$, for all $(h,i), (i,j), (h,j) \in \mathcal{E}$.

Next, let K represent the number of available vehicles. Each vehicle that leaves the depot visits a (sub)set of customers $\mathcal{S} \subseteq \mathcal{C}$ sequentially, which forms a route $\rho(\mathcal{S})$. In this route, all customers in $\mathcal{S}$ must be visited exactly once and the vehicle also has to end its route at the depot. A route has costs $c(\rho(\mathcal{S})) = \sum_{(i,j)\in\rho(\mathcal{S})} c_{ij}$. For a given graph, a lower bound on the number of vehicles that should be considered for constructing a feasible solution can be found by solving a bin-packing problem (Toth and Vigo, 2002).

The fleet of vehicles is furthermore assumed to be homogeneous, i.e., each vehicle is assumed to have the same finite capacity $Q > 0$. This capacity limits the number of customers that can be served by a single vehicle. We assume, without loss of generality, that $q_i \leq Q$ $\forall i \in \mathcal{C}$. For a route $\rho(\mathcal{S})$ to be feasible, it should hold that $\sum_{i\in\mathcal{S}} q_i \leq Q$. An illustration of what a typical optimal solution of the CVRP might look like is displayed in Figure 3.1. In this example there is a single depot and 16 customers. Note that the solution shares some resemblances to a clover leaf, as mentioned in Section 1.

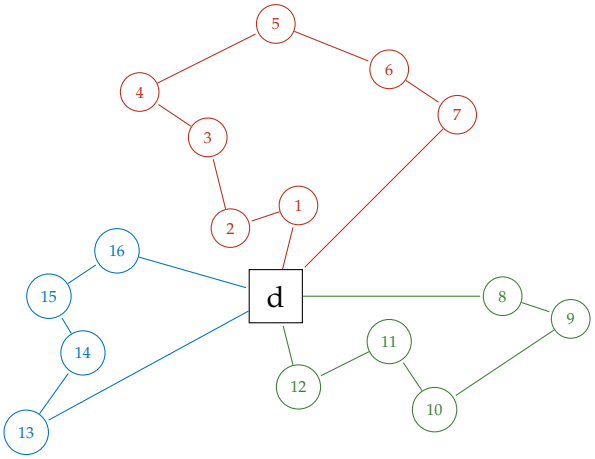


Figure 3.1: Illustration of the routes of a CVRP solution for a single depot, 16 customers and three vehicles. Here, it holds that $\sum_{i=1}^{7} q_i$, $\sum_{i=8}^{12} q_i$ and $\sum_{i=13}^{16} q_i$ are smaller than or equal to Q. Furthermore, only the arcs used in a solution are shown, while for a complete graph all nodes are interconnected. Note that the depot d could also be represented by both node 0 and 17.

Here, the set of customers is partitioned in three subsets such that the cumulative demand of the customers in such a subset does not exceed the vehicle capacity. This is also done in such a way that the total costs of the traversed arcs by the vehicles is minimized. The complexity of the CVRP hence results from the combination of partitioning the set of nodes and finding a least cost path within a subset.

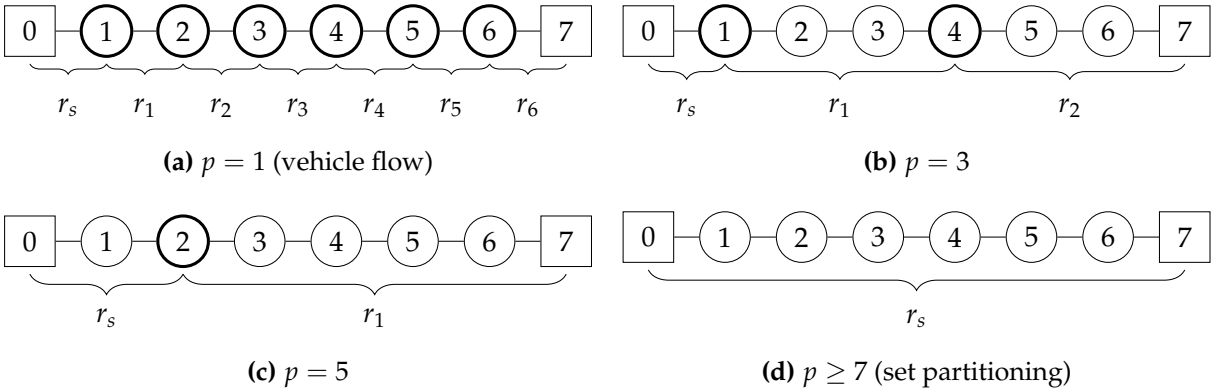
3.2 The p-step Formulation

The p-step formulation combines the power of the vehicle flow and set partitioning formulations by providing an adjustable balance between the tightness of the LP bound and the number of decision variables. Furthermore, it is an intuitive and compact formulation which can easily be visualized. At the time of writing, the p-step formulation is still being developed by P. Munari[4], T. Dollevoet[5] and R. Spliet[6]. For a first introductory paper about this method, we refer the reader to Munari et al. (2016).

Let us introduce the formulation by defining a p-step r, which consists of a pair (P_r, d_r), where P_r is the path and d_r is the starting load when arriving at the first node of P_r. Furthermore, let $\mathcal{R}^p$ be the set containing all feasible p-steps. With a single p-step or multiple consecutive p-steps, a route can be formed in a given graph. Each path P_r should then consist out of exactly p arcs, where p is integer. An exception is made for the first path of the route that starts at 0, which can contain p' arcs, with $p' \in \{1, \ldots, p\}$. An illustration of some routes which can be formed by linking a sequence of p-steps is shown in Figure 3.2 for several values of p. Here, we illustrate a network consisting of the starting depot 0 and the final depot 7, with

[4] Federal University of São Carlos, Brazil. munari@dep.ufscar.br
[5] Erasmus University Rotterdam, the Netherlands. dollevoet@ese.eur.nl
[6] Erasmus University Rotterdam, the Netherlands. spliet@ese.eur.nl

6 customers nodes in between.



(a) $p = 1$ (vehicle flow) **(b)** $p = 3$

(c) $p = 5$ **(d)** $p \geq 7$ (set partitioning)

Figure 3.2: Illustration of a p-step for several values of p. Here, r_s denotes the starting step, which can contain less than p arcs and r_i, $1 \leq i \leq 6$ are the subsequent regular p-steps. The square nodes illustrate the depots and the thick nodes indicate the nodes that form a link between two p-steps.

Note that, as mentioned in the captions of Figures 3.2a and 3.2d, for certain values of p, the problem reduces to the two common formulations for the CVRP as mentioned in Sections 2.1 and 2.2. For $p = 1$, the formulation reduces to a problem for which a decision has to be made per arc whether to use it or not, which is similar to the vehicle flow formulation. Solving a problem instance for this value of p will results in a relatively weak LP bound. On the other side, when $p \geq n + 1$, a single p-step covers a feasible route in the network. This indicates that the decision has to be made which combination of routes should be used such that the customers are partitioned in an optimal way, hence the resemblance to the set partitioning formulation. The latter yields a higher LP bound, although requiring exponentially many variables. The p-step formulation can therefore be seen as a generalization of both, where a trade-off between the strength of the LP bound and the number of decision variables can be made by tuning the value of p.

3.2.1 Mathematical Description

Recall that we use a complete graph $\mathcal{G}(\mathcal{N}, \mathcal{E})$ to describe the network. For every p-step r, let a_r^i be the degree of customer i in P_r, such that for every visited customer this parameter has value 2, while it has value 1 for the first and last node in P_r and 0 otherwise. For the coupling of the p-steps, we introduce the linking parameters e_r^i and q_r^i. Here, e_r^i is an indicator parameter which is 1 if i is the first node of P_r, -1 if i is the last node of P_r and 0 otherwise. The load parameter q_r^i is a measure for the cumulative demand satisfied up until customer i in P_r. It has value $d_r + q_i$ if i is the first customer, $-\left(d_r + \sum_{i \in P_r} q_i\right)$ if i is the last customer of P_r and 0 otherwise. Finally, let b_r^{ij} be 1 if edge $(i, j) \in \mathcal{E}$ is used in r and 0 otherwise.

The first decision variables for this formulation are x_r, which indicate what fraction of p-step r with corresponding cost c_r is used, where c_r is given by $\sum_{(i,j) \in P_r} c_{ij}$. The second variable is the binary variable θ^{ij}, which indicates whether edge $(i, j) \in \mathcal{E}$ is used. The p-step formulation

(PSF) is as follows:

$$
\begin{aligned}
(PSF) \quad \min \quad & \sum_{r \in \mathcal{R}^p} c_r x_r && && (3.1)\\
\text{s.t.} \quad & \sum_{r \in \mathcal{R}^p} a_r^i x_r = 2 && \forall i \in \mathcal{C} && (3.2)\\
& \sum_{r \in \mathcal{R}^p} e_r^i x_r = 0 && \forall i \in \mathcal{C} && (3.3)\\
& \sum_{r \in \mathcal{R}^p} q_r^i x_r \geq 0 && \forall i \in \mathcal{C} && (3.4)\\
& \sum_{r \in \mathcal{R}^p} a_r^0 x_r = K && && (3.5)\\
& \sum_{r \in \mathcal{R}^p} b_r^{ij} x_r = \theta^{ij} && \forall (i,j) \in \mathcal{E} && (3.6)\\
& x_r \in [0,1] && \forall r \in \mathcal{R}^p && (3.7)\\
& \theta^{ij} \in \{0,1\} && \forall (i,j) \in \mathcal{E} && (3.8)
\end{aligned}
$$

The objective function (3.1) minimizes the total costs. Constraints (3.2) ensure that each visited customer has degree equal to 2. Constraints (3.3) are used to link the final node of a p-step to the first of another one. Next, Constraints (3.4) make sure that only feasible p-steps in terms of customer demand are allowed to be linked. Since we have a finite number of vehicles, Constraint (3.5) equates the number of p-steps leaving the depot to the fleet size. Since we cannot use an edge more than once, we impose Constraints (3.6), which also form the link between the two types of decision variables. Finally, Constraints (3.7) and (3.8) restrict the feasible domain of these decision variables. Note that x_r are not binary variables and we thus allow convex combination of p-steps with the same paths and different starting loads to occur in a solution. The elementarity of the arcs is however still guaranteed by (3.6) and (3.8). The number of decision variables is $O\left(n^{p+1}\right)$ and since we also have a polynomial number of constraints, the p-step formulation is compact.

In order to solve a CVRP instance by using the p-step formulation, we do not necessarily need every single variable. Munari et al. (2018) list some properties of this formulation which reduce the total number of variables by deducing which variables can be considered redundant. We list the properties of redundant p-steps in Appendix A. One important property which we would like to emphasize here is that one can show that it is sufficient to only consider the starting load d_r to have either a minimum value of 0 or some maximum value, which greatly reduces the amount of required p-step variables.

4 Methodology

Although the number of feasible p-steps is polynomial, it can still become huge for large values of p. We will therefore use column generation to solve CVRPs using the p-step formulation. We will first give a general description of this method, after which we will derive the pricing problem for (PSF) in particular. Finally, we will describe the exact algorithm that we utilize for solving this pricing problem.

4.1 Column Generation

When a problem instance becomes too large to solve at once, a divide-and-conquer strategy can be used by decomposing the problem into smaller ones which can be solved independently in order to solve the initial problem. The best example of this strategy in combinatorial optimization is column generation. It was first introduced by Gilmore and Gomory (1961) in the context of cutting stock, after which it has been applied to a large number of other problems. The idea is to start by considering a (small) subset of decision variables after which promising ones are sequentially added until all variables required for the optimal solution are found. In the context of p-steps, we thus start with a subset of the feasible p-steps $\widetilde{\mathcal{R}}^p \subset \mathcal{R}^p$. Next, we can solve the LP relaxation of this initial problem with this subset of variables, which is referred to as the *Restricted Master Problem* (RMP). The solution of the RMP can subsequently be used as a lower bound in a branch-and-bound scheme. For the p-step formulation, the RMP is obtained by relaxing (3.8) such that it becomes

$$\theta^{ij} \in [0,1] \qquad \forall (i,j) \in \mathcal{E}.$$

With the dual information of a solution of the RMP, we can determine whether there are p-steps $r \in \mathcal{R}^p \setminus \widetilde{\mathcal{R}}^p$ with negative reduced costs which we can add to $\widetilde{\mathcal{R}}^p$ in order to reach a lower objective value. If no such p-step exists, one can prove that all required variables for the optimal solution of the LP relaxation are contained in $\widetilde{\mathcal{R}}^p$ and thus the global optimal solution can be constructed. In order to find p-steps with negative reduced costs, we need to solve a subproblem called the pricing problem. The combination of column generation and branch-and-bound is therefore also called branch-and-price.

Munari et al. (2018) provide us with some useful theory on the LP bounds of the p-step formulation, which we can use to verify the correctness of our results. For completeness, we list these properties in Appendix B. In short, the LP bound of any problem instance for a given value of p should always be equal to or lower than the LP bound of the same instance with mp, where m is an integer larger than or equal to 2. This leads to the observation that $p = 1$ will always give the weakest bound and $p = n + 1$ will always lead to the tightest LP bound. Note that this also means that choosing $p > n + 1$ becomes redundant.

4.1.1 Pricing Problem

Let λ, μ, ν, ξ and π be the dual variables corresponding to constraints (3.2) - (3.6). For convenience, we define $\lambda_k = \mu_k = \nu_k = 0$ for $k \in \{0, n+1\}$. The reduced cost $RC(r)$ of a given, feasible p-step $r \in \mathcal{R}^p \setminus \widetilde{\mathcal{R}}^p$ is

$$\begin{aligned} RC(r) &= \sum_{(i,j)\in P_r} (c_{ij} - b_r^{ij}\pi_{ij}) - \sum_{i\in P_r} \left(a_r^i\lambda_i + e_r^i\mu_i + q_r^i\nu_i\right) - a_r^0\xi \\ &= \sum_{(i,j)\in P_r} (c_{ij} - \pi_{ij}) + \lambda_f - \lambda_s + \mu_s - \mu_f - \nu_s(d_r + q_s) + \nu_f(d_r + \sum_{i\in P_r} q_i) - \mathbb{1}_{\{s=0\}}\xi. \end{aligned} \tag{4.1}$$

Here, $\mathbb{1}_{\{s=0\}}$ is an indicator function which has value 1 if $s = 0$ and 0 otherwise. It holds that for every p-step, it is always optimal to set d_r either equal to 0 or some maximal value. For a fixed starting node s and final node f ($s \neq f$) of P_r, this maximal value is denoted by $Q_{sf} - \sum_{i\in P_r} q_i$, where

$$Q_{sf} = \begin{cases} Q, & \text{if } s \in 0 \cup \mathcal{C}, f = n+1 \\ Q(\{f\}, p-1), & \text{if } s = 0, f \in \mathcal{C} \\ Q(\{s,f\}, p-1), & \text{if } s \in \mathcal{C}, f \in \mathcal{C} \end{cases}$$

Here, $Q(\mathcal{Y}, y) = Q - \min\{\sum_{i\in\mathcal{S}} q_i : \mathcal{S} \subseteq \mathcal{C} \setminus \mathcal{Y}, |\mathcal{S}| = y\}$ is the remaining capacity of a vehicle that visits the y customers with the lowest demand, excluding the customers in $\mathcal{Y}$. Note that (4.1) has a linear term in d_r with coefficient $\nu_f - \nu_s$, so when $\nu_s \geq \nu_f$, it is optimal to set d_r to its maximal value and 0 otherwise.

Furthermore, we can determine whether the pricing problem (s, f) or its reverse, (f, s), will yield a variable with the lowest reduced cost for $s \neq 0$ and $f \neq n+1$. Let r be a feasible p-step variable with path P_r, which does not start nor end at the depot, and starting load d_r. Next, let r' be the p-step variable with reverse path $P_{r'}$ and starting load $d_{r'}$. Assume, without loss of generality, that $\nu_s \geq \nu_f$, such that $d_r = Q_{sf} - \sum_{i\in P_r} q_i$ and $d_{r'} = 0$. The difference in reduced costs is given by

$$RC(r) - RC(r') = 2\mu_f - 2\mu_s - \nu_s q_s + \nu_f q_f + (\nu_f - \nu_s)Q_{fs}. \tag{4.2}$$

When (4.2) is smaller than 0, we can conclude that r will be the variable with the lowest reduced costs and otherwise, r' will be the most promising variable. When they are equal, we stick to solving the (s, f) pricing problem. This allows is to halve the number of pricing problems to solve that do not start nor end at the depot.

Finally, we can decompose (4.1) into a variable cost per arc c'_{ij} for all arcs $(i, j) \in P_r$ and fixed costs C_{sf}, where

$$c'_{ij} = c_{ij} - \pi_{ij} - 2\lambda_j + \alpha_1 q_j, \tag{4.3}$$

$$C_{sf} = \lambda_f - \lambda_s + \mu_f - \mu_s - \mathbb{1}_{\{s=0\}}\xi + (\nu_f - \nu_s)\alpha_2. \tag{4.4}$$

Here, the values of α_1 and α_2 depend on which value for d_r is used. In particular,

$$\begin{cases} \alpha_1 = \nu_f \text{ and } \alpha_2 = q_s, & \text{if } d_r = 0 \\ \alpha_1 = \nu_s \text{ and } \alpha_2 = Q_{sf}, & \text{if } d_r = Q_{sf} - \sum_{i \in P_r} q_i. \end{cases}$$

The reduced costs of a p-step can now be computed by evaluating

$$C_{sf} + \sum_{(i,j) \in P_r} c'_{ij}. \tag{4.5}$$

Additionally, for a given instance, we can compute an upper bound on the number of traversed arcs p, solely based on the demand of the customers. This is defined by

$$\bar{A} = \max \left\{ 1 + |\mathcal{S}| : \sum_{i \in \mathcal{S}} q_i \leq Q, \mathcal{S} \subseteq \mathcal{C} \right\},$$

which we can use to reduce the search depth for paths the pulse algorithm investigates for different (s, f) pricing problems. In particular, we use the following upper bounds on the number of arcs $\bar{a}$ to be traversed:

$$\bar{a} = \begin{cases} \min\{p, \bar{A}\} & \text{if } s = 0, f = n+1 \\ \min\{p, \bar{A} - p\} & \text{if } s = 0, f \in \mathcal{C} \\ p & \text{if } s \in \mathcal{C}, f \in \mathcal{C} \end{cases} \tag{4.6}$$

Note that $\bar{a}$ is only used for our implementation of the algorithm, while we keep using p to denote the maximum number of arcs for consistency in the following sections. Since s (f) can be any node, except for $n+1$ (0), there are $O(n^2)$ pricing problems in total. The goal is to find a p-step for which (4.5) is negative, which we subsequently add to the RMP. Finding a feasible negative reduced cost p-step can be done by solving an RCESPP.

4.2 The Pulse Algorithm

A common approach for solving RCESPPs is by using labeling algorithms in order to investigate the network in a breadth-first manner. Here, we will describe another, recently developed algorithm introduced by Lozano and Medaglia (2013), called the pulse algorithm. This is an exact algorithm, based on a depth-first investigation of the network. The algorithm is therefore suitable for column generation schemes, as we generally search for a feasible solution with negative reduced costs, rather than the one with the lowest reduced costs per se.

The main idea behind the pulse algorithm is to send out pulses from some start node s to a final node f in a clever way. All pulses recursively generate a path P, with reduced costs $rc(P)$ and cumulative demand $q(P)$, by sequentially adding nodes i. If nothing prevents the pulses from propagating through the network, eventually all possible paths will be enumerated. The core strength of the algorithm lies in the pruning strategies, which aggressively and effectively cut off infeasible or uninteresting paths. Therefore, this algorithm has some striking similarities with branch-and-bound, which uses a smart enumeration of all solutions until the optimal one is found. The number of nodes in a branch-and-bound tree is $O\left(b^d\right)$, where b is the branching factor, i.e., the maximum number of children generated at a parent node, and d is the (search) depth of the tree (Morrison et al., 2016). Since the search depth is either fixed or bounded from above, we should investigate ways to reduce b at each node. This is mainly done by effectively pruning uninteresting branches. In particular, we use four pruning strategies: pruning by (1) *feasibility*, (2) *bounds*, (3) *dominance* and (4) *rollback*, which we will describe in Sections 4.2.1 to 4.2.4. These pruning strategies are based on some general ideas which should be adjusted to the problem on hand. Furthermore, we have adapted the algorithm such that it correctly handles the fixed or bounded path size of p-step variables. The pseudo-code of our pulse algorithm is shown in Algorithm 4.1.

Input : $i, P, q(P), rc(P)$
Output : `void`

```
if isFeasible(i, |P|, q(P)) then
    if ¬ isDominated(i, |P|, q(P), rc(P)) then
        if ¬ Rollback(i, P) then
            P' ← P ∪ {i}
            q(P') ← q(P) + q_i
            for j ∈ Γ⁺(i) do
                if j = f then
                    if s = 0 or |P| = p then
                        P'' ← P' ∪ {f}
                        rc(P'') ← rc(P) + c'_if
                        finalPulse(P'', q(P'), rc(P''))
                    end
                else if j ∉ P and |P| < p then
                    rc(P') ← rc(P) + c'_ij
                    if ¬violatesBound(|P'|, rc(P')) then
                        pulse(j, P', q(P'), rc(P'))
                    end
            end
        end
    end
end
```

Algorithm 4.1: Psuedo code of our pulse algorithm.

Here, $|P|$ denotes the size of the path in terms of the number of nodes within it and $\Gamma^+(i) = \{j \in \mathcal{N} : (i,j) \in \mathcal{E}\}$ is the set of head nodes of the outgoing edges from node i. If the algorithm reaches the final node f and the path has a feasible length, the `finalPulse` method constructs and stores a p-step variable if it has negative reduced cost. In Section 5.1.2 we will investigate the influence of a limit on the number of stored variables per pricing problem. For $(0, f)$ pricing problems, paths with 1 up until and including p arcs are allowed to call the `finalPulse` method, while for all other pricing problems we have to propagate the paths until they have exactly p arcs. Note that since every feasible path for any (s, f) pricing problem will always end on node f, we can initialize a pulse with an initial load of q_f, hence the lack of requirement to update $q(P)$ when f is visited. Furthermore, we start with an empty path $P = \emptyset$ with $rc(P) = C_{sf}$ and the initial node $i = s$.

When a partial path is pruned, a whole subspace of the feasible region is pruned, instead of a single solution, which decreases the search time dramatically. Note that the pulse algorithm cannot be applied for a bidirectional investigation of the network, due to its recursive nature.

4.2.1 Feasibility Pruning

The pulse algorithm we use has four pruning strategies in order to quickly prune infeasible and unpromising paths. These strategies are adapted such that they exploit the fixed path length as efficiently as possible.

The first pruning strategy prunes infeasible paths P of which the cumulative load exceeds the reduced vehicle capacity Q_{sf} when we add the input node i. Since we initialize a pulse with starting load q_f, this pruning strategy might be able to quickly prune infeasible paths. Mathematically expressed, this strategy prunes paths for which

$$q_i + \sum_{j \in P \cup f} q_j > Q_{sf}. \tag{4.7}$$

Next, for $s \neq 0$ we can extend this pruning strategy by taking the fixed path length into account. When a partial path P with $|P|$ nodes is created, we know that we have to visit $p - |P| - 1$ customers after visiting s and adding the new node i in order to reach f. We cannot determine beforehand which customers these will be however, but we know that at least the cumulative demand of a set $\mathcal{S}$ with $p - |P| - 1$ customers with least demand, will be added to $q(P)$. Additionally, when $f \in \mathcal{C}$, we know that in order to create a feasible route, we will need to extend the currently formed path with at least $p - 1$ more customers with minimum demand to reach the depot. We thus prune paths for which

$$q_i + \sum_{j \in P \cup \{s,f\}} q_j > \begin{cases} Q\left(\{s,f\}, 2p - |P| - 2\right), & \text{if } s \in \mathcal{C}, f \in \mathcal{C} \\ Q\left(\{s\}, p - |P| - 1\right), & \text{if } s \in \mathcal{C}, f = n+1 \end{cases} \tag{4.8}$$

Note that if we would take the customers currently on path P into account, this would require an $O(n)$ procedure every time the feasibility is checked, which is not desired. Since

the demands are fixed per instance, we can compute both the terms on the right hand side of (4.8) once beforehand for each relevant (s, f) pricing problem, independent of any path. Furthermore, if $s = 0$, we should always use (4.7), since we do not fix the path lengths for such pricing problems.

4.2.2 Bounds Pruning

The second strategy is used when $p > 1$ and uses the fact that we only need to consider p-step variables with negative reduced costs to add to the RMP. Note that due to the fact that (4.3) includes two terms that depend on the node at the head of an arc, it implies that $c'_{ij} \neq c'_{ji}$, i.e., the reduced costs per arc are not symmetric. We therefore have to decouple the edges $\mathcal{E}$ into a set of arcs $\mathcal{A}$, which we use in this pruning strategy.

Assume that $s \neq 0$ and all complete paths will contain exactly p arcs. Given a partial path P with $a(P)$ arcs, we know that the path will have to use $p - a(P)$ more arcs to form a complete path. We do not know which arcs will eventually be used in a complete path, but at least one of these arcs will have reduced cost equal to or bigger than the arc from a customer i to f with the lowest reduced cost.

Furthermore, when $p > 2$, we know that we will need to add an additional $p - a(P) - 1$ arcs from customer to customer in order to construct a complete path. We can also gather these arcs in a set $\mathcal{A}'$ once per pricing problem. Note that these arcs cannot start nor end at f or the depot. Furthermore, since we evaluate the bounds pruning strategy after at least an arc from s to some customer is added to the path, we should also exclude arcs starting or ending at s from $\mathcal{A}'$.

After we gathered these characteristics at the start of a pricing problem, we can prune partial paths P for which

$$rc(P) + \min_{i \in \mathcal{C}} c'_{if} + \min \left\{ \sum_{(i,j) \in \mathcal{A}'} c'_{ij} : i, j \notin \{s, f, 0, n+1\}, \mathcal{A}' \subseteq \mathcal{A}, |\mathcal{A}'| = p - a(P) - 1 \right\} \geq 0 \tag{4.9}$$

If $s = 0$, we should evaluate all possible path extensions from 1 up until $p - a(P) - 1$ arcs in terms of reduced cost and we can subsequently only prune a partial path when the cumulative reduced cost is greater than or equal to 0 in all of these cases.

Let us consider the following example to illustrate the difference of this pruning strategy dependent on the value of s. Say we have an instance, choose $p = 5$ and all paths for this value of p are feasible in terms of demand. For an arbitrary $(0, f)$ pricing problem, we have determined that the arc from a customer to f with lowest reduced cost $\underline{c'_{if}} = 9$, and $c'_{ij}(\mathcal{A}') = \{100, 200, 300, 400\}$. When we encounter a path P with 2 arcs and $rc(P) = -10$, this path will not be pruned, since we could still create a feasible variable with negative reduced cost containing 3 arcs. On the other hand, consider another pricing problem for which $s \neq 0$ and coincidentally $\underline{c'_{if}}$ and $c'_{ij}(\mathcal{A}')$ have the same values. When we encounter a path P', also

containing 2 arcs and $rc(P') = -300$ for such a pricing problem, it will be pruned, because a feasible path with $p = 5$ arcs will have to be extended with arcs that have a cumulative reduced cost of at least 309. This pruning strategy is therefore significantly stronger when the path sized is fixed rather than bounded from above.

Note that our bounding strategy shares some similarities with the path completion acceleration strategy described in Bolívar et al. (2014). The main difference is, however, that we do not prune by looking at the minimum cost completion of a path, but rather use a weaker but easier obtainable set of lowest cost arcs that are not necessarily used in a complete path.

4.2.3 Dominance Pruning

As third pruning strategy, we can discard paths that are dominated by another path in terms of cost and cumulative demand. Feillet et al. (2004) describe that path P dominates path P' if $rc(P) \leq rc(P')$, $q(P) \leq q(P')$ and *each* extension of P that forms a feasible, complete path from s to f is also feasible for P'. The latter thus requires P and P' to not only have the same most recently added node and the same customers are visited on both paths, but they should additionally have the same number of arcs. We can therefore only compare partial permutations of paths.

To use this pruning strategy, we should introduce a label for every possible path size and node. If we encounter a path P for which $rc(P)$ and $q(P)$ are both lower than the best path currently stored at the label for the same path length and most recently added node, we should additionally compare the nodes in both paths, which is $O(|P|)$. We call this dominance strategy *exact* dominance.

Alternatively, we could ignore the comparison of the nodes in both paths, which might result in the pruning of an optimal path, such that this strategy becomes a heuristic. We refer to this pruning strategy as *relaxed* dominance. This might allow us to quickly reach an LP bound which is close to the optimal solution, after which we potentially only have to repeat a few iterations until we have reached optimality. However, if this strategy prunes too many optimal paths, it might require a lot of time to reconstruct these paths again, such that more time will be required to find the optimal LP solution.

This presents us with three strategies we can apply in order to solve the pricing problems. The first is to only use the exact dominance strategy, after which no extra iterations are needed once a solution is found, since this solution will be optimal. Another option would be to use the relaxed dominance strategy until no more negative reduced cost variables are found. After this, we could either choose to neglect dominance altogether and rely on the other pruning strategies to find the optimal solution, or switch to the exact dominance strategy for the last (few) iteration(s). In order to determine which of these three possibilities has the best performance, we compare them in Section 5.1.1.

4.2.4 Rollback Pruning

Lastly, when $p = n + 1$, we use the *rollback* pruning strategy as used in Lozano et al. (2015). Due to the depth-first search nature of the pulse algorithm, poor choices in early stages can take a while to be backtracked such that more promising areas of the search space will be investigated. This pruning strategy therefore re-evaluates a given sequence of nodes by using a dominance strategy. In particular, given a partial path P, of which the two most recently visited customers are i and j, and the next customers this path encounters is k, this pruning strategy checks whether less reduced cost are required to traverse directly from i to k. It thus simply checks whether

$$c'_{ik} \leq c'_{ij} + c'_{jk}.$$

We do not have to compare the demands of both paths, since the demand of the shorter path is less by definition. Note that this comparison is only done for $p = n + 1$, since this pruning strategy would otherwise disregard the fixed or bounded number of arcs of p-steps. In this case the only upper bound on the amount of arcs is an intrinsic property of the problem instance based on demand, rather than imposed by a property of p-step variables.

4.2.5 Search Direction

Due to the similarities with branch-and-bound, we can also focus on another adjustable aspect we can investigate in order to increase performance: the search direction. If we would iterate over $\Gamma^+(i)$ regularly from 1 to n, we would not necessarily encounter the most promising nodes in the tree first during our depth-first search. We should therefore sort the outgoing arcs for each possible starting node in ascending order in terms of reduced cost, such that we extend our path with the most promising node first. This is analogous to a best-first search.

Since (4.3) contains a term that depends on d_r, which in turn depends on s and f, sorting the arcs should be repeated at the start of each (s, f) pricing problem. Unlike our bounds pruning strategy, the search strategy allows us to nudge the exploration of the nodes in the right direction, instead of aggressively pruning parts of the search space. We therefore resort to sorting the reduced cost per arc once after each time the RMP is solved by excluding the term $\alpha_1 q_j$ from (4.3). This allows us to strongly reduce the required sorting, which is $O\left(n^2 \log(n)\right)$, and use the same set of sorted arcs for each pricing problem hereafter. On the downside, the search direction will be less accurate. We therefore choose to use a middle ground by sorting once after each time the RMP is solved, while replacing α_1 with the lowest value of ν over the customers, $\underline{\nu}$, since $\nu_0 = \nu_{n+1} = 0$ by definition. If $\underline{\nu} > 0$, the sorting will be slightly more accurate for each pricing problem except when $s = 0$ and $f = n + 1$. Mathematically expressed, given a path P of which node i is most recently added, the next node j to be explored will thus be

$$j = \arg\min_{k} \{c_{ik} - \pi_{ik} - 2\lambda_k + \underline{\nu} q_k : k \notin P\}. \tag{4.10}$$

Note that for $p \geq \bar{A}$, we only need to solve the $(0, n+1)$ pricing problem, for which $\nu_s = \nu_f = 0$, so we can set $\underline{\nu} = 0$ in this case.

5 Computational Results

To analyze the performance of our algorithm, we use the symmetric CVRP instances of set E by Christofides and Eilon (1969). The names of these instances have the format E-nN-kK, where N is the number of nodes (the customers and one depot) and K is the number of available vehicles for the instance. The number of customers in this instances ranges from 12 to 100, with varying fleet sizes. When an instance is read, a value of p is chosen, after which some characteristics of the (s, f) pricing problems can be determined beforehand. In particular, we determine whether a particular (s, f) is worth investigating and if so, we remove any infeasible nodes from consideration as described in Appendix A. Additionally, the values of Q_{sf} are computed and stored.

Next, the RMP is initialized with a single dummy variable r_d with starting load $d_{r_d} = 0$, path $P_{r_d} = \{0, 1, \ldots, n+1\}$ and cost $c_{r_d} = \sum_{i=0}^{n} 2c_{ii+1}$, which we assume to be sufficiently high such that the dummy variable will never be used in an optimal solution. For the pricing problem, we use a threshold of -10^{-4} for variables to be considered to have negative reduced costs. We also use this threshold on the right-hand side of (4.9). Furthermore, since each (s, f) pricing problem is an independent subproblem, we solve them in parallel.

Finally, the pulse algorithm is initialized per (s, f) pricing problem by starting with an empty path with s as first node to be visited, initial reduced costs C_{sf} and initial load q_f.

As setup we use a laptop with a 2.8 GHz processor, 8 GB of RAM running Windows 10 and IBM ILOG CPLEX 12.8.0 as LP solver. Additionally, we implemented the algorithm with Java using Eclipse version 4.6.3 and a heap size of 512 MB. For the parallelization, this setup has 4 cores with 8 available logical processors, and we use a fixed thread pool with 12 threads.

5.1 Algorithm Setup

Before we are able to generate consistent results of our methodology, we need to consider some choices we have to make. First, we evaluate which type of dominance pruning strategy has the best performance. Additionally, we discuss which limit on the number of variables each pricing problem should maximally yield benefits the overall performance.

5.1.1 Dominance Pruning

We start by evaluating whether it is worthwhile to either use the exact or relaxed dominance pruning strategy. To conduct these experiments, we also use feasibility and bounds pruning strategies. For each (s, f) pricing problem, we stop the propagation of the pulse algorithm when the first negative reduced cost variable is found.

The first option is to use the exact dominance strategy until the optimal LP solution is found, which introduces an $O(|P|)$ procedure for each partial path that calls the method. A second option is to use the relaxed dominance pruning strategy until no more negative reduced

cost variables are yielded, after which we disable it all together and fully rely on the feasibility and bounds pruning strategies for the last few iterations until the optimal LP solution is found. Lastly, we can use a combination of both by using the latter approach, but while using the exact dominance pruning strategy during the last few iterations. These three options are compared in Table 5.1, by using three instances with medium to large numbers of customers and varying values of p.

Table 5.1: Computation times in seconds for the three choices of applying the dominance pruning strategy. The best computation time per row is emphasized in bold.

	Exact dominance	Relaxed dominance, then exact	Relaxed dominance, then none
E-n33-k4			
$p = 6$	10.48	6.47	**5.91**
$p = 7$	49.11	**18.12**	18.37
$p = 8$	641.29	**175.23**	178.49
E-n76-k7			
$p = 4$	13.79	13.86	**13.63**
$p = 5$	**31.48**	33.10	33.55
$p = 6$	118.93	**44.37**	44.61
E-n101-k14			
$p = 1$	3.53	**3.41**	4.72
$p = 2$	10.14	**10.13**	11.94
$p = 3$	**56.69**	57.01	61.07

These results show that when we always use the $O(|P|)$ procedure required for the exact labeling procedure, this mostly costs more than it yields, as it has the worst overall performance. Especially for E-n33-k4, $p = 8$ and E-n76-k7, $p = 6$ the exact dominance procedure has significantly worse performance. However, when we first use relaxed dominance to get a heuristic solution, it seems slightly faster to use exact dominance hereafter in order to reach the optimal LP bound, although the differences are small compared to no dominance at all during the latter stage.

Further investigation yields that with the use of the relaxed dominance strategy, we are able to compute a lower bound within more than 99.5% of the optimal LP solution, such that the exact procedure will only be required to yield a few negative reduced cost variables at most. During these last iterations, we expect that especially the bounds pruning strategy will be very effective, since there are not many arcs left with low negative reduced cost, such that the $O(|P|)$ procedure will not be used redundantly very often. We therefore choose to use the relaxed dominance strategy, after which we use the exact dominance pruning strategy when the final heuristic solution is found in order to find the optimal LP bound.

5.1.2 Number of Variables per Pricing Problem

Thus far, we have chosen to solve each (s, f) pricing problem until the first negative reduced cost variable was found, after which we stop the propagation of the pulses. However, there is a trade-off between the number of partial paths that are generated in a pricing problem and the decrease in objective value of the RMP it yields. We can investigate whether the overall computation times can be reduced when we allow for more negative reduced cost variables to be found per pricing problem. To conduct our experiments, we use feasibility and bounds pruning, the dominance pruning strategy described in the previous section, and only change the number of negative reduced cost variables to be maximally sought during a pricing problem. The result is shown in Figure 5.1.

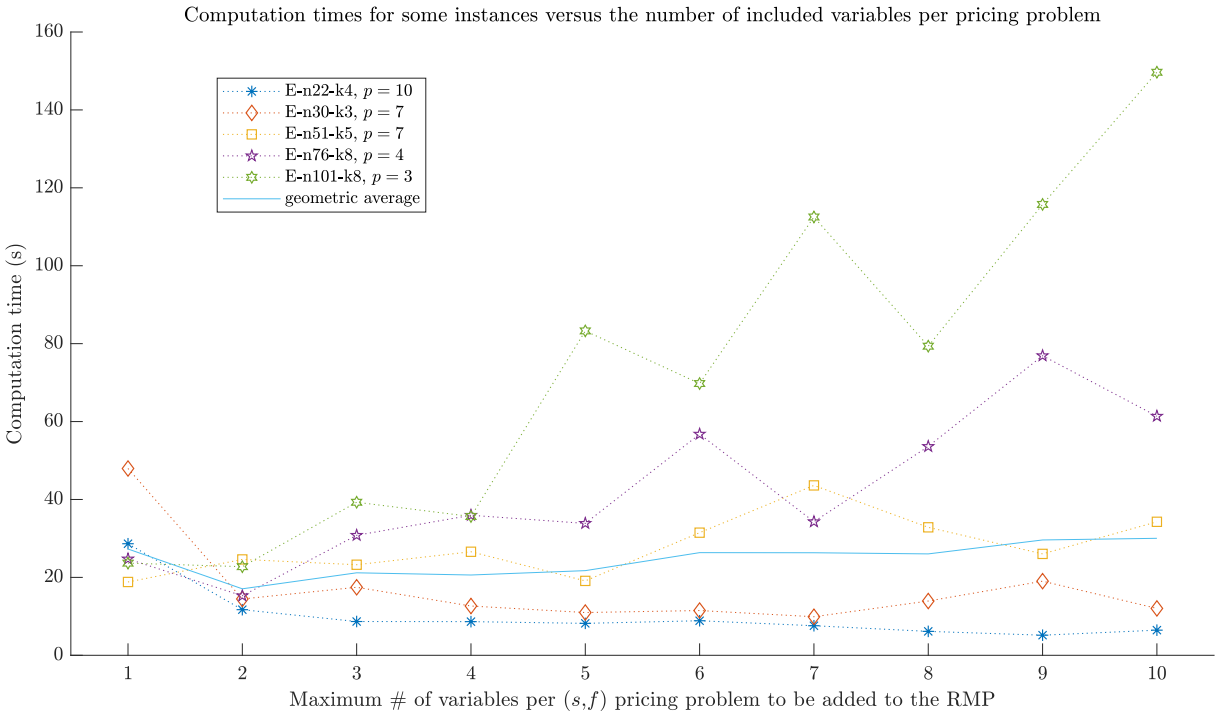


Figure 5.1: The effect on the computation times for numerous limits on the number of variables per (s, f) pricing problem for some instances and values of p. There is no clear overall pattern to be observed, as the effect seems to be highly instance dependent.

This shows some interesting behavior. For E-n22-k4 with $p = 10$, the computation time seems to decrease as we allow more variables to be found during a single pricing problem. However, for E-n51-k5 with $p = 7$ we can observe the opposite behavior, where it seems to be optimal to solve each pricing problem until the first negative reduced cost variable is found. In general, the computation times do not seem to either decrease or increase monotonically. These results indicate that there is no clear, consistent pattern to be observed when everything is kept constant but this limit on the number of variables per pricing problem. However, we

need to fix this parameter in order to continue with generating our results. Since the geometric average is minimized when we allow at most 2 variables to be found per pricing problem, we use this value in the remainder of this section.

5.2 Performance of the Pulse Algorithm

At this point we can continue with constructing consistent computation times for the LP bounds of the benchmark instances. Tables 5.2 and 5.3 respectively show these LP bounds and the corresponding computation times in seconds for all instances of set E for $1 \leq p \leq 8$ and $p = n + 1$, with an imposed time limit of one hour. Parts of these results, indicated by the dashed boundary in the tables, can be compared with those obtained by Munari et al. (2018), which use a labeling algorithm similar to that of Feillet et al. (2004). These are displayed in Tables C.1 and C.2 in Appendix C.

Table 5.2: LP bounds for all instances of the E set for $1 \leq p \leq 8$ and $p = n + 1$. The dashed lines indicate which bounds we can compare to Munari et al. (2018). The bounds emphasized in bold are tighter than the bounds found with the labeling algorithm with *ng*-path relaxation. A single dash indicates that the bound could not be computed within one hour.

p	1	2	3	4	5	6	7	8	$n+1$
E-n13-k4	239.3	239.7	243.2	247.0	247.0	247.0	247.0	247.0	247.0
E-n22-k4	350.0	350.5	354.5	359.7	364.8	369.8	372.4	373.9	373.9
E-n23-k3	529.9	531.8	**535.9**	538.8	**543.1**	**540.4**	544.6	545.0	**565.3**
E-n30-k3	448.7	449.8	451.3	454.8	457.7	458.6	459.0	460.5	-
E-n31-k7	363.3	363.6	368.3	**370.2**	**368.2**	**373.3**	377.5	378.3	**378.3**
E-n33-k4	784.4	785.8	793.5	798.2	**802.3**	**809.1**	811.8	813.6	-
E-n51-k5	499.4	499.6	500.5	504.1	503.9	**506.2**	507.7	509.8	-
E-n76-k7	644.3	644.8	646.8	649.0	651.3	653.9	655.1	656.3	-
E-n76-k8	690.5	691.9	695.6	698.9	703.0	706.4	708.5	711.3	-
E-n76-k10	771.8	773.7	778.7	786.3	792.3	799.3	802.5	804.4	-
E-n76-k14	933.8	938.4	947.3	965.7	981.0	991.2	998.8	-	-
E-n101-k8	768.9	769.5	771.2	773.3	773.9	775.1	776.3	779.0	-
E-n101-k14	994.3	997.5	1005.9	1015.0	1023.8	1029.3	1036.2	-	-

Table 5.3: Computation times in seconds for all instances of the E set for $1 \leq p \leq 8$ and $p = n + 1$. The dashed lines indicate which computation times we can compare to Munari et al. (2018). The times emphasized in bold are computation times which are faster than the labeling algorithm. A single dash indicates that the bound could not be computed within one hour.

p	1	2	3	4	5	6	7	8	$n+1$
E-n13-k4	0.05	0.04	0.03	0.03	0.03	0.04	0.03	0.03	0.03
E-n22-k4	0.09	0.16	0.30	**0.40**	**0.32**	**0.07**	0.40	1.43	0.45
E-n23-k3	0.10	0.09	0.36	0.81	**0.93**	**1.69**	4.67	7.18	167.36
E-n30-k3	0.14	0.56	0.85	**1.24**	**1.52**	**1.84**	3.91	13.80	-
E-n31-k7	0.13	0.66	**0.94**	**1.44**	**1.88**	**3.21**	10.50	80.55	46.35
E-n33-k4	0.15	0.66	**1.03**	**1.61**	**2.47**	**4.34**	11.79	118.48	-
E-n51-k5	0.44	1.25	**2.06**	**3.41**	**5.03**	**6.98**	24.74	39.49	-
E-n76-k7	1.35	4.38	9.04	20.80	39.26	81.41	212.90	559.14	-
E-n76-k8	1.26	4.16	7.69	15.56	58.11	140.24	238.81	1005.65	-
E-n76-k10	1.11	2.92	8.42	21.44	35.06	69.83	229.03	2749.08	-
E-n76-k14	1.15	3.77	16.44	27.63	43.81	130.02	905.48	-	-
E-n101-k8	3.76	11.60	24.91	62.30	155.93	567.94	957.78	2228.51	-
E-n101-k14	3.70	11.16	33.00	88.54	203.75	490.67	1182.49	-	-

Here we can see that for every instance, the LP bound for a certain p is correctly always lower than or equal to the LP bound of the same instance for an integer multiple of p. Due to the fact that the labeling algorithm uses ng-path relaxations to decrease the computation times, this also causes the bounds to weaken, such that we have been able to compute tighter bounds for some instances and values of p. Especially for $p = 5$ and $p = 6$, the pulse algorithm (strongly) outperforms the labeling algorithm. The biggest improvement is made for E-n51-k5 with $p = 6$, where a slightly tighter bound is found with a computation time which is 41 times faster. For $p = 1$ and $p = 2$, we can observe that the labeling algorithm is slightly faster, although the computation times are in the same order of magnitude. Note that the performance of the labeling algorithm for $p = 7$, $p = 8$ and instances with 75 and 100 customers are not known, such that we cannot compare the computation times.

The greatest downside of the pulse algorithm can, however, also be clearly observed for the case when $p \to n + 1$. The corresponding LP bound is always the tightest for any instance and this can be found within reasonable computational times with the labeling algorithm. For the pulse algorithm, the computation times for $p = n + 1$ become intractable. This effect can clearly be observed for E-n23-k3, where the labeling algorithm computes the LP bound within less than 12 seconds, while the pulse algorithm requires about 3 minutes. Similar behavior can be observed for E-n31-k7, where the pulse algorithm is almost 35 times slower. However, for both instances, the LP bound is significantly stronger due to the lack of cycles in the solution. For all other instances, the optimal LP bound for $p = n + 1$ cannot be found within one hour.

Note, however, that algorithms exist which are are also significantly faster than the labeling algorithm of Munari et al. (2018) for solving the classical set partitioning formulation, i.e., when $p = n + 1$, like Martinelli et al. (2014). For this case, a more specialized algorithm may thus be worthwhile.

Another observation we make is that due to the fact that the labeling algorithm uses *ng*-path relaxation, the corresponding LP bound for $p = n + 1$ can even be lower than the LP bound for $p < n + 1$ with the pulse algorithm. This occurs for E-n31-k7, where the labeling algorithm finds an upper LP bound of 377.8 for $p = n + 1$, but the pulse algorithm finds an even tighter bound of 378.3 for $p = 8$, although requiring greatly larger computation time.

We cannot conclude which of the two algorithms performs best in general. The main difference between the algorithm is that the labeling algorithm uses a breadth first investigation of the network, which could be one of the causes of the differences in computation times. It might be interesting to investigate in detail which algorithm performs best for each value of p and why. For now, it seems best to let the value of p decide which algorithm to use for solving the pricing problems to obtain the best overall results.

5.2.1 Pruning Strategies

The core strength of the pulse algorithm lies within the effectiveness of the pruning strategies. It might therefore be interesting to analyze the frequency of the use of each strategy. A visualization of the relative frequency of the number of partial paths which are pruned by each strategy for various instances and values of p is shown in Figure 5.2.

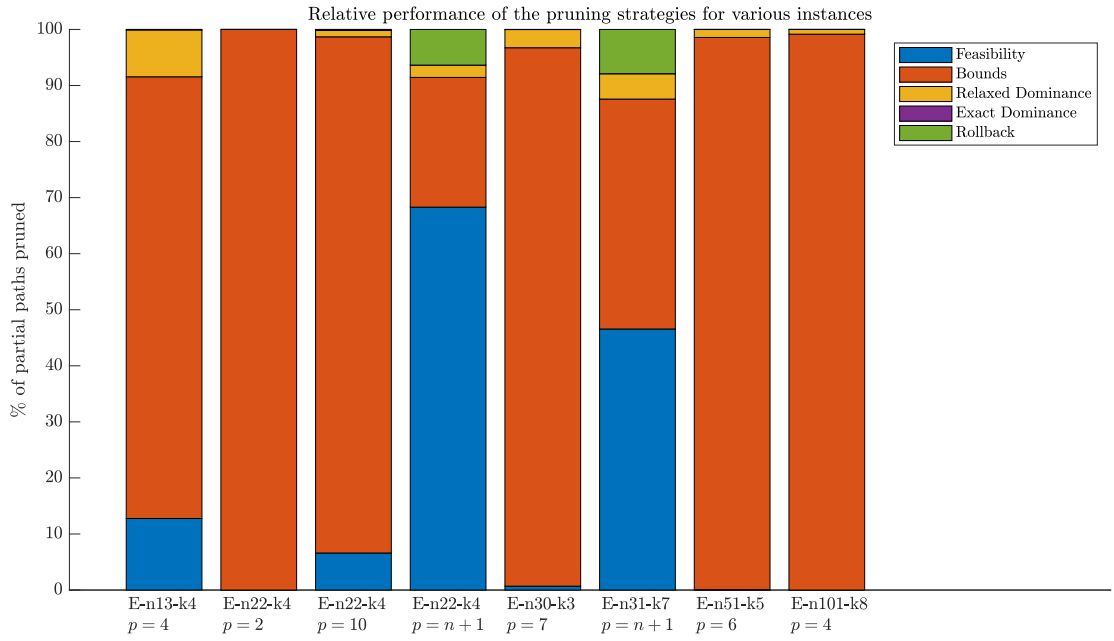


Figure 5.2: Relative frequency of the pruning strategies for a selection of instances and numerous values of p. The bounds pruning strategy seems to have the best overall performance.

From this figure, we can deduce that the bound pruning strategy has the the best overall performance, with it being responsible for up to 100% of the total pruning for E-n22-k4, $p = 2$, while it remains very strong for all other values of p. This indicates that even though the bounds need to be calculated at the start of each (s, f) pricing problem, it yields a very effective pruning strategy. The infeasibility pruning strategy performs better as the value of p increases, which makes sense since more infeasible paths will occur. Especially for $p = n + 1$, the feasibility pruning strategy can be stronger than pruning by bounds. The relaxed dominance strategy is not used very often, and the exact dominance strategy cannot be seen in this stacked bar graph due to the fact that it is only used during the final (few) iteration(s), where its frequency gets overshadowed by the other pruning strategies. Finally, the rollback pruning strategy seems to indeed have a significant positive effect when $p = n + 1$.

Note that we cannot directly conclude which pruning strategy actually reduces the computation time the most, as it is not clear what size of the search space is discarded by pruning a partial path. We can therefore extend our analysis by singling out two instances and values of p and determine how many partial paths are pruned per occurring path size. These are displayed for E-n22-k4 with $p = n + 1$ and E-n51-k5 with $p = 6$ in Tables 5.4 and 5.5 respectively. In both cases, it holds that none of the pruning strategies are used when $|P|$ is equal to 0 or p. Note that for E-n22-k4, $p = n + 1$, we actually use $p = \bar{a}$, which is equal to 11.

$\|P\|$	Feasibility	Bounds	Relaxed Dominance	Exact Dominance	Rollback
1	0	0	0	0	0
2	0	0	353	0	**3460**
3	341	0	5386	11	**12661**
4	29162	1	**55647**	49	33865
5	**288626**	1078	262657	320	84514
6	**1097759**	54124	506126	1065	127271
7	**1615762**	385795	350849	1220	83137
8	**760330**	630622	51719	259	10764
9	49682	**219407**	60	0	16
10	11	**1463**	0	0	0

Table 5.4: Number of partial paths pruned by each strategy for each possible partial path size, denoted by the number of nodes within it, i.e., $|P|$ for E-n22-k4 and $p = n + 1$. The best performing pruning strategy per row is emphasized in bold. For $|P|$ equal to 0 or p ($\bar{a}$), none of the pruning strategies are used.

$\|P\|$	Feasibility	Bounds	Relaxed Dominance	Exact Dominance
1	0	**336692**	0	0
2	8109	**22341825**	1911672	6721
3	127597	**139174854**	4429423	22997
4	278708	**318543427**	3926327	32605
5	107526	**310244855**	1270570	18957

Table 5.5: Number of partial paths pruned by each strategy for each possible partial path size, denoted by the number of nodes within it, i.e., $|P|$ for E-n51-k5 and $p = 6$. The best performing pruning strategy per row is emphasized in bold. For $|P|$ equal to 0 or p, none of the pruning strategies are used. Furthermore, since $p < n + 1$, we do not use the rollback pruning strategy.

In Table 5.4 we can see that the feasibility pruning strategy is indeed the most effective one, which can also be seen in Figure 5.2. The effectiveness increases as the size of the partial path becomes larger, but decreases when the paths almost have full length. This effect can also be seen for the bounds and dominance pruning strategies. Although the rollback strategy is not used very often relatively, it turns out to have the best performance for $|P|$ equal to 2 and 3. For such small path sizes, the volume of the discarded search space will be greatly larger than for bigger partial paths. The rollback strategy could therefore even be the cause for the largest reduction in computation time. For this instance, it is therefore hard to deem which pruning strategy has the best overall performance on its own.

On the other hand, in Table 5.5, the results are more unambiguously. The bounds pruning strategy does not only seem to be responsible for the largest number of pruned partial paths in total, but also for each path size for which the pruning strategies are used. For this instance and value of p, we can thus conclude that the bounds pruning strategy is indeed the best performing one. We expect that similar results will be obtained for instances that visually have the same relative performance in Figure 5.2, like E-n30-k3 with $p = 7$ and E-n101-k8 with $p = 4$.

5.2.2 Parallelization of Pricing Problems

In the start of this section, we mentioned that we use 12 threads to solve the (s, f) pricing problems in parallel. It is interesting to determine whether this also causes a reduction of the computation time of (about) a factor 12. The effect of the number of threads on the computation time for a few instances and values of p with similar running times are shown in Figure 5.3.

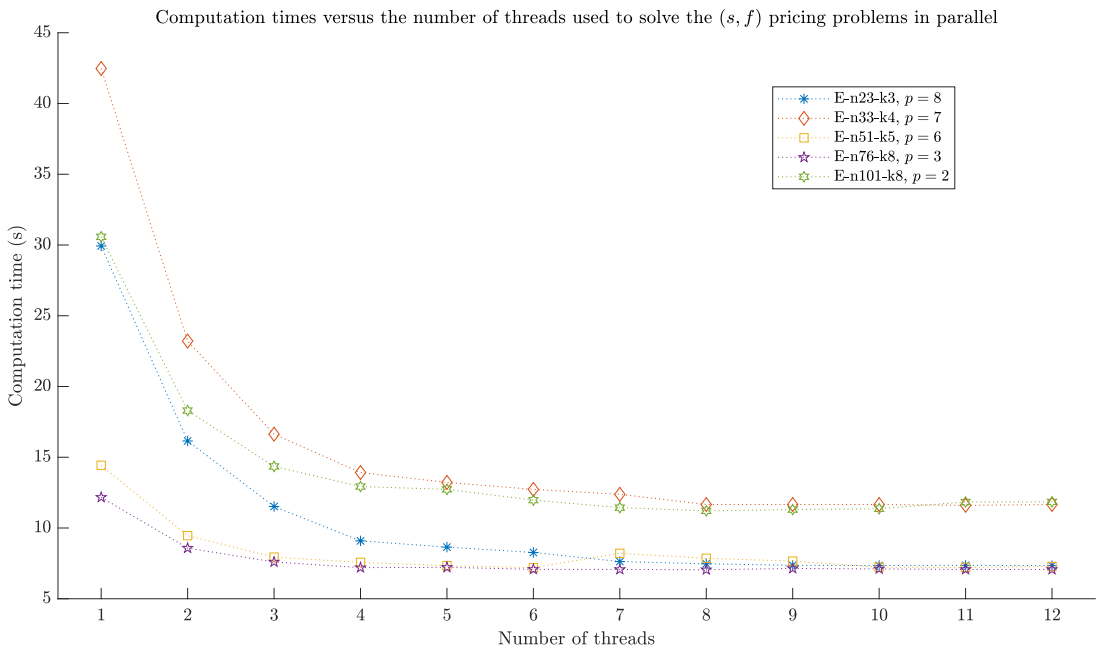


Figure 5.3: Computation times versus the number of threads used to solve the (s, f) pricing problems in parallel. We can observe a tail-off effect in the computation times, which points to the fact that either there are one or more pricing problems which cause a bottleneck in the running time or by the limited number of available cores or memory that our setup has.

Here we can observe that the parallelization indeed has a benefit in the performance of solving the pricing problems. However, when more than 4 threads are used, this effect quickly diminishes, which is caused by pricing problems that all other threads need to wait on until they are solved. If we use more than 8 threads, no significant computational benefit is gained. Further investigation yields that this is not necessarily the $(0, n+1)$ pricing problem, but in general pricing problems that either start or end at the depot relatively take the most time to solve. This is probably caused by the fact that the pruning strategies are relatively the weakest overall for these pricing problems. Note that when we use $p = n+1$, only the $(0, n+1)$ pricing problem will need to be solved, which negates the effect of solving the pricing problems in parallel.

6 Conclusions

This thesis covers the application of the pulse algorithm as method to solve the pricing problem arising from column generation for the p-step formulation. This algorithm has been adapted such that it uses the characteristics of this generalized vehicle routing formulation and its corresponding reduced costs structure.

We use four pruning strategies in order to effectively prune uninteresting or infeasible paths, of which two are adapted to exploit the fixed path length of p-step variables. The bounds pruning strategy seems to have the best performance. We use one of these pruning strategies partially as heuristic which effectively solves the pricing problems, mostly up to more than 99.5% of the optimal solution. The optimal LP solution is mostly found relatively quickly hereafter.

The obtained results are compared to a labeling algorithm. The pulse algorithm has significantly better performance, especially for $p = 5$ and $p = 6$, where tighter bounds can be obtained in more than 40 times smaller computation times. On the other hand, the labeling algorithm is able to compute LP bounds for $p = n + 1$ in the order of seconds, while the pulse algorithm tends to have intractable running times for these instances. However, there exist highly specialized algorithms that solve problem instances also much faster than the labeling algorithm in this case. This illustrates that not a single algorithm seems fit for all values of p. A combination of both the labeling and pulse algorithms could yield the best overall performance, where the value of p can be used to decide which one is most useful.

6.1 Future Research

This thesis has aimed to lay a solid foundation for the use of the pulse algorithm to solve the pricing problems arising from the p-step formulation. From this point onward, there are still a number of interesting topics that can be investigated to enhance the performance even further, aside from the improvement or addition of pruning strategies used in the pulse algorithm.

First and foremost, additional research can be done in significantly improving the performance of the pulse algorithm for $p = n + 1$. In this case, the upper bound on the number of arcs in the path is an intrinsic property of the instance, rather than it being restricted by p. This might make certain assumptions or choices invalid or sub-optimal. Another core difference of the labeling algorithm compared to the pulse algorithm is that it uses a breadth-first investigation of the network instead of depth-first. This also might be one of the reasons that the case when $p = n + 1$ cannot be solved efficiently by the pulse algorithm. In Bolívar et al. (2014), the concept of a pulse queue is introduced, which allows to adjust the search nature of the algorithm. In principle, a fully breadth-first investigation of the network with the pulse algorithm can be achieved with a pulse queue. It might therefore be interesting to investigate whether this could yield improvements. A pulse algorithm especially designed to solve the pricing problems for $p = n + 1$ could be constructed, such that we might find similar computation

times as the labeling algorithm and with potentially tighter bounds.

Another feature that might be open to optimization is the number of partial paths constructed per pricing problem. This seems to be rather unpredictable and heavily instance dependent, while the computation times can be affected significantly by the imposed value. In follow up research, a warm-up phase could be used for a small value of p like 1 or 2, which tends to have low running times, in order to determine which number of variables seem to be best to yield per pricing problem. This value can then be fixed and applied for all higher values of p. It is not guaranteed that this value will be optimal for all values of p, but it might be able to reduce the computation times for some large instances or values of p.

Furthermore, another topic which can be interesting to investigate, arises from the fact that we initially use a heuristic procedure to compute the LP bounds. If we combine this with the fact that the LP bound of an instance with mp, for an integer $m \geq 2$, should always be at least as tight as the bound of the same instance with p, we might be able to stop after the heuristic procedure. Namely, if the LP bound of p is known and mp yields an equal LP bound after the heuristic procedure, we know that the optimal path cannot have been pruned, such that the exact procedure does not have to be ran. This can also be said if the LP bound for $p = n + 1$ after the heuristic procedure is equal to the optimal LP bound of $p < n + 1$. This effect can be observed for E-n22-k4, where $p = 8$ yields the same bound as $p = n + 1$ after the heuristic procedure, such that the negation of the dominance and rollback procedure is not needed and we can thus conclude earlier that we have reached the optimal LP bound.

In our algorithm, we solve each pricing problem in parallel, which reduces the overall computation times. In Lozano et al. (2015), a parallel version of the pulse algorithm itself is also proposed, which is also able to reduce the computation times. We have not yet been able to implement this parallel version of the algorithm, which we additionally propose as topic for future research.

Finally, our pulse algorithm could be embedded in a sophisticated branch-and-price-and-cut scheme, where valid inequalities can be used to boost the performance even further.

Any of these improvements could contribute in solving the pricing problems as fast as possible, which in turn leads to an optimal utilization of the elegant p-step formulation.

Acknowledgements

I would express my gratitude to R. Spliet for his supervision and guidance during my thesis. He always took the time to read all my work in detail and give excellent feedback in order for my research to have a steady and high pace. Also, I would also to thank T. Dollevoet for his help with the comparison between their labeling algorithm and the pulse algorithm, and his overall feedback.

Furthermore, I would like to express my thanks to L. Lozano, A. L. Medaglia and D. Duque for providing me with their Java implementation of the pulse algorithm. This greatly helped me with the development and implementation of my adaptation of the algorithm. I hope my thesis is able to contribute to the overall research and use of the pulse algorithm.

Finally, I would like to thank my family and close friends for their continuous support during the final months of my studies.

Appendices

A Redundant p-step Variables

This section serves to list all redundant p-step variables, which are not needed in order to solve CVRP instances. For proofs on why these can be considered redundant, we refer the reader to Munari et al. (2018).

First, since there is no restriction on the value of d_r, there will be an infinite amount of p-steps to consider. However, there will never be any p-step variables that have a starting load lower than 0 or higher than $Q_{sf} - \sum_{i \in P_r} q_i$, where

$$Q_{sf} = \begin{cases} Q, & \text{if } s \in \mathcal{C}, f = 0 \\ Q(\{f\}, p-1), & \text{if } s = 0, f \in \mathcal{C} \\ Q(\{s, f\}, p-1), & \text{if } s \in \mathcal{C}, f \in \mathcal{C} \end{cases}$$

with $Q(\mathcal{Y}, y) = Q - \min\{\sum_{i \in \mathcal{S}} q_i : \mathcal{S} \subseteq \mathcal{C} \setminus \mathcal{Y}, |\mathcal{S}| = y\}$. All other p-steps are convex combinations of p-steps with the same path and one of either starting loads, such that they are not extreme points of the convex hull. It is thus sufficient to only include p-steps r that

1. start at 0 with starting load $d_r = 0$,
2. do not start at 0 nor end at $n+1$ with either a starting load of $d_r = 0$ or $d_r = Q_{sf} - \sum_{i \in P_r} q_i$,
3. do not start at 0 and end at $n+1$ with starting load $d_r = Q_{sf} - \sum_{i \in P_r} q_i$.

Aside from the starting load, we can also make some statements on which p-step variables are redundant for an optimal integer solution due to the cumulative demand of the customers on the path. It is sufficient to only include p-steps r that

1. start at 0 or $j \in \mathcal{C}$ and end at $n+1$ for which both

   $$\sum_{i \in P_r} q_i \leq Q$$

   and

   $$q_j \leq Q(\{j\}, p-1)$$

2. start at 0 and end at $k \in \mathcal{C}$ for which

   $$\sum_{i \in P_r} q_i \leq Q(\{k\}, p-1)$$

3. start at $j \in \mathcal{C}$ and end at $k \in \mathcal{C}$ ($k \neq j$) for which both

   $$\sum_{i \in P_r} q_i \leq Q(\{j, k\}, p-1)$$

   and

   $$q_j + q_k \leq Q(\{j, k\}, 2p-2)$$

Thirdly, we can remove variables based on an upper bound on the number of arcs for a problem instance $\bar{A}$, which is defined as

$$\bar{A} = \max \left\{ 1 + |\mathcal{S}| : \sum_{i \in \mathcal{S}} q_i \leq Q \right\}$$

Using this quantity, we can remove all p-step variable that

1. start at 0, end at $k \in \mathcal{C}$ and traverse more than $\bar{A} - p$ arcs,
2. start at $j \in \mathcal{C}$ and end at $k \in \mathcal{C}$ $(j \neq k)$ for $p \geq \frac{\bar{A}}{2}$

Finally, for paths that do not both start and finish at the depot, we can make some statements about nodes that will never be able to be visited for feasible paths. In particular, let $i, j, k \in \mathcal{C}$ be three different nodes, such that we can remove all p-steps that

1. start at 0, visit j and end at k for which

$$q_j + q_k > Q(\{j, k\}, p - 1)$$

2. start at i, visit j and end at k for which

$$q_i + q_j + q_k > Q(\{i, j, k\}, 2p - 3)$$

3. start at i, visit j and end at $n + 1$ for which

$$q_i + q_j > Q(\{i, j\}, p - 2)$$

We can thus remove nodes which cause the violation of any of these statements from the graph $\mathcal{G}$ for a given pricing problem.

B The LP Bounds of the p-step Formulation

In this section we describe some theory on the LP bounds of the p-step formulation for the sake of completeness. For proofs, we refer the reader to Munari et al. (2018). Let z_p be the LP bound of the p-step formulation for a particular value of p. The propositions and corollaries are the following:

1. For any $p \in \{1, \ldots, n\}$ and $m \geq 2$, such that $pm \leq n + 1$, it holds that $z_p \leq z_{pm}$.
2. For any $1 < p \leq n + 1$, it holds that $z_1 \leq z_p$.
3. For any $1 \leq p < n + 1$, it holds that $z_p \leq z_{n+1}$.
4. For any $p, q \in \{1, \ldots, n + 1\}$, such that $p < q$ and q is not an integer multiple of p, there exists a CVRP instance for which $z_p > z_q$.

5. For any $p, q \in \{1, \ldots, n+1\}$, such that $p < q$, there exists a CVRP instance for which $z_p = z_q$.

6. For any $p, q \in \{1, \ldots, n+1\}$, such that $p < q$, there exists a CVRP instance for which $z_p < z_q$.

The main conclusion from this theory is that the LP bound increases from its minimum z_1 to its maximum z_{n+1}, although not necessarily monotonically. For the same problem instance with p and an integer multiple, mp with $m \geq 2$, the LP bound must be at least as tight for mp as the one found by using p. Otherwise, the LP bound z_p can either be lower, higher or equal to z_{p-1}.

C Performance of the Labeling Algorithm

Tables C.1 and C.2 display the LP bounds and corresponding computation times in seconds of the labeling algorithm used by Munari et al. (2018). They conducted their experiments on instances of set E for 12 to 50 customers, $1 \leq p \leq 6$ and additionally $p = n+1$. Although these results were obtained using a different setup, benchmark tests showed that there is no significant difference in computational power between setups, such that we do not have to apply a correction factor for the computation times.

Table C.1: LP bounds obtained using the labeling algorithm for instances of the E set ranging from 12 to 50 customers, for $1 \leq p \leq 6$ and $p = n+1$. The bounds for $p = 5$, $p = 6$ and $p = n+1$ are sometimes weaker than those found with the pulse algorithm due to the application of ng-path relaxation (with a neighborhood of size 4).

Instance \ p	1	2	3	4	5	6	$n+1$
E-n13-k4	239.3	239.7	243.2	247.0	247.0	247.0	247.0
E-n22-k4	345.0	350.5	354.5	359.7	364.8	369.8	373.9
E-n23-k3	529.9	531.8	535.4	538.8	543.0	539.6	560.0
E-n30-k3	448.7	449.8	451.3	454.8	457.7	458.6	474.6
E-n31-k7	363.3	363.6	368.3	370.0	368.0	372.8	377.8
E-n33-k4	784.4	785.8	793.5	798.2	802.0	805.4	818.4
E-n51-k5	499.4	499.6	500.5	504.1	503.9	506.1	517.1

Table C.2: Computation times obtained by the labeling algorithm in seconds for instances of the E set ranging from 12 to 50 customers, for $1 \leq p \leq 6$ and $p = n + 1$.

Instance \ p	1	2	3	4	5	6	$n+1$
E-n13-k4	0.01	0.01	0.00	0.03	0.00	0.02	0.00
E-n22-k4	0.02	0.05	0.09	0.80	0.61	0.66	0.17
E-n23-k3	0.03	0.06	0.23	0.64	3.63	6.54	11.75
E-n30-k3	0.03	0.15	0.33	4.06	9.70	13.93	8.84
E-n31-k7	0.05	0.17	1.11	4.21	9.34	12.86	1.33
E-n33-k4	0.09	0.24	1.17	4.28	16.87	36.84	17.59
E-n51-k5	0.37	1.07	3.43	22.79	79.32	286.66	15.90

References

Baldacci, R., Christofides, N., and Mingozzi, A. (2008). An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385.

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.

Bolívar, M. A., Lozano, L., and Medaglia, A. L. (2014). Acceleration strategies for the weight constrained shortest path problem with replenishment. *Optimization Letters*, 8(8):2155–2172.

Bramel, J. and Simchi-Levi, D. (2002). Set-covering-based algorithms for the capacitated vrp. In *The vehicle routing problem*, pages 85–108. SIAM.

Christofides, N. and Eilon, S. (1969). An algorithm for the vehicle-dispatching problem. *Journal of the Operational Research Society*, 20(3):309–318.

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

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.

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

Duque, D., Lozano, L., and Medaglia, A. L. (2015a). An exact method for the biobjective shortest path problem for large-scale road networks. *European Journal of Operational Research*, 242(3):788–797.

Duque, D., Lozano, L., and Medaglia, A. L. (2015b). Solving the orienteering problem with time windows via the pulse framework. *Computers & Operations Research*, 54:168–176.

Eurostat, E. C. et al. (2016). Energy, transport and environment indicators - 2016 edition. *Publications Office of the European Union, Luxembourg*.

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*, 44(3):216–229.

Flood, M. M. (1956). The traveling-salesman problem. *Operations Research*, 4(1):61–75.

Fukasawa, R., Longo, H., Lysgaard, J., de Aragão, M. P., Reis, M., Uchoa, E., and Werneck, R. F. (2006). Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106(3):491–511.

Gilmore, P. C. and Gomory, R. E. (1961). A linear programming approach to the cutting-stock problem. *Operations Research*, 9(6):849–859.

Harper, C., Hendrickson, C. T., and Samaras, C. (2018). Investigating the effects of reserved lanes for commercial truck platooning on congestion: Pennsylvania turnpike case study. Technical report, Instituto Mexicano del Transporte.

Laporte, G., Mercure, H., and Nobert, Y. (1986). An exact algorithm for the asymmetrical capacitated vehicle routing problem. *Networks*, 16(1):33–46.

Laporte, G., Nobert, Y., and Desrochers, M. (1985). Optimal routing under capacity and distance restrictions. *Operations Research*, 33(5):1050–1073.

Leggieri, V. and Haouari, M. (2017). Lifted polynomial size formulations for the homogeneous and heterogeneous vehicle routing problems. *European Journal of Operational Research*, 263(3):755–767.

Little, J. D., Murty, K. G., Sweeney, D. W., and Karel, C. (1963). An algorithm for the traveling salesman problem. *Operations Research*, 11(6):972–989.

Lozano, L., Duque, D., and Medaglia, A. L. (2015). An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science*, 50(1):348–357.

Lozano, L. and Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384.

Lysgaard, J., Letchford, A. N., and Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2):423–445.

Martinelli, R., Pecin, D., and Poggi, M. (2014). Efficient elementary and restricted non-elementary route pricing. *European Journal of Operational Research*, 239(1):102–111.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329.

Morrison, D. R., Jacobson, S. H., Sauppe, J. J., and Sewell, E. C. (2016). Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102.

Munari, P., Dollevoet, T., and Spliet, R. (2016). A generalized formulation for vehicle routing problems. *arXiv preprint arXiv:1606.01935*.

Munari, P., Dollevoet, T., and Spliet, R. (2018). Notes on a p-step formulation for the CVRP.

Pessoa, A., De Aragão, M. P., and Uchoa, E. (2008). Robust branch-cut-and-price algorithms for vehicle routing problems. In *The vehicle routing problem: Latest advances and new challenges*, pages 297–325. Springer.

Ralphs, T. K., Kopman, L., Pulleyblank, W. R., and Trotter, L. E. (2003). On the capacitated vehicle routing problem. *Mathematical Programming*, 94(2-3):343–359.

Toth, P. and Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1-3):487–512.

Vigo, D. and Toth, P. (2015). *Vehicle routing: problems, methods and applications*. Society for Industrial and Applied Mathematics.

Yaman, H. (2006). Formulations and valid inequalities for the heterogeneous vehicle routing problem. *Mathematical Programming*, 106(2):365–390.