NYILATKOZAT

Név: Götz Ádám
Elte, Természettudományi Kar, Matematika BSc.
Neptun azonosító: FJ49CM
Szakdolgozat címe: Label Setting Algorithms for Solving the Column Generation Subproblem of the Vehicle Routing Problem

A szakdolgozat szerzőjeként fegyelmi felelősségem tudatában kijelentem, hogy a dolgozatom a saját, önálló szellemi alkotásom, abban a hivatkozások és idézések a standard szabályok szerint, következetesen lettek alkalmazva. Mások által írt részeket a megfelelő idézés nélkül nem használtam fel.

Budapest, 2023.12.31.

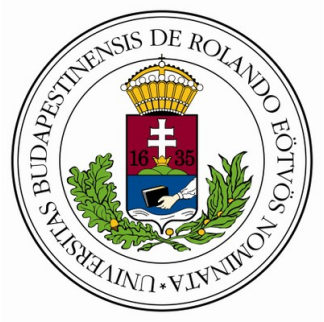
Götz Ádám

Eötvös Loránd University - Faculty of Science

Label Setting Algorithms for Solving the Column Generation Subproblem of the Vehicle Routing Problem

Götz Ádám
BSc in mathematics

Supervisor:
Szabó Eszter
PhD student
Department of Operations Research



Budapest, 2023

Acknowledgements

I would like to express my gratitude to my family, friends, and teachers for their emotional and material support. I am especially thankful to fiance, Kata, who stood by me and believed in me during the most difficult moments, and to Eszter, who patiently offered explanations for topics that occurred throughout the writing my thesis.

Contents

1 Introduction **1**

2 Preliminaries **4**
2.1 Graph theory 4
2.2 Mathematical programming 5
2.3 Travelling Salesman Problem 6
2.4 Branch and Bound 7
2.5 Column generation 8
2.6 Branch and Price 10

3 Capacitated Vehicle Routing Problem with Time Windows **11**
3.1 Naive formulation 13
3.2 Set partitioning model 15
3.3 Set covering model 15
3.4 Column generation for CVRPTW 16

4 Elementary Shortest Path Problem with Resource Constraints **18**
4.1 General label setting algorithm 19
4.2 Bidirectional labeling algorithm 21
4.3 State space augmenting algorithms 26

5 Heuristics **29**
5.1 A greedy approach 30
5.2 Clarke and Wright's Savings algorithm 30

6 Summary **33**

7 Bibliography **35**

List of frequently used abbreviations in my thesis

The following mathematical symbols and abbreviations are going to be used in my thesis.

$\in$ - an element of a set

$\notin$ - not an element of a set

$\mathbb{R}$ - set of real numbers

$\mathbb{Z}$ - set of integers

$\mathbb{R}^n$ - a n dimensional vector over $\mathbb{R}$

$\mathbb{R}^{m \times n}$ - a matrix of size $m \times n$ with m rows and n columns over $\mathbb{R}$

IP - Integer programming

LP - Linear programming

MILP - Mixed integer linear programming

VRP - Vehicle routing problem

CVRP - Capacitated vehicle routing problem

VRPTW - Vehicle routing problem with time windows

CVRPTW - Capacitated vehicle routing problem with time windows

SPPRC - Shortest path problem with resource constraints

ESPPRC - Elementary shortest path problem with resource constraints

S-ESPPRC - S-Elementary shortest path problem with resource constraints

CG - Column generation

NP-Complete - Nondeterministic polynomial time - complete

TSP - Traveling salesman problem

Chapter 1

Introduction

Vehicle routing (VRP) [Dantzig and Ramser, 1959] is a combinatorial optimization problem that aims at finding a cost optimal set of routes in a graph given certain constraints. It is a more general version of the well known travelling salesman problem (TSP) with multiple agents. VRP emerges from logistics, where we need to satisfy the demands of customers for products at different locations. Decision versions of VRPs answer the question: Given a graph G, is there a feasible set of routes that serve all customers and have an associated cost less than $K \in \mathbb{R}$. If this decision can be made quickly, one can use this result to find an optimal solution efficiently via binary search or other techniques.

Decision versions of VRPs are NP-Complete, since a witness can be checked in polynomial time and and VRP is generalisation of the travelling salesman problem that is proven to be NP-complete [Karp, 1972]. In conclusion, while the idea of a polynomial time optimization algorithm for VRP is unrealistic, a well-designed approach can effectively address VRP for a restricted number of customers, offering practical applicability in real-life scenarios. To compare the efficiency of different algorithmic approaches, one can use benchmark data sets. One famous and frequently used benchmark for the Capacitated Cehicle Routing Problem with Time Windows (CVRPTW) is the Solomon data set.

Real life applications require multiple additional constraints including factors such as availability of customers and staff, waiting times, vehicle capacities, priorities among deliveries and possible combinations of these. Some of the known variants of VRP are the following. In each problem the goal is to optimize its objective function that usually means the minimizing the sum of total travel costs.

1. Multi-Depot Vehicle Routing Problem (MDVRP) As the name suggests, in MDVRP there are multiple depots and each depot has an associated fleet of vehicles. In real life it could be a franchise with several shops across the country. A solution to this version is presented by [Mirabi et al., 2016].

2. Capacitated Vehicle Routing Problem (CVRP). In CVRP, each vehicle has a limited capacity, that cannot be exceeded by its load. It is useful if the package sizes are comparable to the size of the transporting vehicle. For more detail see the article [Toth and Vigo, 2002].

3. Pickup and Delivery Problem (PDP). PDP is divided into to parts: Vehicles have to collect items from a subset of customers and deliver it to an other subset of customers. Food delivery or e-commerce businesses may use some variant of this problem. An example for a different objective function is given by [Guo and Wang, 2023] where the goal is to maximize the total number of collected goods.

4. Vehicle Routing Problem with Backhauls (VRPB). In VRPB, some vehicles are used to transport goods from a central depot to customers, while others are used to collect goods from customers and bring them back to the depot. An in-depth traversal of the topic given by [Santos et al., 2019].

5. Split Delivery Vehicle Routing Problem (SDVRP) [Archetti and Speranza, 2012]. SDVRP allows a customer to be visited by multiple vehicles, that can split the shipment into smaller tasks. This is applicable when the shipment is much larger than one transport vehicle. An example can be the transportation of construction material to large building sites.

6. Vehicle Routing Problem with Time Windows (VRPTW). In VRPTW each customer has a specific time window during which they can be visited and there is also a time frame in which the business can operate. With the additional capacity constraint CVRPTW is at spotlight of this thesis.

7. Multi Commodity Vehicle Routing Problem (MCVRP). In the article of [Dellaert et al., 2021] it is combined with CVRPTW. In MCVRP there are multiple commodities that may not be compatible with each vehicle nor with other goods. Chemicals and food products may be a good example for that.

8. Stochastic Vehicle Routing Problem (SVRP). In many cases costs, waiting times and task duration at a customer behave like random variables. SVRP offers a method to incorporate randomness into our solution. An example of this approach is given by [Adulyasak and Jaillet, 2016].

The aim of my thesis is to build up the mathematical background for finding an exact solution for a family of vehicle routing problems [Michelini, 2019] with a strong focus on the CVRPTW (Capacitated Vehicle Routing Problem with Time

Windows). The framework for the solution is the following. A set covering model is created, and a linear relaxation of the problem is solved with branch and price. The obtained pricing subproblem is an elementary shortest path problem with resource constraints (ESPPRC). ESPPRC is solved by a state space augmenting dynamic programming algorithm [Lozano et al., 2016] and its outcome is used to generate columns in the branch and price scheme. A representation of the process is visualized in figure 2.1. At the end of my thesis I discuss some heuristics that can provide a good "warm start" for more complex methods.

Chapter 2

Preliminaries

2.1 Graph theory

Definition (Edge weighted digraphs)**.** *Let $G = (V, A)$ be a collection of nodes V and edges A. Each edge connects two distinct vertices. An edge $e_{ij} \in A$ goes from vertex i to j and has a weight c_{ij}.*

Definition (Path)**.** *An $s - t$ path p is an alternating sequence of vertices and edges starting with vertex $s \in V$ and ending in vertex $t \in V$. Each edge e_{ij} in p is preceded by i and followed by j in the sequence. Note that this definition allows the repetition of nodes and edges.*

Definition (Multiplicity of nodes)**.** *Let p be a path in a directed graph G. Define $M_p(i)$ to be the multiplicity of node $i \in V$ on p. $M_p(i)$ denotes the number of times i appears along p.*

Definition (Path elementarity)**.** *Let p be a path in a directed graph G. Path p is elementary if $M_p(v) \leq 1$ for all $v \in V(p)$. Elementary paths are also called routes or simple paths. The set of elementary paths between s and t is denoted with $\mathscr{P}_{st}$. In my thesis $\mathscr{P}$ usually stands for $\mathscr{P}_{st}$, unless indicated otherwise.*

Definition (Path cost)**.** *Let p be a path in G, with edge weights c_{ij} for edges $e_{ij} \in A$. We define the cost of p to be the sum of all edges along p. Let $E(p)$ be set of edges that are on p.*

$$c(p) = \sum_{(i,j) \in E(p)} c_{ij}$$

2.2 Mathematical programming

Mathematical programming is a problem solving technique. In its most general form it consists of a problem space, defined by a set of constraints $\mathcal{H}$. Let X be a set of solutions that respects all constraints $h \in \mathcal{H}$ and an objective function $f(x) : x \to \mathbb{R}$. The goal is to find elements x that minimize/maximize the value of $f(x)$ for elements $x \in X$.

Linear programming

Linear programming is a member of the mathematical programming methods, that is characterised by both of its constraints and objective function being linear.

Definition (Primal problem)**.** *The standard form of a linear programming problem is*

$$Ax = b$$

$$x \geq 0$$

$$max\ cx$$

where A is an $m \times n$ matrix and $x \in \mathbb{R}^n$ column vector and $c \in \mathbb{R}^m$ is a row vector. This is called the primal problem and if a vector $x \in \mathbb{R}^n$ satisfies

$$A_{i*}x = b_i \qquad\qquad \forall i := 1...m$$

and cx is indeed maximal, then x is a primal optimal solution.

Sometimes it is easier to model problems using inequalities or omitting the non-negativity constraint. Note that all of these alternative formulations can be rewritten into the standard form above. In VRP and many other applications, instead of maximizing the objective function, a minimization is needed. A transformation between minimization and maximization is also easy in each direction, (the objective function must be multiplied by -1).

Definition (Dual problem)**.** *Every system of linear (in)equalities has a dual problem. The corresponding dual problem for the standard form in the previous definition is the following.*

$$yA \geq c$$

$$min\ yb$$

Where A, b and c are the same vectors and matrices as in the primal problem and y is a row-vector. y is a dual solution if

$$yA_{*j} \geq c_j \qquad \forall j := 1...n$$

and optimal if it also minimizes the function yb.

To solve the primal and dual problems simultaneously, one can use the simplex algorithm. For LP problems there exists a polynomial algorithm [Karmarkar, 1984], but typically a variant of the simplex method is used, which is exponential in worst-case but polynomial in expected run-time [Borgwardt, 1987].

Integer programming

Integer programming denotes mathematical programming problems that require their solution vector's elements to be integers.

Definition (Integer linear programming, ILP)**.** *Integer linear programming is a special optimization problem, where $x \in \mathbb{Z}^n$ meanwhile being a solution to a set of linear inequality constraints.*

Integer linear programming is NP-Complete since any given solution can be verified in polynomial time and every SAT (Satisfiability Problem) formula can be translated into an integer programming problem also in polynomial time. It is proven by the Cook-Levin theorem that SAT is in NP-C [Cook, 1971], [Levin, 1973] thus IP is NP-Complete as well. For every IP problem there exists a linear programming relaxation that omits integrality constraints. This relaxation does usually not provide a good approximation of the integer optimum, but it can serve as a lower bound for minimization and as an upper bound for maximization problems.

2.3 Travelling Salesman Problem

Travelling Salesman is a widely known optimization problem, that searches for the shortest circle visiting all vertices exactly once in a given weighted, directed graph $D = (V, A)$. Sometimes it is assumed that weights are symmetric, meaning $c_{ij} = c_{ji}$, non-negative and satisfy the triangle inequality (metric TSP). The decision version TSP is NP-complete since deciding the existence of a Hamiltonian cycle in a graph is NP-complete.

There are known approximation algorithms that find solutions close to optimum. One famous algorithm from Christofides provides a 1.5-approximation ratio

for the TSP [Christofides, 1976]. TSP can be formulated as an integer programming problem as follows.

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.3.1}$$

$$x_{ij} \in \{0,1\} \tag{2.3.2}$$

$$\sum_{j\in\delta^+(i)} x_{ji} = 1 \qquad \forall i \in V \tag{2.3.3}$$

$$\sum_{j\in\delta^-(i)} x_{ij} = 1 \qquad \forall i \in V \tag{2.3.4}$$

$$\sum_{\delta^-(S)} \geq 1 \qquad \forall S \subset V, S \neq \emptyset \tag{2.3.5}$$

x_{ij} is a binary variable that is 1 if the edge (i,j) is used and 0 otherwise. The variable c_{ij} denotes the cost of edge (i,j). The set $\delta^+(i)$ is the set of predecessors and $\delta^-(i)$ is the set of successors of vertex i. Condition 2.3.5 states that every real subset of V must have at least 1 outgoing edge that prevents multiple disjoint circles as a solution.

It is easy to see that this formulation requires an exponential number of rows that are difficult to handle both in space and in time. Column (or in this case rather row) generation proves to be a powerful tool that can be used to solve to linear relaxation of the problem. Initially only a polynomial size subset of the constraints are considered. In each step some rows, violating the original constraints, are added to the examined part. This process is further described in paragraph 2.5.

2.4 Branch and Bound

Branch and bound is a technique to solve combinatorial optimization problems. The general scheme consists of the following steps. Branch and bound divides the problem into smaller subproblems, that are solved recursively. One important attribute of the subproblems is that the solution of the original problem is also a solution to one of the smaller problems. In each iteration the algorithm discards subproblems those theoretical optimum is worse than the best solution found so far. If there are no more subproblems to be examined, it returns the best solution so far, that is indeed optimal. Here is an example using BNB to solve an IP problem. Let S_0^* be an optimization problem defined as

$$S_0^* := \{Ax = b; x \geq 0; \min cx, x \in \mathbb{Z}^m\}.$$

Algorithm 2.1 Branch and bound

1: Let S_0 be the the LP-relaxation of S_0^*
2: Let OPT be the smallest integer solution found so far
3: $OPT := EstimatedOptimum(S_0^*)$
4: Let L be a list of subproblems.
5: $L \leftarrow S_0$
6: **while** L is not empty **do**
7: Select $S \in L$
8: Remove S from L
9: $OPT^* := Min(S)$
10: **if** $OPT^* > OPT$ **then**
11: Continue
12: **else**
13: $x^* := ArgMin(S)$
14: **if** x^* is integral **then**
15: $OPT := OPT^*$
16: **else**
17: Let x_i^* be a fractional component.
18: $S_1 = AddConstraint(S, x_i \leq \lfloor x_i^* \rfloor)$
19: $S_2 = AddConstraint(S, x_i \geq \lceil x_i^* \rceil)$
20: $L \leftarrow S_1, S_2$

In line (3) an upper bound can be calculated using heuristics. Frequently used heuristics for VRPTW are described in section 5. $L \leftarrow S$ adds the subproblem S to the list of subproblems L. An LP-solver calculates $Min(S)$ for S returning its minimum. $ArgMin(S)$ returns the vector x for which cx is minimal. $AddConstraint(S, s')$ adds the constraint s' to the problem S.

2.5 Column generation

Some problems are easier to formulate using the linear combination of feasible configurations, such as VRP. Let us consider a matrix in which columns represent feasible paths of one vehicle. A solution to the VRP is the union of several vehicle paths and can be described as a combination of their associated columns. Since the number of columns is exponentially large in this case, the problem needs a different approach. Experiments show that column generation is efficient for solving such type of problems. The following is a general scheme for generating columns that are going to used for finding solutions.

1. Initialize the Restricted Master Problem (RMP) with columns for which there is a primal solution.

2. Calculate the primal and dual solutions of the RMP.

3. Check if the dual solution satisfies the inequalities for all columns. If not, add the column violating the inequalities to the RMP and go back to (2).

4. There must be no more columns violating the constraints. As a conclusion the primal and dual solutions are both feasible, and due to complementary slackness the value of their objective functions must be equal and the optimum have been found.

We can check an exponential number of columns in a reasonable time in the following way. The dual optimal solution to the RMP is denoted with η. Let B be a matrix formed from the basis vectors of the primal optimal solution in the Simplex algorithm. B is non singular, therefore η can be calculated via

$$\eta = c_B \cdot B^{-1}.$$

Let us introduce the following notations. $\theta_r := \eta A_r$ where A_r is the column of A associated with route r and the cost of route r as $c_r := c(r)$. The dual problem states that

$$\theta_r \leq c_r$$

must be fulfilled for all feasible routes r. Reformulated, the reduced cost function is the following.

$$0 \leq c_r - \theta_r$$

One can notice that it is sufficient to calculate

$$\min c_r - \theta_r. \tag{2.5.1}$$

If the minimum value is less than zero then there is a column that does not satisfy the dual inequalities and it is added to the RMP. If the minimum is greater or equal than zero, then all values are greater or equal than zero, which implies the feasibility of the dual solution. As described above, a solution must be optimal if it is both primal and dual feasible and for which complementary slackness is also fulfilled.

Solving the minimization problem 2.5.1 leads to a combinatorial optimization problem that is more manageable than the original problem. In CVRPTW the dual problem is an elementary shortest path problem with resource constraints (ESPPRC) that can be solved using labeling algorithms. Solving ESPPRC is described in detail in sections 4.1 and 4.3.

2.6 Branch and Price

Branch and price is a combination of branch and bound and column generation in the same model. The framework is outlined in figure 2.1. After branching, two new instances are created of the problem with added constraints, described in algorithm 2.1. After examining both branches, the algorithm returns to the parent node. If there are no more parent nodes to return to, the algorithm terminates with the best solution found so far. Colors indicate that the algorithm modifies or leaves currently inspected "iteration".

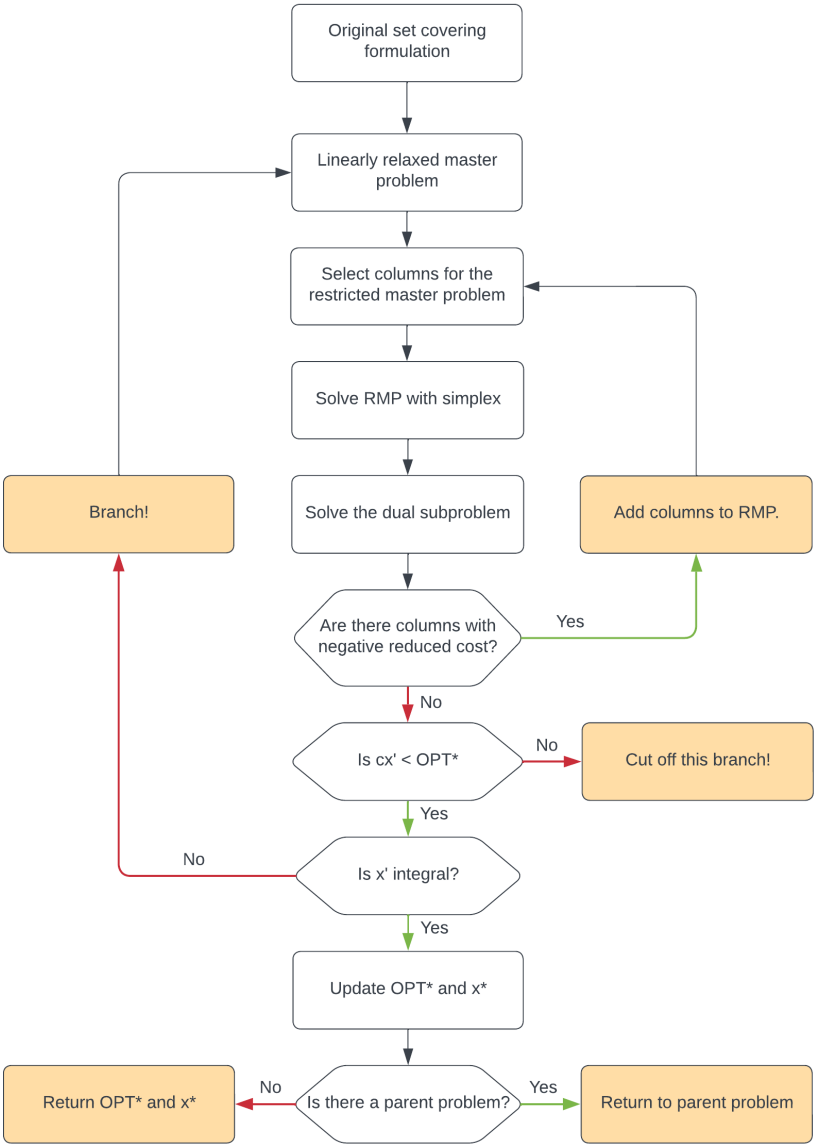


Figure 2.1: Simplified flowchart of the Branch and Price framework

Chapter 3

Capacitated Vehicle Routing Problem with Time Windows

CVRPTW is the combination of the Capacitated Vehicle Routing Problem and the Vehicle Routing Problem with Time Windows. The goal is to minimize travel costs such that the solution respects capacity and time window constraints. Besides travel times, tasks of different lengths at each node are added to the problem. To incorporate this into the model, closing times are adjusted with consideration of task duration. In order to better simulate real-world scenarios, vehicles are granted the flexibility to wait at their next destination prior to commencing their tasks.

In following chapters the depot is split in nodes s and t unless noted otherwise. Node s is the starting node and it has only the outgoing edges of the depot whilst t denotes the end node with incoming edges of the depot.

Definition (Resource feasibility)**.** *Let $\mathcal{R}$ be the set of resources and $\mathscr{W}$ be the resource limit vector $\mathscr{W} = (\mathscr{W}_1, ..., \mathscr{W}_\mathcal{R})$ with $\mathscr{W}_i \in \mathbb{R}^+$. Let $w_{ij}^k \in \mathbb{R}^+$ be the consumption of resource $\mathscr{W}_k$ on an edge $(i,j) \in A$ and let $w_{ij} = (w_{ij}^1, ..., w_{ij}^\mathcal{R})$ be the collection of consumed resources on the e_{ij}. On each path p the accumulated consumption of resource $\mathscr{W}_k$ is computed by*

$$\mathcal{W}^k(p) := \sum_{(i,j)\in A(p)} w_{ij}^k$$

and $W(p)$ is defined as the collection of resources consumed along p

$$\mathcal{W}(p) = \big(W^1(p), ..., W^\mathcal{R}(p)\big)$$

A path p is resource feasible if

$$\mathcal{W}(p) \leq \mathscr{W}$$

meaning that the inequality holds for all $k \in \{1, ..., \mathcal{R}\}$

$$\mathcal{W}^k(p) \leq \mathscr{W}_k$$

Definition (Feasibility of multiple paths)**.** *A set* $S \subseteq \mathscr{P}$ *of paths is feasible, if* p *is feasible* $\forall p \in S$*.*

In the above definition of resource feasibility, consumption of resources happens on edges. Resource consumption on vertices can be transformed easily into the "edge-resource" form by adding the consumption of node $j \in V$ to all edges $(i, j) \in A$.

Definition (Sufficiency of multiple paths)**.** *A set* $S \subseteq \mathscr{P}$ *of paths is called sufficient if every node* $v \in V$ *is an element of at least one path* $p \in S$*. It translates to the real world as demands of all customers being satisfied.*

Definition (Cost of multiple paths)**.** *Let* $c(S)$ *denote the cost of a set of paths as in*

$$c(S) := \sum_{p \in S} c(p)$$

Definition. *Resource Constrained Vehicle Routing Problem searches for a sufficient set of resource constrained elementary paths* $S^* \subseteq \mathscr{P}$ *from* s *to* t *that is minimal, meaning*

$$c(S^*) \leq c(S), \ \forall S \subseteq \mathscr{P} \text{ that are sufficient and feasible}$$

CVRP is an instance of a resource constrained VRP with only one capacity-resource. VRPTW is somewhat more complex due to the inclusion of time constraints for all nodes. For this to work with the previous definitions only the time constraints of the nodes on the current path are taken into account.

3.1 Naive formulation

In this section a detailed mixed integer formulation of the combined CVRPTW problem is presented. The following notations are used in this section.

V: Set of nodes (other than the depot).

A: Set of arcs.

s: Depot, with only the outgoing edges.

t: Depot, with only the incoming edges.

U: Set of vehicles.

$u \in U$: One of the vehicles.

x_{ij}^u: Decision variable, denoting the usage of arc (i, j) by vehicle u.

c_{ij}: Cost of arc $(i, j) \in A$

a_i: Opening time of node i.

b_i: Adjusted closing time$i$.

s_i: Task duration at node i.

T_{ij}: traveling time between (i, j).

t_i^u: Time variable. Starting time of vehicle u at node i.

M: A very large number.

$\delta^+(i) := \{\forall j : (j, i) \in A\}$ - Set of vertices prior to node i.

$\delta^-(i) := \{\forall j : (i, j) \in A\}$ - Set of vertices subsequent to node i.

Q: Capacity of each vehicle

d_i: Size of the demand of customer i.

The CVRPTW model is described by the following (in)equalities and objective function.

$$\min \sum_{u \in U} \sum_{(i,j) \in A} c_{ij} x_{ij}^u \tag{3.1.1}$$

subject to

$$x_{ij}^u \in \{0, 1\} \tag{3.1.2}$$

$$\sum_{u \in U} \sum_{j \in \delta^-(i)} x_{ij}^u \geq 1 \qquad \forall i \in V \qquad (3.1.3)$$

$$\sum_{j \in \delta^-(s)} x_{sj}^u \leq 1 \qquad \forall u \in U \qquad (3.1.4)$$

$$\sum_{i \in \delta^+(j)} x_{ij}^u - \sum_{i \in \delta^-(j)} x_{ji}^u = 0 \qquad \forall u \in U, j \in V \qquad (3.1.5)$$

$$a_i \leq t_i^u \leq b_i \qquad \forall u \in U, i \in V \qquad (3.1.6)$$

$$t_i^u + s_i + T_{ij} - t_j^u \leq M(1 - x_{ij}^u) \qquad u \in U,\ (i,j) \in A,\ j \neq t \qquad (3.1.7)$$

$$t_i^u + s_i + T_{it} - b_t \leq M(1 - x_{it}^u) \qquad u \in U,\ i \in \delta^+(t) \qquad (3.1.8)$$

$$\sum_{i \in V} d_i \sum_{j \in \delta^+(i)} x_{ij}^u \leq Q \qquad \forall u \in U \qquad (3.1.9)$$

Details and explanations for the model are listed below.

- We assume that c_{ij} is uniform for all vehicles, symmetric and satisfies the triangle inequality. Usually c_{ij} is the result of running an All Pairs Shortest Path algorithm on the original data.
- Closing time of node i is shifted by its task duration $b_i := b_i^* - s_i$, where b_i^* denotes the original closing time.
- Uniformity among vehicle capacities is assumed by using a common Q. This could be uniquely defined for each vehicle but it would make the model even more complicated.
- Constraint 3.1.2 prescribes that x_{ij}^u is a binary variable. In a solution x_{ij}^u is 1, if vehicle u uses arc (i, j) and 0 otherwise.
- The objective function 3.1.1 prescribes that the sum of the cost of all used vertices should be minimized.
- Inequality 3.1.3 guarantees that every node must be visited by at least one vehicle.
- Equality 3.1.4 assures that each vehicle leaves the depot at most once.
- Equality 3.1.7 makes sure that x^u is a flow, which means that every vehicle going into node i leaves the node.
- Inequality 3.1.6 prescribes that each task must be started once the node is open and it must be finished before it is closed.
- In inequality 3.1.7 the following is described: If an arc x_{ij}^u is used, the task at vertex j can only be started after arriving at vertex j. This property is

not required for the depot as it would make the solution infeasible. This is due to the fact that it eliminates cycles (if task lengths and travelling times have non-zero values).

- Inequality 3.1.8 is the previous requirement rewritten for the depot. It ensures the feasibility of starting a task at node i and going back to the depot before its closing time.

- Inequality 3.1.9 assures that each vehicle has a load less than its capacity. The model assumes that each vehicle reaching node i is delivering a load as well. This implies that loads cannot be split up.

It turns out that this formulation is not practical when building large scale models but it provides a starting point to better understand the following approaches.

3.2 Set partitioning model

Let Ω denote the set of feasible routes that satisfy constraints described in subsection 3.1. a_{ir} is 1, if route r visits vertex i and 0 otherwise. In set partitioning c_r is total length of route $r \in \Omega$. The ILP form of the the model is the following.

$$\min \sum_{r \in \Omega} c_r y_r \tag{3.2.1}$$

subject to

$$y_r \in \{0, 1\} \tag{3.2.2}$$

$$\sum_{r \in \Omega} a_{ir} y_r = 1 \qquad \forall i \in V \tag{3.2.3}$$

y_r is a decision variable, that is 1, when route r is used and 0 otherwise. Our goal is to find a solution that visits each vertex exactly once. Equations in 3.2.3 establish that every node must be visited by exactly one route.

3.3 Set covering model

Set covering is a seemingly weaker version of a set partitioning model, where 3.2.3 is changed to 3.3.3. It implies that every node must be covered by at least one route instead of a strict equality.

$$\min \sum_{r \in \Omega} c_r y_r \tag{3.3.1}$$

subject to

$$y_r \in \{0, 1\} \tag{3.3.2}$$

$$\sum_{r \in \Omega} a_{ir} y_r \geq 1 \qquad \forall i \in V \tag{3.3.3}$$

This formulation is usually preferred to a set partitioning model, because the dual variables of the relaxed problem are non-negative, that leads to a more stable algorithm.

3.4 Column generation for CVRPTW

Set covering described in 3.3 is an IP problem with an exponential number of columns, since the number of feasible routes in a graph is exponential as well. Branch and Price is generally a good candidate to solve this type of problems. The speed of the algorithm heavily depends on the starting configuration and chosen algorithm to solve the pricing subproblem. Besides, additional inequalities can be added to the master problem to narrow down the search space. In my thesis, my focus will be on improving the pricing algorithm, but alongside I will introduce some heuristics and inequalities alongside to boost performance. As described in section 2.5 finding a column that violates the dual constraint

$$0 \leq c_r - \theta_r$$

leads to an elementary shortest path problem with resource constraints (ESP-PRC). Let $E(r)$ be the set of arcs that belong to a route r. Let $C(r)$ be the reduced cost of r.

$$C(r) = c_r - \theta_r$$

In more detail $C(r)$ can be written out as

$$C(r) = \sum_{(i,j) \in E(r)} c_{ij} - \sum_{(i,j) \in E(r)} \eta_i.$$

This also defines a reduced edge weight $C_{ij} = c_{ij} - \eta_i$. Let $r^* \in \Omega$ be path for which the following holds

$$C(r^*) \leq C(r), \qquad \forall r \in \Omega.$$

If $C(r^*) < 0$, then a column with negative reduced cost have been found that can be added to the subproblem. If $C(r^*) \geq 0$, then $C(r) \geq 0$ must be fulfilled for all routes $r \in \Omega$.

The objective of the ESPPRC is to find a shortest path from the depot to the depot without the repetition of nodes in a graph meanwhile obeying resource limitations. The depot is usually split into two nodes, the source s and the sink node t. ESPPRC is NP-Complete as it can be reduced to the longest path problem (since the reduced edge weight C_{ij} can be negative as well).

Chapter 4

Elementary Shortest Path Problem with Resource Constraints

The Elementary Shortest Path Problem with Resource Constraints arises as we are try to solve the pricing subproblem of the VRP with at least one additional constraint such as time windows or capacities of resources. In the following chapter a more general definition of the ESPPRC is introduced and I present three algorithms with progressively increasing complexity to solve the described problem efficiently.

Definition (ESPPRC)**.** *Consider a directed graph $G = (V, A)$ with no restriction on edge weights $c_{ij} \in \mathbb{R}$. Let $\mathcal{R}$ denote a set of resources and $\mathcal{W} = (\mathcal{W}_1, ..., \mathcal{W}_{\mathcal{R}})$ be a resource limit vector. Resource consumption on edge $(i, j) \in A$ is denoted by $w_{ij} = (w_{ij}^1, ..., w_{ij}^{\mathcal{R}})$, where $w_{ij}^{\mathcal{R}} \in \mathbb{R}^+$. ESPPRC searches for an elementary $s-t$ path $p \in \mathscr{P}_{st}$ that is resource feasible with respect to $\mathcal{W}$ while minimizing $c(p)$, the total cost along p.*

In the setting of column generation $c_{ij} = C_{ij}$, where C_{ij} is the reduced cost defined in the previous section. Resources and resource constraints remain the same naturally occurring time and capacity constraints introduced in chapter 3.

To find an elementary shortest path respecting resource constraints labeling algorithms are proposed. A label setting algorithm is a dynamic programming method that sends "labels" from already reached vertices to their neighbouring vertices in each iteration. Each route has an associated label and each label contains information about their route such as its cost and its accumulated resource consumption. Efficient labeling algorithms must have a set of additional rules to limit the unnecessary generation of labels.

4.1 General label setting algorithm

In labeling algorithms each v_i vertex has an associated set of labels, that represent feasible paths from the source node to vertex v_i. Each label can be described by a tuple like $l_{v_i} = (C_{v_i}, \mathscr{W}_{v_i})$. For the sake of simplicity, vertex v_i is denoted by i and the label belonging to v_i is denoted by $l_i = (C_i, \mathscr{W}_i)$. The symbol $\mathscr{W}_i$ stands for two resources τ_i and q_i. Resource τ_i measures the time elapsed between leaving the source node s until the arrival at node i. and resource q_i is the accumulated load on the same path p. C_i is the reduced cost of path p. Label l_i can be written out as

$$l_i = (C_i, \tau_i, q_i).$$

Let Γ_i denote the set of all labels l_i at node i. In the beginning of the algorithm there is a single label in Γ_s with values of $l_s = (0, 0, 0)$. New labels are dynamically sent from already reached vertices to their neighbours. The updates are carried out until no more possible extensions are available. Rules for extending labels are the following.

$$C_j := C_i + c_{ij} - \eta_j \tag{4.1.1}$$

$$\tau_j := \max\{\tau_i + s_i + T_{ij}, a_j\} \tag{4.1.2}$$

$$q_j := q_i + d_j \tag{4.1.3}$$

Rule 4.1.1 ensures that cost C_j at label l_j remains a reduced cost. Rule 4.1.2 states that the start time at node j cannot be less than the sum of the start time at i, the task length s_i and the travel time T_{ij} or less then the opening time a_j. Rule 4.1.3 prescribes that the accumulated load of a vehicle that visits node j after visiting node i is the sum of its accumulated load until i and the load at j.

Label l_j is added to Γ_j if $\tau_j \leq b_j$ and $q_j \leq Q$ are fulfilled. Domination rules are introduced to reduce the number of labels generated. The labels at each node are the elements of a partially ordered set. Label l_i^1 *dominates* label l_i^2 if and only if

$$C_i^1 \leq C_i^2 \tag{4.1.4}$$

$$\tau_i^1 \leq \tau_i^2 \tag{4.1.5}$$

$$q_i^1 \leq q_i^2 \tag{4.1.6}$$

and at least one of the inequalities are strict. l_i^2 is *dominated* by l_i^1 because all feasible extensions of l_i^2 is worse than the extension of the label l_i^1. Each time a label is extended to node j, the following routine is executed.

- If l_j is dominated by at least one label in Γ_j, the l_j is discarded.

- If l_j is not discarded, l_j is added to Γ_j and all labels dominated by l_j are removed from Γ_j.

The algorithm terminates as there are no more labels to extend, returning the minimal element of Γ_t.

Elementarity constraints

So far, this algorithm provides a solution to the shortest path problem with resource constraints (SPPRC) without the exclusion of non-elementary paths. Early implementations of the algorithm did not try to eliminate all cycles. The standard method was to remove 2-cycles by forbidding the visit of direct predecessors, because it was relatively easy to implement. A new component p_i is added to label $l_i = (C_i, \mathcal{W}_i, p_i)$, that contains the parent node of i on the associated path. A new extension rule is added that a label cannot be sent to its direct predecessor [Desrochers et al., 1992]. To solve the ESPPRC, N elementarity resources are added to each label [Feillet et al., 2004]. This problem is also called resource constrained shortest path problem with node resources.

Definition. *Each label in Γ_i is defined by the following tuple.*

$$l_i := (C_i, \tau_i, q_i, (E_i^k)_k) \qquad (E_i^k)_{k \in N} \tag{4.1.7}$$

The notation $(E_i^k)_{k \in N}$ can be unpacked as

$$l_i := (C_i, \tau_i, q_i, E_i^1, ..., E_i^N). \tag{4.1.8}$$

For each label E_i^k is 1 if vertex k is visited by the associated path and 0 otherwise. Resource extension functions are completed with the following rule:

$$E_j^k := \begin{cases} E_i^k + 1, & \text{, if } k = j \\ E_i^k & \text{, if } k \neq j \end{cases} \tag{4.1.9}$$

A partial path with label l_j is elementary if and only if $E_i^k \leq 1$ for all $k \in N$. Otherwise it cannot be added toΓ_j. A new domination rule for node resources is also added to the program.

$$E_i^{k,1} \leq E_i^{k,2} \qquad \forall k \in N \tag{4.1.10}$$

The complete algorithm for solving ESPPRC [Michelini, 2019] is given in algorithm (4.1).

Algorithm 4.1 Monodirectional dynamic programming algorithm for ESPPRC

```
1: procedure MDP
2:     //Initialization//
3:     Γ_s ← {(0, 0, 0, 0)}
4:     E ← s
5:     for all i ∈ V \ {s} do
6:         Γ_i ← ∅
7:     //Search//
8:     while E ≠ ∅ do
9:         Select i ∈ E
10:        //Extension//
11:        for all l_i ∈ Γ̄_i do
12:            for all j ∈ δ_i^- such that E_i^j = 0 do
13:                l_j ← EXTEND(l_i, j)
14:                if l_j is feasible then
15:                    LDCR(Γ_j, l_j)
16:                if Γ̄_j ≠ ∅ and j ≠ t then
17:                    E := E ∪ {j}
18:            E := E \ {i}
19:    return p*
```

Definition. *As introduced before Γ_i denotes set of labels at node i. In $\overline{\Gamma}_i \subseteq \Gamma_i$ are the labels that are yet to be extended in the algorithm.*

Definition. *Let $\mathscr{P}^*$ denote the set of paths assigned to labels at Γ_t at the algorithm termination and p^* be the path in $\mathscr{P}^*$ that minimizes its associated reduced cost $C(p)$.*

Vertices that are to be evaluated are in set E. These are nodes that already have some assigned labels that are yet to be extended. EXTEND(l_i, j) is a function that calculates the l_j using the label extension rules 4.1.1, 4.1.2, 4.1.3 and 4.1.9. LDCR(Γ_j, l_j) (Label dominance comparison routine) is a routine that compares l_j to labels in Γ_i according to dominance rules 4.1.4, 4.1.5, 4.1.6 and 4.1.10. If l_j is dominated by any label in Γ_j it has to be discarded and any label in Γ_j dominated by l_j has to be removed from Γ_i and $\overline{\Gamma}_i$. If l_j is not dominated by any of the labels, it is added to Γ_j and $\overline{\Gamma}_j$. The algorithm terminates in finite steps since the accumulated resources of not yet extended labels are monotonically increasing and resource constraints are finite as well.

4.2 Bidirectional labeling algorithm

Time complexity of algorithm 4.1 is heavily reliant on the number of labels created during its completion. To improve expected run-time, bidirectional label extension

is proposed by [Righini and Salani, 2006]. The main idea is to apply forward and backward searches starting from the source and the sink simultaneously and combining their results. With some additional bounding rules, this reduces the expected number of generated labels significantly.

To implement bidirectional search, new sets of labels are introduced. Γ_i^f is the set of *forward* labels of node v_i, previously denoted by Γ_i and Γ_i^b is the new set of *backward* labels associated with v_i. Forward labels are extended just as regular labels in the previous algorithm. A backward label

$$l_i^b = (C_i^b, \tau_i^b, q_i^b, (E_i^k)_k^b) \in \Gamma_i^b$$

represents the path from node i to the sink. $(C_i^b, q_i^b, (E_i^k)_k^b)$ are analogous to the forward case, they denote the cost, the accumulated load and the elementarity constraints respectively. The resource τ_i^b stands for the time passed from the end of service at node i until entering the sink. Backward label extensions are performed starting at the sink with the label $l_t^b = (0, 0, 0, 0) \in \Gamma_t^b$. In each iteration, labels in Γ_i^b are forwarded to the predecessors of node i. Time τ is not symmetrical for forward and backward extensions, [Righini and Salani, 2006] propose the idea of *backward time windows*. Regular time windows of node v_i are shifted by the task duration s_i:

$$[a_i^b, b_i^b] := [a_i + s_i, b_i + s_i]$$

Note that $b_i + s_i$ is the actual closing time since b_i was defined as $b_i = b_i^* - s_i$. Let $T := \max_{i \in V}\{b_i + s_i + T_{it}\}$ be the time of the latest possible arrival at node t calculated from the adjusted closing times of each vertex. This is because the latest feasible departure from v_i is $b_i + s_i$, since time window b_i is already adjusted in the CVRPWTW model by task length s_i. Reminder that t_{it} is the traveling time from v_i to the sink. Backward label extensions on arc $(j, i) \in A$ are carried out in the following way.

Resource extension rules for backward labels

Label l_i^b is extended to label l_j^b.

$$\tau_j^b := \max\{\tau_i^b + s_i + T_{ij}, T - b_j^b\} \tag{4.2.1}$$

$$C_j^b := C_i^b + c_{ji} - \eta_j \tag{4.2.2}$$

$$q_j^b := q_i^b + d_j \tag{4.2.3}$$

$$E_j^{k,b} := \begin{cases} E_i^{k,b} + 1 & \text{, if } k = j \\ E_i^{k,b} & \text{, if } k \neq j \end{cases} \tag{4.2.4}$$

Backward label l_j^b is feasible regarding capacity and elementarity constraints if and only if the next inequalities are fulfilled.

$$\tau_j^b \leq T - a_j^b \tag{4.2.5}$$
$$q_j^b \leq Q \tag{4.2.6}$$
$$E_j^{k,b} \leq 1 \qquad \forall k \in N \tag{4.2.7}$$

Feasibility constraint 4.2.5 checks that one leaves node j later than its latest feasible arrival at j. Inequality 4.2.6 verifies that capacity constraints are fulfilled and 4.2.7 is for checking elementarity conditions.

Domination rules for backward labels

$$C_i^{b,1} \leq C_i^{b,2} \tag{4.2.8}$$
$$\tau_i^{b,1} \leq \tau_i^{b,2} \tag{4.2.9}$$
$$q_i^{b,1} \leq q_i^{b,2} \tag{4.2.10}$$
$$E_i^{k,b,1} \leq E_i^{k,b,2} \qquad \forall k \in N \tag{4.2.11}$$

Similar to the forward case, a backward label $l_i^{b,2}$ is *dominated* by label $l_i^{b,1}$ if and only if inequalities 4.2.8, 4.2.9, 4.2.10, 4.2.10 and 4.2.11 are satisfied with at least one of the inequalities being strict.

Label concatenation

Forward and backward labels are joined via label concatenation. For each edge $(i,j) \in A$ concatenated cost and resources of l_i^f and l_j^b are calculated with these functions:

$$\overline{C}(i,j) := C_i^f + C_j^b + c_{ij} \tag{4.2.12}$$
$$\overline{\tau}(i,j) := \tau_i^f + s_i + T_{ij} + s_j + \tau_j^b \tag{4.2.13}$$
$$\overline{q}(i,j) := q_i^f + q_j^b \tag{4.2.14}$$

$\overline{C}(i,j)$ is a function that calculates total cost of a concatenated path. Function $\overline{\tau}(i,j)$ calculates the total time elapsed between the departure from the source until arrival at the sink. $\overline{q}(i,j)$ is the total accumulated load alongside the forward and backward paths. A concatenated path is feasible if the following inequalities are fulfilled.

$$\overline{\tau} \leq T \tag{4.2.15}$$

$$\overline{q} \leq Q \tag{4.2.16}$$

$$E_i^{k,f} + E_j^{k,b} \leq 1 \qquad \forall k \in N \tag{4.2.17}$$

Inequalities 4.2.15, 4.2.16 are a trivial requirement, since total duration and load must not break time window and capacity constraints. Inequality 4.2.17 states that every node k should exclusively be visited by either the forward or the backward path.

Restriction of elapsed time for forward and backward labels

It is apparent that bidirectional label extension leads to a duplication of states that makes the computation of a solution more difficult to handle. In their paper [Righini and Salani, 2006], authors describe two approaches to solve the issue of state duplication: arc bounding and resource bounding.

For each label there is a set of feasible arcs on which label extensions can be performed. Arc bounding computes an upper bound for size of this set by solving a multi-knapsack problem with regards to the resource constraints. Resource bounding selects a *critical* resource that is monotonous along each path. Authors chose τ as it fulfills this requirement. The idea is to bound the length of the partial paths, by introducing further extension rules on its labels. Both forward and backward label extension rules are complemented with conditions

$$\tau_i^f \leq \frac{T}{2}$$
$$\tau_i^b \leq \frac{T}{2}$$

where T is an upper limit for the path duration from s to t. This does not exclude optimal solution, because each $s-t$ path can be concatenated in a way, that i and j are in the middle with respect to the elapsed time.

Duplicate elimination after label concatenation

Each feasible path can be constructed with multiple concatenations. For example, let (i, j, k) be subsequent vertices in a given feasible path. Concatenation of appropriate labels (l_i^f, l_j^b) and (l_j^f, l_k^b) produce the same outputs. To avoid unnecessary duplicates we can use the critical resource from the previous paragraph. From all possible concatenations of an $s-t$ path, the one is used, that minimizes

the distance

$$\Phi_{(i,j)} := |\tau_i^f - \tau_j^b|.$$

To find the edge, for which $\Phi_{(i,j)}$ is indeed minimal, a duplicate eliminating algorithm 4.2 [Michelini, 2019] can be used.

Algorithm 4.2 Duplicate elimination

```
1:  procedure HALFWAY(l_i^f, l_j^b)
2:      Φ_(i,j) := |τ_i^f − τ_j^b|
3:      if τ_i^f < τ_j^b then
4:          τ_j^f := τ^f(l_i^f, j)
5:          Φ_(j,j+1) ← |τ_j^f − τ_{j+1}^b|
6:          if Φ_(i,j) < Φ_(j,j+1) then
7:              return true
8:          else
9:              return false
10:     else
11:         τ_i^b := τ^b(l_j^b, i)
12:         Φ_{i−1,i} ← |τ_{i−1}^f − τ_i^b|
13:         if Φ_(i,j) ≤ Φ_(i−1,i) then
14:             return true
15:         else
16:             return false
```

Algorithm 4.2 returns **true** if (l_i^f, l_j^b) are halfway in the concatenated path and returns **false** otherwise. $\tau^f(l_i^f, j)$ is a function that returns the critical resource consumption of the forward extended label. $\tau^b(l_j^b, i)$ computes the same for the backward extension from l_j^b. For any τ^f and τ^b, algorithm HALFWAY has a constant run-time. Bidirectional search is described in algorithm 4.3 and the concatenation algorithm is given in 4.5.

Algorithm 4.3 Bidirectional dynamic programming algorithm for ESPRC

```
1:  procedure BDDPA
2:      //Initialization//
3:      for all i ∈ V do
4:          Γ_i^f ← ∅, Γ_i^b ← ∅
5:      Π ← ∅, Γ_s^f ← {(0,0,0,0)}, Γ_t^b ← {(0,0,0,0)}, E ← {s,t}
6:      //Search//
7:      while E ≠ ∅ do
8:          Select i ∈ E
9:          EOL(i)
10:         E := E \ {i}
11:     CONCATENATE(G, Γ^f, Γ^b)
12:     return best path in Π
```

Algorithm 4.4 Extending all labels of $\overline{\Gamma}_i^f$ and $\overline{\Gamma}_i^b$

```
1: procedure EOL(i)
2:     //Extensions//
3:     for all l_i^f ∈ Γ̄_i^f do //Forward extension//
4:         for all j ∈ δ_i^- do
5:             if E_i^j = 0 then
6:                 l_j ← EXTEND^f(l_i, j)
7:                 if l_j is feasible then
8:                     Γ_j^f ← LDCR(Γ_j^f, l_j)
9:                 if Γ̄_j^f ≠ ∅ then
10:                    E := E ∪ {j}
11:    for all l_i^b ∈ Γ̄_i^b do //Backward extension//
12:        for all k ∈ δ_i^+ do
13:            if E_i^{k,b} = 0 then
14:                l_k^b ← EXTEND^b(l_i^b, k)
15:                if l_k^b is feasible then
16:                    Γ_k^b ← LDCR(Γ_k^b, l_k^b)
17:                if Γ_k^b ≠ ∅ then
18:                    E := E ∪ {k}
```

Algorithm 4.3 extends labels simultaneously starting from s and t, limited by resource extension rules and the limit of $\tau_i^f \leq T/2$ and $\tau_i^b \leq T/2$. Extensions, similar to algorithm 4.1 are described in 4.4. Function CONCAT uses concatenation rules 4.2.12,4.2.13 and 4.2.12.

Algorithm 4.5 Label concatenation for bidirectional search

```
1: procedure CONCATENATE(G, Γ^f, Γ^b)
2:     for all (i, j) ∈ A do
3:         for all l_i^f ∈ Γ_i^f do
4:             for all l_j^b ∈ Γ_j^f do
5:                 if FEASIBLE(l_i^f, l_j^b) and HALFWAY(l_i^f, l_j^b) then
6:                     P := CONCAT(l_i^f, l_j^b)
7:                     Π := Π ∪ {P}
8:     return best path in Π
```

4.3 State space augmenting algorithms

Introducing N node resources in 4.1.8 adds more complexity to the problem that makes computation of an optimal solution even harder. It meant that initial algorithms eliminated two-cycles but they did not incorporate all node resources. Among others, [Kohl, 1995] and [Boland et al., 2006] proposed a restriction that

elementarity resources should only be demanded for a subset of nodes. Their strategy was to dynamically extend the set of nodes those elementarity is demanded until the point where a solution would be elementary with regards to all nodes.

Definition (S-ESPPRC)**.** *S-ESPPRC is a relaxation of the elementary shortest path problem with resource constraints. Given $S \subseteq V$, the objective of S-ESPPRC is finding a resource constrained shortest path p in the graph G from the starting node s to the end node t such that the cost $C(p)$ along p is minimal and no node $v \in S$ is visited more than once.*

In case of $S = \emptyset$, S-ESPPRC reduces to the SPPRC and if $S = V$ then S-ESPPRC is equivalent to the ESPPRC.

Definition. *Labels of the S-ESPPRC at node i are defined as follows.*

$$l_i^S = (C_i, \tau_i, q_i, (E_i^k)_k), \ k \in S \tag{4.3.1}$$

$$l_i^S = (C_i, \tau_i, q_i, E_i^{S_1}, ..., E_i^{S_{|S|}}) \tag{4.3.2}$$

Where $S_1, ..., S_{|S|}$ define the elements of S in an arbitrary order.

To solve S-ESPPRC a modification of the Mdp or Bddpa described by algorithms 4.1 and 4.3 can be used where labels are restricted as in 4.3.2. The modified versions are going to be referred as S-Mdp and S-Bddpa.

Definition. *Let $\mathscr{P}^*(S)$ be the the set of paths assigned to labels at node t at the end of the S-ESPPRC algorithm for a given set S and let p_S^* be a path in $\mathscr{P}^*(S)$ that minimizes path cost $C(p)$.*

In case there are more than one minimal cost path then an arbitrary choice can be made. The following. For the algorithm to be deterministic the sub-procedure terminates as the first path is found. The general scheme is described in algorithm (4.6).

Algorithm 4.6 General state space augmenting algorithm (GSSAA) for S-ESPPRC

```
1: procedure GSSA
2:     //Initialization//
3:     S := ∅
4:     Use S-MDP(S) to get 𝒫*(S) and s*_P
5:     while s*_P is not elementary do
6:         UPDATE(S, 𝒮*(S))
7:         Use S-MDP(S) to get 𝒫*(S) and s*_P
8:     return s*_P
```

Instead of using S-MDP, GSSA can also use the bidirectional (S-BDDPA) version of the label setting algorithm. There are several strategies to write the UPDATE$(S, \mathscr{P}^*(S))$ function. Note that all strategies include choosing based on information obtained from $\mathscr{P}^*(S)$.

1. **HMO (Highest multiplicity on the optimal path - one node)**

   In this approach p_S^* is chosen from $\mathscr{P}^*(S)$ and node i is selected that maximizes $M_{p_S^*}(i)$. If there are multiple nodes with the same multiplicity, the algorithm chooses the first one along p. The UPDATE$(S, \mathscr{P}^*(S))$ procedure just sets $S := S \cup \{i\}$.

2. **HMO-All (Highest multiplicity on the optimal path - all nodes)**

   This algorithm uses p_S^* just as the algorithm above but now all nodes maximizing $M_{p_S^*}$ are added to the updated set. Formally $S' := \{\forall j \in V : M_{p_S^*}(j) \geq M_{p_S^*}(i),\ \forall i \in V\}$ and UPDATE$(S, \mathscr{P}^*(S))$ sets $S := S \cup S'$.

3. **MO-All (Multiplicity more than one on the optimal path)**

   This algorithm also uses p_S^* but S' contains all nodes that have a multiplicity more than one. Formally $S' := \{\forall i \in V : M_{p_S^*}(i) > 1$ and the UPDATE function is defined as $S := S \cup S'$.

4. **M-All (Multiplicity more than one on one of the paths)**

   This approach uses a random path $p \in \mathscr{P}^*(S)$ instead of p_S^*. S' is defined as $S' := \{\forall j \in V : M_p(j) \geq M_p(i),\ \forall i \in V\}$ and UPDATE is $S := S \cup S'$.

5. **All (no relaxation)**

   This is the previous label setting formulation that leads to the ESPPRC with S initialized as $S := V$. There is no augmenting of S since there is only one iteration of the algorithm.

In each algorithm the size of S is strictly increasing with each iteration. HMO has the slowest growth as there is only one node being added and M-All has the fastest pace. For comparison, the original label setting algorithm was also tested by [Boland et al., 2006]. Numerical experiments showed that HMO performed best in both time and memory usage the non-relaxed algorithm had the worst performance.

Chapter 5

Heuristics

When tackling an optimization problem, a good "starting" solution can significantly improve expected running-time. To find a good feasible solution, heuristics can be applied. Heuristics are usually based on intuition and do not have a strong mathematical background.

As defined in 3 The goal of VRP is to find $S^* \subset \mathscr{P}$ that is sufficient and feasible, for which

$$C(S^*) \leq C(S')$$

for all sufficient and feasible sets $S' \subseteq \mathscr{P}$, where

$$C(S') := \sum_{p \in S'} c_p$$

. Heuristic algorithms search for $\hat{S}$ that are sufficient and feasible and

$$C(\hat{S}) \leq \alpha C(S^*)$$

where $\alpha \geq 1$ is called the approximation-ratio if it holds for all inputs. The set $\hat{S}$ is not required to be optimal, but one needs to be able to calculate it relatively fast. For many algorithms $\alpha(N)$ is a function of N that depends on the input size and it is not a constant like in simpler examples. Heuristics often find local optima, that are not guaranteed to be optimal but are than other configurations in their "neighbourhood".

In this section, two heuristic approaches are given. The first is a greedy algorithm, which is a simplified version of the mono-directional label setting algorithm discussed in 4.1. The second algorithm is the Savings algorithm by Clarke and Wright, a heuristic that gained popularity in the 1960s as one of the first techniques to build up answer to the problem systematically.

5.1 A greedy approach

A greedy algorithm chooses its direction based on properties that are locally optimal. Greedy algorithms provide an optimal solution on matroids but for VRP it is easy to construct counterexamples for which greedy algorithms provide a solutions that are far off from optimality. In algorithm 5.1, a simple example is given for a greedy approach.

Algorithm 5.1 Greedy path construction

```
1: //Initialization//
2: V_visited := {s}
3: V_unvisited := V \ {s, t}
4: C_total = 0
5: v := s
6: //Search//
7: while V_unvisited ≠ ∅ do
8:     S := FEASIBLE(δ⁻(v) ∩ V_unvisited)
9:     if S = ∅ then
10:        if v = s then
11:            return Unfeasible.
12:        else
13:            C_total := C_total + c_vt
14:            v := s
15:    else
16:        w := ARGMIN{c_vj | j ∈ S}
17:        C_total := C_total + c_vw
18:        l_w := EXTEND(l_v, w)
19:        V_visited := V_visited ∪ {w}
20:        V_unvisited := V_unvisited \ {w}
21:        v := w
22: return C_total
```

The function FEASIBLE(A) selects feasible nodes from set A. A node $j \in V$ following $i \in V$ is feasible if

$$t_i + s_i + T_{ij} \leq b_j \tag{5.1.1}$$

$$t_i + s_i + T_{ij} + s_j + T_{jt} \leq b_t \tag{5.1.2}$$

$$q + q_j \leq Q \tag{5.1.3}$$

5.2 Clarke and Wright's Savings algorithm

The following algorithm as a heuristic for the CVRP was first proposed by [Clarke and Wright, 1964]. This is a slightly modified version of it for directed graphs. Nodes are denoted by P_i using the original notation of Clarke and Wright. Let P

be a path containing node P_i and let P_{i-1}, P_{i+1} be nodes preceding and succeeding P_i on path P.

1. Initialize paths $P^i := (P_0, P_i, P_0)$ of length 2 for all nodes. After initialization every point is on exactly one path.

2. In every iteration the paths for each pair (P_i,P_j) can be written as

$$P^i := (P_0, ..., P_{i-1}, P_i, P_{i+1}, ..., P_0)$$
$$P^j := (P_0, ..., P_{j-1}, P_j, P_{j+1}, ..., P_0)$$

   Since these paths directed, let us consider the following concatenations of P_i and P_j.

   (a) In this case the P^i is redirected at P_i into P_j and cut off parts are reconnected with each other.

$$P^1 := (P_0, ..., P_{i-1}, P_i, P_j, P_{j+1}, ..., P_0)$$
$$P^2 := (P_0, ..., P_{j-1}, P_{i+1}, ..., P_0)$$

   (b) This version is the interchanged version of (a).

$$P^1 := (P_0, ..., P_{j-1}, P_j, P_i, P_{i+1}, ..., P_0)$$
$$P^2 := (P_0, ..., P_{i-1}, P_{j+1}, ..., P_0)$$

   (c) The next case is similar to (a) but cut off parts are rerouted into the depot.

$$P^1 := (P_0, ..., P_{i-1}, P_i, P_j, P_{j+1}, ..., P_0)$$
$$P^2 := (P_0, ..., P_{j-1}, P_0)$$
$$P^3 := (P_0, P_{i+1}, ..., P_0)$$

   (d) This is similar to (b) but cut off parts are rerouted similar to (c).

$$P^1 := (P_0, ..., P_{j-1}, P_j, P_i, P_{i+1}, ..., P_0)$$
$$P^2 := (P_0, ..., P_{i-1}, P_0)$$
$$P^3 := (P_0, P_{j+1}, ..., P_0)$$

   If P_i and P_j are on the same path, only (c) and (d) are possible choices.

3. Let s_{ij} denote the maximal savings between P_i and P_j that is the maximum of the following values.

$$s_{ij}^{(a)} := c(P_i, P_{i+1}) + c(P_{j-1}, P_j) - c(P_i, P_j) - c(P_{j-1}, P_{i+1})$$
$$s_{ij}^{(b)} := c(P_j, P_{j+1}) + c(P_{i-1}, P_i) - c(P_j, P_i) - c(P_{i-1}, P_{j+1})$$

$$s_{ij}^{(c)} := c(P_i, P_{i+1}) + c(P_{j-1}, P_j) - c(P_i, P_j) - c(P_{j-1}, P_0) - c(P_0, P_{i+1})$$
$$s_{ij}^{(d)} := c(P_j, P_{j+1}) + c(P_{i-1}, P_i) - c(P_j, P_i) - c(P_{j-1}, P_0) - c(P_0, P_{i+1})$$

4. In each round calculate the s_{ij} savings from (a) to (d) for all pairs (P_i, P_j) and save the maximal feasible savings for each pair. A saving is feasible if the concatenated path is feasible.

5. Reduce total cost by concatenating P_i and P_j for (i, j) that maximizes s_{ij}. If $\max s_{ij} \leq 0$ return the current set of paths.

Chapter 6

Summary

In my thesis I reviewed parts of the theory of solving the Vehicle Routing Problem with Time Windows. The main idea of my thesis was to formulate VRP as a set covering problem that can be solved with branch and price.

As part of branch and price, a new sub-problem of identifying columns that violate dual constraints arose. The dual subproblem was formulated as an Elementary Shortest Path Problem with Resource Constraints, for which I presented three dynamic programming algorithms. Each algorithm uses the idea of sending labels to neighbouring vertices iteratively. Labels contain information about their assigned route, such as accumulated load, route length and route duration. As there are an exponential number of labels generated, a restriction of unnecessary label generation is a must. To restrict the generation of labels and reducing time and space complexity, two more advanced algorithms are introduced. The first algorithm takes a bidirectional approach, and the second algorithm relaxes the elementary condition of the ESPPRC and augments its state space only requiring necessary node resources.

Last but not least I examined two heuristics to solve VRP with less computation power. Heuristics do not provide an optimal solution but sometimes these algorithms are good enough for one's purposes or can be used as a starting point for BNP. There are many heuristics and metaheuristics that provided a closer-to-optimal solution that are not part of my thesis but could be useful in future work.

In the future I would like to build up a general VRP model using Python, since up until now I only implemented part of the heuristics, and label setting algorithms mentioned in the thesis. Python-MIP seems to be a great library to start with as it supports open-source and state-of-the-art commercial solvers such as COIN-OR[1] and Gurobi[2]. From a mathematical perpective I want to explore

[1]COIN-OR Linear Programming Solver - CLP (https://github.com/coin-or/Clp)
[2]GUROBI - (https://www.gurobi.com/)

valid inequalities and more advanced heuristics, that can help creating an algorithm that can efficiently operate with inputs of real world scale. As part three I would like to create graphical user interface (GUI), to make it into a user friendly application for others to experiment with.

Chapter 7

Bibliography

[Adulyasak and Jaillet, 2016] Adulyasak, Y. and Jaillet, P. (2016). Models and algorithms for stochastic and robust vehicle routing with deadlines. *Transportation Science*, 50(2):608–626.

[Archetti and Speranza, 2012] Archetti, C. and Speranza, M. G. (2012). Vehicle routing problems with split deliveries. *International Transactions in Operational Research*, 19(1-2):3–22.

[Boland et al., 2006] Boland, N., Dethridge, J., and Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34:58–68.

[Borgwardt, 1987] Borgwardt, K. H. (1987). *The Polynomiality of the Expected Number of Steps*, pages 142–186. Springer Berlin Heidelberg, Berlin, Heidelberg.

[Christofides, 1976] Christofides, N. (1976). Worst-case analysis of a new heuristic for the travelling salesman problem. *Operations Research Forum*, 3.

[Clarke and Wright, 1964] Clarke, G. and Wright, J. W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581.

[Cook, 1971] Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA. Association for Computing Machinery.

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

[Dellaert et al., 2021] Dellaert, N., Van Woensel, T., Crainic, T. G., and Dashty Saridarq, F. (2021). A multi-commodity two-echelon capacitated vehicle routing problem with time windows: Model formulations and solution approach. *Computers Operations Research*, 127:105154.

[Desrochers et al., 1992] 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.

[Feillet et al., 2004] 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*, 44:216–229.

[Guo and Wang, 2023] Guo, Q. and Wang, N. (2023). The vehicle routing problem with simultaneous pickup and delivery considering the total number of collected goods. *Mathematics*, 11(2).

[Karmarkar, 1984] Karmarkar, N. (1984). A new polynomial-time algorithm for linear programming. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, page 302–311, New York, NY, USA. Association for Computing Machinery.

[Karp, 1972] Karp, R. (1972). Reducibility among combinatorial problems. volume 40, pages 85–103.

[Kohl, 1995] Kohl, N. (1995). *Exact methods for time constrained routing and related scheduling problems*. PhD thesis.

[Levin, 1973] Levin, L. A. (1973). Universal sequential search problems. *Probl. Peredachi Inf.*, 9(3):265–266.

[Lozano et al., 2016] 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.

[Michelini, 2019] Michelini, S. (2019). A comparative study of labeling algorithms within the branch-and-price framework for vehicle routing with time windows.

[Mirabi et al., 2016] Mirabi, M., Shokri, N., and Sadeghieh, A. (2016). Modeling and solving the multi-depot vehicle routing problem with time window by considering the flexible end depot in each route. *International Journal of Supply and Operations Management*, 3(3):1373–1390.

[Righini and Salani, 2006] Righini, G. and Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273. Graphs and Combinatorial Optimization.

[Santos et al., 2019] Santos, M., Amorim, P., Marques, A., Carvalho, A., and Barbosa-Povoa, A. (2019). The vehicle routing problem with backhauls towards a sustainability perspective: a review. *TOP*, 28.

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