UNIVERSITÀ DEGLI STUDI DI PADOVA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

UNIVERSITY OF PADOVA

DEPARTMENT OF INFORMATION ENGINEERING

Master's Degree in Computer Engineering

A Branch-and-Cut based Pricer for the Capacitated Vehicle Routing Problem

Graduate:

Davide PARO

ID: 1207154

Supervisor:

Prof. Domenico SALVAGNIN

Co-Supervisor:

Prof. Roberto ROBERTI

ACADEMIC YEAR: 2021–2022

GRADUATION DATE: July 14th, 2022

A Branch-and-Cut based Pricer for the Capacitated Vehicle Routing Problem

Davide Paro

July 14th, 2022

Document last updated on July 4, 2022

All things are difficult before they are easy.

– Thomas Fuller

Abstract

The *Capacitated Vehicle Routing Problem*, CVRP for short, is a combinatorial optimization routing problem in which, a geographically dispersed set of customers with known demands must be served by a fleet of vehicles stationed at a central facility. *Column generation* techniques embedded within *branch-price-and-cut* frameworks have been the de facto state-of-the-art dominant approach for building exact algorithms for the CVRP over the last two decades. The *pricer*, a critical component in column generation, must solve the *Pricing Problem* (PP), which asks for an *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC) in a reduced-cost network. Little scientific efforts have been dedicated to studying branch-and-cut based approaches for tackling the PP.

The ESPPRC has been traditionally relaxed and solved through dynamic programming algorithms. This approach, however, has two major drawbacks. For starters, it worsens the obtained dual bounds. Furthermore, the running time degrades as the length of the generated routes increases. To evaluate the performance of their contributions, the operations research community has traditionally used a set of historical and artificial test instances. However, these benchmark instances do not capture the key characteristics of modern real-world distribution problems, which are usually characterized by longer routes.

In this thesis, we develop a scheme based on a branch-and-cut approach for solving the pricing problem. We study the behavior and effectiveness of our implementation in producing longer routes by comparing it with state-of-the-art solutions based on dynamic programming. Our results suggest that branch-and-cut approaches may supplement the traditional labeling algorithm, indicating that further research in this area may bring benefits to CVRP solvers.

Sommario

Il *Capacitated Vehicle Routing Problem*, abbreviato come CVRP, è un problema di ottimizzazione combinatoria d'instradamento nel quale, un insieme geograficamente sparso di clienti con richieste note deve essere servito da una flotta di veicoli stazionati in una struttura centrale. Negli ultimi due decenni, tecniche di *Column generation* incorporate all'interno di frameworks *branch-price-and-cut* sono state infatti l'approccio stato dell'arte dominante per la costruzione di algoritmi esatti per il CVRP. Il *pricer*, un componente critico nella column generation, deve risolvere il *Pricing Problem* (PP) che richiede la risoluzione di un *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC) in una rete di costo ridotto. Pochi sforzi scientifici sono stati dedicati allo studio di approcci branch-and-cut per affrontare il PP.

L'ESPPRC è stato tradizionalmente rilassato e risolto attraverso algoritmi di programmazione dinamica. Questo approccio, tuttavia, ha due principali svantaggi. Per cominciare, peggiora i dual bounds ottenuti. Inoltre, il tempo di esecuzione diminuisce all'aumentare della lunghezza dei percorsi generati. Per valutare la performance dei loro contributi, la comunità di ricerca operativa ha tradizionalmente utilizzato una serie d'istanze di test storiche e artificiali. Tuttavia, queste istanze di benchmark non catturano le caratteristiche chiave dei moderni problemi di distribuzione del mondo reale, che sono tipicamente caratterizzati da lunghi percorsi.

In questa tesi sviluppiamo uno schema basato su un approccio branch-and-cut per risolvere il pricing problem. Studiamo il comportamento e l'efficacia della nostra implementazione nel produrre percorsi più lunghi comparandola con soluzioni all'avanguardia basate su programmazione dinamica. I nostri risultati suggeriscono che gli approcci branch-and-cut possono supplementare il tradizionale algoritmo di etichettatura, indicando che ulteriore ricerca in quest'area possa portare benefici ai risolutori CVRP.

RINGRAZIAMENTI

Vorrei innanzitutto ringraziare la mia famiglia per il sostegno datomi e per aver sempre creduto in me. È solo grazie a loro che ora mi trovo qui, alla fine di questo tortuoso percorso di studi. Sono entusiasta di aver raggiunto questi obbiettivi, nemmeno il me stesso del passato ci avrebbe creduto, e invece eccomi qui, ed è solo grazie a voi. Grazie mamma, papà e Massimo. Vi voglio bene. Ringrazio la mia compagna Nicla, per l'affetto e per gli ultimi bellissimi due anni universitari passati assieme. Mi hai aiutato in momenti difficili, fornendomi consigli e aiutandomi a prendere decisioni. Mi riempie di gioia ripensare ai nostri incontri tra le strade padovane durante le pause tra una lezione e l'altra. Oppure le nostre lunghe (circa) sessioni di studio nelle aule Tito Livio e Marsala. Oppure ancora le nostre serate "pizzata" del Venerdì. Fantastici ricordi. Colgo l'occasione per ringraziare anche Emanuela, Claudio e nonna Flavia per l'ospitalità e cordialità fornitami. Ringrazio la struttura Murialdo per avermi ospitato durante la mia carriera universitaria e i Murialdini che sono stati ottimi compagni di cena, di uscite serali e di calcetto. Già che ci sono ringrazio pure la mensa Murialdo, rimani sempre la migliore.

Infine, vorrei ringraziare i miei supervisori Domenico Salvagnin e Roberto Roberti, a cui porgo la mia gratitudine, sia per l'aiuto fornito durante la stesura di questo lavoro, sia per avermi permesso di studiare un problema di ricerca interessante.
I would also like to thank Ruslan Sadykov for his generous assistance in setting up *BaPCod* and reviewing the C++ code of the implemented model.

Padova
July 14th, 2022 Davide Paro

ALLA MIA FAMIGLIA.

Contents

List of Figures

CHAPTER 1

Introduction

1.1 The Capacitated Vehicle Routing Problem

The *Capacitated Vehicle Routing Problem* (**CVRP**), first presented in Dantzig et al. (1959) under the name of "truck dispatching problem", is one of the most studied combinatorial optimization routing problems. The CVRP is an NP-hard (in the strong sense) problem that can be considered a generalization of the well-known Travelling Salesman Problem (TSP). The TSP (Flood, 1956) is an NP-hard (Garey et al., 1976) ubiquitous combinatorial optimization problem in the operations research field, that asks for the determination of a Hamiltonian circuit of minimum cost (Croes, 1958; Laporte, 1992; Johnson et al., 1997; Applegate et al., 2006; Gutin et al., 2006; Hoffman et al., 2013). The CVRP can be defined verbally as finding an optimal route for a transportation/distribution/delivery problem starting from a common point called the depot, where a homogeneous fleet composed of a fixed number of trucks, subject to capacity constraints, need to serve customer demands of a single good (e.g. delivery of gasoline to gas stations). Given as input a weighted graph representing the road network, the customer demands and the vehicle capacity, the problem consists in determining a set of non-oriented routes, one for each vehicle, of minimal overall travel distance starting and ending at the depot. The set of routes needs to serve all the customers in the road network exactly once while satisfying the vehicle capacity bound (Toth et al., 2014). A diagram showing an example of a

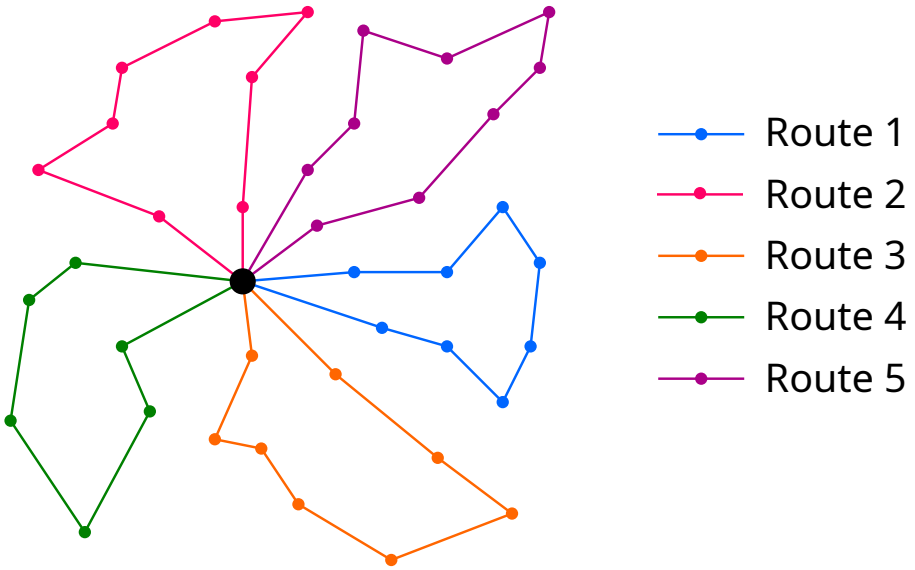


Figure 1.1: An illustration of an optimal CVRP solution (instance named P-n40-k5). There are 39 customers to serve and 5 trucks with a capacity of 140 units each. Each color in the picture represents a different route that each truck has taken. Instead, the colored dots represent the customers served by each truck on its route. The customer demands are not depicted for the sake of clarity. The larger black dot in the center represents the depot where each route must begin and end. Credits: http://vrp.galgos.inf.puc-rio.br/index.php/en/plotted-instances?data=P-n40-k5.

CVRP problem along with its optimal solution is provided in fig. 1.1.

Investigating effective CVRP solution methods may result in significant real-world economic savings for the management of the provision of goods or services in a distribution system. Optimal delivery planning can reduce the overall transportation, goods costs, and waiting time experienced by the customers. As a result, researching efficient exact algorithms and mathematical models for solving and describing real-world distribution problems becomes critical for the operational management of a cost-effective planning process (Toth et al., 2002; Toth et al., 2014).

The CVRP is part of a larger class of problems known as the Vehicle Routing Problems (VRPs). There are many variations of VRPs proposed in the literature, including the *Vehicle Routing Problem with Time Windows* (VRPTW) (Schrage, 1981) and many others. The vehicles in the VRPTW are subject to capacity

constraints and must serve each customer within an allocated time window slot. Nonetheless, CVRP is the simplest VRP variant to describe, and to this day, it remains the most central and studied routing problem. For a comprehensive taxonomy of the many VRP variants, refer to Eksioglu et al. (2009) and Braekers et al. (2016).

While effective (meta-)heuristic algorithms have been proposed and applied successfully to many VRP variants to obtain good-enough solutions in reduced computation time, the focus of this thesis is on exact algorithms for solving the CVRP.

We can find major contributions employing heuristics for the VRP, among others, in Clarke et al. (1964), Desrochers et al. (1989), Paessens (1988), and Foster et al. (1976). Meta-heuristics approaches for the VRP can be found in Gendreau et al. (1994), Cordeau et al. (2012), Toth et al. (2003), Li et al. (2005), Pisinger et al. (2007), Kytöjoki et al. (2007), Nagata et al. (2009), Vidal et al. (2012), and Subramanian et al. (2013), just to name a few.

For a more comprehensive survey on (meta-)heuristics for the VRP, refer to Golden et al. (1998), Gendreau et al. (2002), Gendreau et al. (2008), Laporte et al. (2014), and Elshaer et al. (2020).

Exact algorithms are typically slower than (meta-)heuristics, but given enough computation time, they can produce a proven optimal solution. They accomplish this by closing the objective function's primal-dual bound gap.

We strongly recommend the book "*Vehicle Routing: Problems, Methods, and Applications*" of Toth et al. (2014) for a comprehensive overview/survey on the CVRP and VRPTW problems, as well as other common VRP variants. This book served as a good reference and was instrumental in laying the groundwork for the first chapters of this thesis. We also gathered additional information from other VRP surveys in the works of Cordeau et al. (2007), Baldacci et al. (2012), Caceres-Cruz et al. (2015), and Costa et al. (2019).

1.2 Thesis Contributions

Modern CVRP solvers are typically developed and tested on test sets comprised of historical benchmark instances. The principal historical test instances used to evaluate the performance of the scientific contributions have been divided into families. Each family is represented by a single upper case letter. The major CVRP test sets proposed by the operations research community over the years are summarized here:

- The set `E` is proposed in Dantzig et al. (1959), Christofides et al. (1969), Gaskell (1967), and Gillett et al. (1974). The majority of the `E` test instances were generated at random, while the remainder lack description on their origin.
- The set `M` was proposed in Christofides (1979) and was obtained by aggregating instances from `E` set.
- The set `F` was presented in Fisher (1994) and was obtained from an actual distribution problem of groceries in the city of Ontario.
- The sets `A`, `B` and `P` were all proposed in Augerat et al. (1995). The sets `A`, `B` were generated at random, while the set `P` was generated from sets `A`, `B` and `E` by changing the capacities.

These historical benchmark instances bear little resemblance to real-world distribution problems. They are either too homogeneous or too artificial while not covering the key characteristics found in contemporary real-world distribution problems. The historical instances presented in the literature are typically characterized by stringent vehicle capacities, which give rise to optimal solutions characterized by short routes each visiting few customers. Instead, real-world contemporary distribution problems are characterized by much longer routes since the vehicle capacity is rarely the bottleneck in practice. Despite Uchoa et al. (2017) proposing the `X` test-set, a broader and trendier common denominator of diverse instances for the CVRP, the historical test instances were (and continue to be) the primary central test-bed for comparing and assessing the performance of CVRP contributions.

The *labeling algorithm* is a dynamic programming-based algorithm used to solve the pricing sub-problem induced from *Column Generation* (CG) schemes applied to the VRP. We will delve into these topics in greater detail in chapter 4. The labeling algorithm is a crucial component of the contemporary state-of-the-art CVRP solvers (Gutiérrez-Jarpa et al., 2010; Archetti et al., 2011; Bettinelli et al., 2011; Contardo et al., 2014; Contardo et al., 2015; Pecin et al., 2017a; Pecin et al., 2017b; Pessoa et al., 2020a). As we shall discuss later, the performance of the labeling algorithm degrades as the vehicle capacity increases, see discussion in section 4.3.

In this thesis, we investigate the feasibility and competitiveness of a branch-and-cut algorithm implemented through a commercial MIP software package for solving the pricing problem. The goal is to empirically compare the performance of the proposed branch-and-cut framework against the state-of-the-art labeling algorithm while empirically measuring how the two frameworks behave as the lengths of the route they need to generate increases.

Jepsen et al. (2014) proposed a branch-and-cut framework for the CPTP and tested it in the context of pricing for the CVRP. Jepsen et al. (2014) demonstrated that, while the labeling algorithm was much faster in many instances, the BAC framework exhibited better performance for some bulkier cases. See later discussion in section 5.1. They demonstrated that further research on BAC-pricers could yield approaches that could supplement the traditional pricing labeling algorithm. We revisit the work of Jepsen et al. in this thesis. We test whether recent advancements in MIP optimizers have made them competitive at solving the pricing problem or whether modern dynamic programming algorithms have rendered BAC approaches completely obsolete.

1.3 Outline

This section provides an overview of the work's contents. The thesis' contents can be divided into three major portions, which we will list here.

The first portion provides introductory technical material regarding the CVRP and contemporary resolution methods for this problem. Chapter 2 formalizes the

CVRP through multiple mathematically rigorous integer programming models. Chapter 3 summarizes the notable contributions of the operations research practitioners regarding CVRP's exact approaches. The branch-and-price (BPC) algorithm, column generation, and the pricing sub-problem are presented in chapter 4. The most recent exact algorithms for solving routing problems are all BPC-based. Recent research has empirically demonstrated that BPC algorithms perform admirably in the routing problem domain.

In the second part of the thesis, we will go over the pricing sub-problem in greater detail, as well as our branch-and-cut (BAC) pricer implementation. The pricing sub-problem is a combinatorial optimization problem that arises when a CVRP is solved using a BPC approach. Chapter 5 introduces and discusses the pricing sub-problem through multiple mathematically rigorous integer programming models. The implementation details of our proposed branch-and-cut pricer are presented in chapter 6.

The third part of the thesis evaluates the competitiveness of the proposed BAC framework in solving the pricing sub-problem. We evaluate its performance and compare it to modern cutting-edge solutions. In chapter 7 we present the evaluation setup, we list the instances employed, and finally, we show the empirical results along with a discussion. Finally, in chapter 8 we summarize the work, review the results and offer helpful tips while discussing potential implementation improvements.

CHAPTER 2

Mathematical formulations for the CVRP

Integer Programming (IP) is a mathematical optimization tool that can describe combinatorial optimization problems using constraints, typically defined through linear inequalities. The constraints limit the optimization problem's solution space. IP-based formulations also include a list of decision variables and a linear objective function to be optimized.

While we have already described the CVRP verbally in chapter 1, the CVRP is traditionally defined more rigorously through (Mixed) Integer Programming (MIP, IP) formulations, sometimes also known simply as models. We present three commonly used CVRP mathematical models:

1. The two-index arc flow (2F) formulation (Laporte et al., 1983; Laporte et al., 1985; Laporte et al., 1986) is presented in section 2.2.
2. The three-index arc flow (3F) formulation (Golden et al., 1977) is presented in section 2.3.
3. The Set Partitioning (SP) formulation (Balinski et al., 1964) is presented in section 2.4.

The first two formulations, traditionally designed for branch-and-cut (BAC) algorithms, employ a polynomial number of variables but use an exponential

number of constraints. Instead, the latter formulation employs an exponential number of decision variables. Despite its age, the SP formulation has become the de facto state-of-the-art core ingredient for solving CVRP problems in recent years, thanks to algorithmic advances and the development of efficient branch-and-price (BAP) frameworks (see Pessoa et al., 2020b). Baldacci et al. (2004) contains another less ordinary formulation based on a two-commodity flow description.

While the first two formulations will not be the primary focus of this thesis, they are necessary for understanding some crucial details. We will predominantly focus our efforts on the SP formulation. See chapter 4.

We begin by defining some basic notation and mathematical quantities in section 2.1 that will be used throughout the rest of the thesis.

2.1 Mathematical notation

The CVRP is traditionally described as a node routing problem modeled through a symmetric and complete graph, where: (i) the vertices of the network represent the customers and the depot, (ii) the edges represent road interconnections. Let $G = (V, E)$ denote a **complete undirected network** where: $V = \{0, 1, \ldots, N-1\}$ denotes the set of nodes, $E = \{e = \{i, j\} = \{j, i\} \mid i, j \in V, i \neq j\}$ the set of undirected edges, and N the total number of nodes in the network. The value $0 \in V$ is used to denote the depot node. The edge set E has size $|E|$, which can be computed through combinatorial enumeration: $|E| = \frac{N(N+1)}{2}$. For convenience, we define $V_0 = V \setminus \{0\}$ to express the set of customers, and $N_0 = N - 1$ to denote the total number of customers. Let $\delta(S)$ with $S \subseteq V$ denote the edges crossing the set S and its complement $\overline{S} = V \setminus S$. More formally we can express $\delta(S)$ as $\delta(S) = \left\{e = \{i, j\} = \{j, i\} \in E \mid i \in (S \cap V), j \in \left(\overline{S} \cap V\right)\right\}$. For brevity, we also define $\delta(i) = \delta(\{i\})$ to denote the set of edges incident to node $i \in V$. We also define $E(S) = \{e = \{i, j\} = \{j, i\} \in E \mid i, j \in S\}$ to denote the set of edges having both end points in set $S \subseteq V$.

Let $q_i \in \mathbb{R},\ q_i \geq 0 \quad \forall i \in V$ denote the demand function, which represent the required demand that need to be served for vertex $i \in V$. For convenience, we

use a fictitious demand for the depot: $q_0 = 0$. Given a set $S \subseteq V$, we define $q(S) = \sum_{i \in S} q_i$ as the total demand of the set $S \subseteq V$. Let $c_{ij} \in \mathbb{R}, c_{ij} > 0$ denote the distance function between a pair of nodes $i, j \in V$. The loop arcs $\{i,i\} \notin E$ in CVRP are traditionally not allowed, thus we fix $c_{ii} = \infty$. We assume a Euclidean CVRP problem, i.e. the distance function is symmetric $c_e = c_{ij} = c_{ji}$ and satisfies the triangle inequality $c_{ij} \leq c_{ih} + c_{hj}$. We are also given the total number of identical trucks $K \in \mathbb{N}_+$ and an upper bound $Q \in \mathbb{R}_+$ representing the capacity of each truck. For convenience of notation, we also define $\mathscr{K} = \{1, \ldots, K\}$ to denote the set of trucks. Given a set $S \subseteq V_0$, we denote by $r(S)$ as the minimum number of vehicles required to serve all customers $i \in S$. The value of $r(S)$ can be obtained by solving an NP-hard *Bin Packing Problem* (BPP) associated with the CVRP and set S. As we will see later, it is often simpler to link $r(S)$ to the trivial lower bound of the BPP (Martello et al., 1990a; Martello et al., 1990b):

$$r(S) \geq \left\lceil \frac{q(S)}{Q} \right\rceil. \tag{2.1}$$

A route p is a loop-back sequence $p = (p_0, p_1, \ldots, p_u, p_{u+1})$, with $p_0 = p_{u+1} = 0$ in which $\{p_1, \ldots, p_u\} \subseteq V_0$ customers are visited. The route p has a travel cost of $c_p = c(p) = \sum_{i=0}^{u} c_{p_i, p_{i+1}}$ with resource consumption $q_p = q(p) = \sum_{i=0}^{u} q_i$. A feasible solution to a CVRP problem consists of **exactly** K routes (or circuits) p_k associated with each vehicle $k \in \mathscr{K}$ starting and ending at the depot node, where: (i) each customer is visited once (elementarity condition) and (ii) the demand covered by each route p_k does not exceed the vehicle capacity $q(p_k) \leq Q \quad \forall k \in \mathscr{K}$. An optimal solution to the CVRP minimizes the sum of the overall edge weights across all tours. The traditional definition of CVRP requires that all vehicles be fully utilized. In other words, each p_k route must serve at least one customer. However, we note that using fewer vehicles may sometimes yield better solutions.

The TSP can be considered a special case of CVRP where $Q \geq q(V)$ and $K = 1$. Therefore, all the relaxations and many results obtained for the TSP are valid, or at least extendable, to the CVRP.

Some variants of the basic version of the CVRP, not considered in this thesis, allow using only a subset of the total available vehicles, or consider a hetero-

geneous fleet characterized by different capacities $Q_1, \ldots, Q_k$. The remainder of this section introduces the two most common IP mathematical formulations for the CVRP's classical version.

2.2 The two-index arc flow (2F) formulation

The two-index arc flow (2F) formulation was first presented in Laporte et al. (1983) and Laporte et al. (1985) for the symmetric case, and later generalized to the undirected case in Laporte et al. (1986).

We define a set of integer decision variables $x_e \in \{0,1,2\}$ to indicate the number of times a vehicle traverses each edge $e \in E$ in the optimal solution. The model contains $O(N^2)$ integer variables. The two-index arc flow (2F) formulation can be described as an Integer Program (IP):

$$\min_x \quad z_{2F}(x) = \sum_{e \in E} c_e x_e \tag{2.2}$$

$$\sum_{e \in \delta(i)} x_e = 2 \qquad \forall i \in V_0 \tag{2.3}$$

$$\sum_{e \in \delta(0)} x_e = 2K \tag{2.4}$$

$$\sum_{e \in \delta(S)} x_e \geq 2r(S) \qquad \forall S \subseteq V_0,\ |S| \geq 1 \tag{2.5}$$

$$x_e \in \{0,1,2\} \qquad \forall e \in \delta(0) \tag{2.6}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \setminus \delta(0) \tag{2.7}$$

where $z_{2F}(x)$, as defined in (2.2), is the objective function meant to minimize the overall routing cost (travel time). Constraint (2.3) is the degree constraint which imposes flow conservation: exactly two incident edges must be picked for each customer. Constraint (2.4) is the degree constraint at the depot, it forces that exactly $2K$ incident edges at the depot are picked, thus forcing exactly K routes to be constructed. Constraints (2.6) and (2.7) forces each edge to be traversed at most once, except for all edges incident at the depot. The edge-case modeled in constraint (2.6) is necessary to allow single-customer routes. Constraint (2.5), are the so-called Capacity Cut Constraints (CCC), also called Rounded Capacity

Constraints (RCC), they function both: (i) as Subtour Elimination Constraints (SECs), by imposing connectivity of the solution by avoiding the formation of spurious unconnected subtours and (ii) as a capacity constraint, by imposing that any customer set S is crossed by a number of edges not smaller than $r(S)$. Remember that $r(S)$ represents the minimum number of vehicles required to serve all customers in a given S; additionally, $r(S)$ always satisfies $r(S) \geq 1$ for non-trivial CVRP instances where $q(S) > 0$.

It was shown in Martello et al. (1990b) and Cornuejols et al. (1993), that it is possible to replace $r(S)$ in constraint (2.5), with the much simpler BPP lower bound $\lceil q(S)/Q \rceil$ thus obtaining the following inequality:

$$\sum_{e \in \delta(S)} x_e \geq 2 \left\lceil \frac{q(S)}{Q} \right\rceil \quad \forall S \subseteq V_0, |S| \geq 1. \tag{2.8}$$

The looser inequality of eq. (2.8) is sufficient to solve the 2F formulation optimally. However, a better lower bound for the BPP can improve the linear relaxation, reducing the number of branching occurrences.

The CCC constraints in eq. (2.5), may be transformed in the so called Generalized Subtour Elimination Constraints (GSEC) (Laporte et al., 1985), by means of the degree constraints (2.3) and (2.4):

$$\sum_{e \in E(S)} x_{ij} \leq |S| - r(S) \quad \forall S \subseteq V_0,\ |S| \geq 1, \tag{2.9}$$

where, again, $r(S)$ may be replaced by the trivial BPP's lower bound $\lceil q(S)/Q \rceil$.

The GSECs avoid the formation of spurious unconnected subtours and impose that at least $r(S)$ edges leave set S. The number of GSEC (or CCC) inequalities appear in exponential number in the two-index formulation model, thus making a direct solution of the linear relaxation impractical. To overcome this issue, it is possible to avoid adding the GSEC inequalities statically in the model. Instead, an appropriate cutting-plane algorithm and separation procedure may be employed to generate dynamically only the necessary (violated) GSEC constraints during the running time of the branch-and-cut algorithm.

The so-called compact models (Miller et al., 1960; Christofides, 1979; Desrochers et al., 1991) make use of a polynomial number of constraints.

Unfortunately, the linear relaxations produced by these compact formulations are significantly weaker. The SECs are mathematically proven to be "strong" for the TSP: they are facet-defining for the TSP polytope by uniquely characterizing its convex-hull (Grötschel et al., 1975). Similar arguments, however, cannot be applied to the CVRP due to its more complicated structure. Studying the polyhedral properties of even the standard variation of the VRP has yielded few satisfactory results, see Campos et al. (1991) and Cornuejols et al. (1993).

Unfortunately, as Augerat et al. (1995) demonstrated, the separation problem for the CCCs is NP-complete, limiting the applicability of the 2F formulation for solving the CVRP. As a result, several authors (Augerat et al., 1995; Augerat et al., 1998; Ralphs et al., 2003) developed fractional separation heuristics for the CCC. However, an exact CCC separation is still required for rejecting non-feasible integral solutions and ensuring the correctness of the approach. Lysgaard (2003) implements efficient heuristics for separating the CCC of eq. (2.5).

2.3 The three-index arc flow (3F) formulation

When it comes to modeling more "colorful" CVRP variants, the two-index arc flow (2F) model lacks sufficient descriptive power. For example, the simple CVRP variant where the fleet of trucks is heterogeneous and characterized by capacities $Q_1, \ldots, Q_K$ cannot be described with the 2F formulation, since there's no definite one-to-one mapping on which truck crosses an edge $e \in E$.

The three-index arc flow (3F) formulation, first presented in Golden et al. (1977), fixes this issue. The 3F formulation makes use $O(N^2K + NK)$ integer decision variables. A set of integer variables $x_{ek} \in \{0,1,2\},\ e \in E,\ k \in \mathscr{K}$ is used to encode the number of times a truck k traverses an edge $e \in E$. A new set of binary variables $y_{ik} \in \{0,1\},\ i \in V,\ k \in \mathscr{K}$ is used to model whether truck k covers a node $i \in V$.

The three-index arc flow (3F) formulation can be described as an Integer Program (IP):

$$\min_{x,y} \quad z_{3\mathrm{F}}(x,y) = \sum_{k \in \mathscr{K}} \sum_{e \in E} c_e x_{ek} \tag{2.10}$$

$$\sum_{k \in \mathscr{K}} y_{ik} = 1 \qquad \forall i \in V_0 \quad (2.11)$$

$$\sum_{k \in \mathscr{K}} y_{0k} = K \quad (2.12)$$

$$\sum_{e \in \delta(i)} x_{ek} = 2y_{ik} \qquad \forall i \in V,\ \forall k \in \mathscr{K} \quad (2.13)$$

$$\sum_{i \in V} q_i y_{ik} \leq Q \qquad \forall k \in \mathscr{K} \quad (2.14)$$

$$\sum_{e \in \delta(S)} x_{ek} \geq 2y_{hk} \qquad \forall h \in S,\ \forall S \subseteq V_0,\ |S| \geq 2,\ \forall k \in \mathscr{K} \quad (2.15)$$

$$x_{ek} \in \{0,1,2\} \qquad \forall e \in \delta(0),\ \forall k \in \mathscr{K} \quad (2.16)$$

$$x_{ek} \in \{0,1\} \qquad \forall e \in E \setminus \delta(0),\ \forall k \in \mathscr{K} \quad (2.17)$$

$$y_{ik} \in \{0,1\} \qquad \forall i \in V,\ \forall k \in \mathscr{K} \quad (2.18)$$

where $z_{3F}(x,y)$, as defined in (2.10), is the objective function to be minimized (i.e. the overall travel distance). Constraint (2.11) forces all customers to be served exactly once. Constraint (2.12) forces all the truck tours to start at the depot and end at the same spot. Constraint (2.13) binds the y_{ik} variables to the corresponding x_{ijk} variables, by ensuring that if a truck's route passes through a vertex, then the corresponding node is marked as visited. Constraint (2.14) is the capacity upper bound constraint and it ensures that the demand served by each truck does not exceed the truck capacity. Constraints (2.15) are the Generalized Subtour Elimination Constraints (GSECs), they impose the connectivity of the route and are used to avoid the formation of spurious unconnected subtours. Constraints (2.16) and (2.17) forces each edge to be traversed at most once, except for all edges incident at the depot. The edge-case modeled in constraint (2.16) is necessary to allow single-customer routes. Finally, constraint (2.18) bounds and forces integrality for the y_{ik} binary variables.

Constraint (2.15) may be replaced in an equivalent form with traditional (non generalized) TSP subtour elimination constraints (see Fisher et al., 1981):

$$\sum_{e \in E(S)} x_{ek} \leq |S| - 1 \quad \forall S \subseteq V_0, |S| \geq 2,\ \forall k \in \mathscr{K}. \quad (2.19)$$

Since (2.15), or equivalently (2.19), are exponential in the number of nodes N, they are usually not inserted statically in the model but are instead generated

lazily within the running time of the resolution process.

The three-index arc flow (3F) model generalizes the two-index (2F) version. The 2F formulation may be viewed as a special case of the 3F formulation by aggregating all x_{ek} into a single variable x_e:

$$x_e = \sum_{k \in \mathcal{K}} x_{ek} \quad \forall e \in E. \tag{2.20}$$

However, the three-index arc flow formulation suffers a major downside compared to the two-index version. When modeling homogeneous fleets CVRPs, the 3F model suffers from a multiplicity of the solution space. In fact, since all the vehicles share the same capacity, distinct feasible solutions can be obtained through symmetry by simply permuting the identity $k \in \mathcal{K}$ of each truck.

2.4 The Set Partitioning (SP) formulation

The Set Partitioning (SP) formulation, also known as Path Flow formulation, is an extended formulation originally presented in Balinski et al. (1964). It operates differently from the two/three-index arc flow formulation or many further commonly encountered IP models. The SP formulation uses a tiny number of constraints while offloading much of the search-space complexity into an exponential number of binary variables.

The SP formulation can be viewed as a Dantzig-Wolfe decomposition (Dantzig et al., 1960) and commodity aggregation (Desaulniers et al., 1998) of the three-index arc flow formulation. The Dantzig-Wolfe reformulation approach is an application of a decomposition principle in which one addresses numerous smaller structured sub-problems rather than being confronted with the original problem. This approach is helpful, especially when the original problem's resolution complexity exceeds what can be solved in a reasonable amount of time (Vanderbeck, 2005).

Let $P = \{p \mid p \text{ is a single-truck elementary feasible route}\}$ be the set of all feasible routes. Let $\lambda_p \in \{0,1\}$ be a binary variable indicating whether route p is selected. Let $a_{ip}, a_{ep} \in \mathbb{Z}$ be "static encodings" (integer coefficients) for a route $p \in P$ that respectively count the number of times vertex $i \in V$ or an edge

$e \in E$ is covered. Any $p \in P$ is feasible if it satisfies the following two conditions: $a_{ip} \in \{0,1\} \quad \forall i \in V$ and $q(p) \leq Q$.

We recall that $c_p = c(p)$ represents the cost of a feasible route $p \in P$, which can be trivially computed in $O(N)$. The SP model forces K routes $(p_1, \ldots, p_K) \in P^K$ to be included in the optimal solution.

The SP formulation is described through an Integer Program (IP):

$$\min_{\lambda} \quad z_{\text{SP}}(\lambda) = \sum_{p \in P} c_p \lambda_p \tag{2.21}$$

$$\sum_{p \in P} \lambda_p = K \tag{2.22}$$

$$\sum_{p \in P} a_{ip} \lambda_p = 1 \qquad \forall i \in V_0 \tag{2.23}$$

$$\lambda_p \in \{0,1\} \qquad \forall p \in P \tag{2.24}$$

where, $z_{\text{SP}}(\lambda)$, as defined in (2.21), is the objective function to be minimized (i.e. the overall travel distance). Constraint (2.23) forces each customer to be covered by exactly one route. Constraint (2.22) enforces that exactly K routes are selected. Finally, constraint (2.24) is the bounding and integrality constraints for binary variables $\lambda_p \; \forall p \in P$.

The set of feasible routes P may also be extended to include non-elementary routes without affecting the correctness of the approach. Thanks to (2.23) any route $p \in P$ which does not satisfy $a_{ip} \in \{0,1\} \quad \forall i \in V$ will not belong to any optimal solution regardless of whether the extension is applied or not.

As one might expect, the SP formulation cannot be instantiated or directly solved because of the exponential number of binary variables. A variant of the SP formulation, on the other hand, can be efficiently addressed by Column Generation (CG) approaches embedded within a branch-and-price (BAP) framework. Branch-and-price frameworks and column generation have been applied with notable success to vehicle routing problems. Refer to chapter 4 for a discussion on column generation and BAP frameworks applied to the CVRP.

As Toth et al. (2002) point out, if the distance matrix satisfies the triangle inequality, then it is possible to rewrite the SP formulation into an equivalent Set Covering (SC) formulation by substituting (2.23) in favor of the simpler

inequality:

$$\sum_{p \in P} a_{ip} \lambda_p \geq 1 \quad \forall i \in V_0, \tag{2.25}$$

therefore the SC formulation may be expressed as an Integer Program (IP):

$$\min_{\lambda} \quad z_{\text{SC}}(\lambda) = \sum_{p \in P} c_p \lambda_p \tag{2.26}$$

$$\sum_{p \in P} \lambda_p = K \tag{2.27}$$

$$\sum_{p \in P} a_{ip} \lambda_p \geq 1 \qquad \forall i \in V_0 \tag{2.28}$$

$$\lambda_p \in \{0,1\} \qquad \forall p \in P. \tag{2.29}$$

Under the triangle inequality assumption, any feasible solution for the SC formulation (2.26)–(2.29) is also feasible for the SP formulation (4.1)–(4.4). Transforming the SP to the SC formulation vastly shrinks (halves) the size of the dual solution space.

The SP and SC formulation have two main advantages compared to the formulations presented in previous sections. First, their linear relaxation provides excellent lower bounds (Bramel et al., 1997), compared to the 2F and 3F formulations. Second, they can handle many VRP variants (and more) even described through very complex constraints, since their characterizations are captured within the definition of the set P itself.

CHAPTER 3

Literature Review

This chapter lists some significant contributions regarding exact approaches in the CVRP and VRPTW domains.

Dantzig et al. (1959) were the pioneers of the contemporary vehicle routing problem who introduced the CVRP problem to the masses. A few years later, Clarke et al. (1964) proposes an effective greedy heuristic algorithm for tackling the CVRP. The presentation of this new problem led to a whole new branch of scientific research within the operations research community. In the last 60 years, a huge number of contributors in the operations research field have studied and proposed many mathematical models and algorithms, both exact and meta-heuristic, for solving the VRP.

Until roughly the late 1980s, exact approaches to the CVRP were based on tree-search algorithms employing branch-and-bound schemes (see Pierce, 1969; Christofides et al., 1969; Christofides et al., 1981; Laporte et al., 1986), occasionally employing Lagrangian duality relaxation (see Fisher, 1994; Miller, 1995), or additive bounding procedures (see Fischetti et al., 1994; Hadjiconstantinou et al., 1995).

Some authors applied branch-and-cut (BAC) algorithms to the VRP from the 1980s to the early 2000s. Laporte et al. (1983) and Laporte et al. (1985) proposed the two-index arc flow formulation and a branch-and-cut algorithm for optimally solving the CVRP. Augerat (1995), later proposed a branch-and-cut scheme integrating additional valid inequalities. A few years later, Lysgaard et

al. (2004) proposed a branch-and-cut framework and a set of improved separation procedures for the valid CVRP inequalities that were known at the time. Other notable BAC-based contributions can be found in Araque G et al. (1994), Augerat et al. (1995), Achuthan et al. (1996), Blasum et al. (2000), Ralphs et al. (2003), Achuthan et al. (2003), and Baldacci et al. (2004). Baldacci et al. was able to solve a 135 customers-sized routing problem from the F test-set (Fisher, 1994) for the first time.

Despite the promising results, nonetheless, the branch-and-cut contributions of the time demonstrated that CVRP instances with even a few customers (less than 100) proved exceedingly challenging to solve exactly.

Desrosiers et al. (1984) and Agarwal et al. (1989) are a few very early attempts at applying the *Set Partitioning* (SP) formulation and a *column generation* scheme to routing problems. The *Set Partitioning* formulation and the *column generation* will be presented in more details in sections 2.4 and 4.1 respectively. These authors demonstrated that the column generation scheme proved very satisfactory for solving the VRPTW under tight constraints but did not achieve remarkable results for the CVRP at the time. As a result, until the early 2000s, branch-and-cut approaches were deemed the best approach for solving the CVRP.

Desrochers et al. (1992) proposed a dynamic programming-based label-correcting algorithm for the SPPRC, and devised it within a *branch-and-price* (BAP) scheme to tackle the VRPTW. This labeling algorithm is still a key component of modern BPC schemes for efficiently solving the *pricing problem*. *Branch-and-price* schemes and the *pricing-problem* will be both discussed in the dedicated chapter 4. The 2-paths inequalities were introduced Kohl et al. (1999), which were later generalized in Desaulniers et al. (2008). Kohl et al. (1999) was one of the first works to attempt to integrate cutting planes within a BAP framework for solving the CVRP.

Fukasawa et al. (2006) achieved one of the outstanding breakthroughs in column generation and VRP research. Fukasawa et al. were the first authors to propose a BPC algorithm that incorporated a powerful robust-cuts-based cutting-plane approach normally found in BAC algorithms. Fukasawa et al. integrated the cutting-planes proposed in Lysgaard et al., 2004, and the column genera-

tion considered only q-routes with k-cycles elimination. See chapter 4 and section 4.3 for a thorough introduction. Fukasawa et al. (2006) was able to solve many CVRP instances for up to 100 customers approximately. From the work of Fukasawa et al. until now, the column generation and the label-correcting algorithm have stood as the de facto procedures of modern and efficient VRP solvers.

Baldacci et al. (2008) later proposed a slightly different approach compared to Fukasawa et al. (2006). They employ a column generation scheme considering elementary routes with additional cutting planes. Furthermore, they used a bounding procedure to improve the dual bound and reduce the number of routes to consider. They turned off branching and used *route enumeration* at the root node to generate a comprehensive MIP model to be solved directly. Refer to section 4.2 for additional details regarding *route enumeration*. Pessoa et al. (2008) improves over the work of Fukasawa et al. (2006) by integrating the works of Fukasawa et al. (2006) and Baldacci et al. (2008) through hybridization between traditional branching and enumeration. Instead, Contardo et al. (2011) takes a different approach. It considers q-routes with 2-cycle eliminations and employs non-robust k-CECs cuts to enforce elementarity.

The *ng-routes*, first proposed in Baldacci et al. (2011), are a very effective partial (soft) elementarity relaxation commonly found in contemporary efficient BPCs for the VRP. The ng-routes relaxation significantly reduces the pricing problem's complexity while producing reasonable dual-bounds. Røpke (2012) revisits the work Fukasawa et al. (2006) and applies the ng-routes relaxation and strong branching schemes. Refer to section 4.3 for a thorough introduction to the ng-routes relaxation.

Contardo et al. (2014) outperforms Contardo et al. (2011) by proposing: an effective enumeration scheme with q-routes with 2-cycle elimination, ng-routes relaxation with ng-sets of size 8, non-robust k-CECs cuts to avoid larger cycles and the *decremental state-space relaxation* technique. They were able to solve CVRP instances made up of 151 customers using this approach. The *decremental state-space relaxation* is extended to handle ng-sets in Martinelli et al. (2014), where the size of the ng-sets size is dynamically enlarged to seek better dual bounds. For more information on the *decremental state-space relaxation*,

see section 4.3.

Pecin et al. (2017b) proposed another modern BCP algorithm for solving the CVRP. They solved 200-customers-CVRP instances and with successes in solving some 360-customers instances. Their work featured the following modern components: (i) bidirectional label-setting algorithm for solving the pricing problem, (ii) ng-routes relaxation, (iii) non-robust lm-SRCs cuts, (iv) modern dual smoothing and (v) route enumeration. Pecin et al. (2017a) extends the work of Pecin et al. (2017b) to the VRPTW. On the day of writing, Pessoa et al. (2018a) and Pessoa et al. (2020a) are the current state-of-the-art BCP framework for solving the CVRP based on column generation with ng-routes relaxation. Pessoa et al. (2020a) was able to optimally solve six opened CVRP instances from the X test-set (Uchoa et al., 2017) of up to 548 customers.

Due to significant advances in exact BPC algorithms over the last decades, today, we are able to optimally solve CVRPs with more than 300 customers with ease (Costa et al., 2019).

Once we introduce the pricing sub-problem induced by BAP/BPC frameworks in chapter 4, we will discuss additional contributions regarding this problem domain, see section 4.3. For a literature review on branch-and-cut approaches for pricing, see later discussion in section 5.1.

CHAPTER 4

Branch and Price

In this chapter, we will discuss the branch-and-price (BAP) framework and the *Column Generation* (CG) approach, which are two fundamental components used by modern CVRP solvers. The focus of our presentation will be on these two critical components in the context of the CVRP. For a review of CG approaches to solving arbitrary MIP problems, see Vanderbeck (2005), Lübbecke et al. (2005), and Desrosiers et al. (2011). Moreover, see Barnhart et al. (1998) and Desrosiers et al. (2005) for a primer introduction to branch-and-price schemes. We refer the reader to Feillet (2010)'s work, which includes a helpful tutorial on CG and BAP algorithms specific to VRPs.

Branch-and-price (BAP) frameworks are in essence a branch-and-bound (BAB) scheme (Land et al., 2010) that originates when solving the SP/SC formulation for VRPs (see section 2.4). To seek the optimal solution, BAB-based schemes use a search tree and a pruning strategy by bounding the objective function. As opposed to widerspread branch-and-cut (BAC) frameworks, their primary focus is the usage of a Column Generation (CG) technique for improving the dual bound, see Righini et al. (2008). BAP frameworks were first applied successfully to the Cutting Stock problem in Gilmore et al. (1961). The so-called branch-price-and-cut (BPC) extends the traditional BAP framework. Additional cutting planes are added in the BPC framework to strengthen the linear relaxations associated with each node, further improving the dual bounds.

According to Sadykov (2019), BAC approaches struggle to solve problems

for which the linear relaxations are not sufficiently tight, even when adding cutting planes. This is especially true for problems with complex combinatorial structures, such as routing. Substantially increasing the number of decision variables is understood to improve the linear relaxations. Unfortunately, the large number of decision variables necessitates a method for dynamically generating them. Branch-and-price extends traditional BAC frameworks by dynamically generating the required decision variables via *Column Generation*.

In the following section 4.1 we will illustrate the column generation algorithm and the pricing problem, two vital pieces in branch-and-price frameworks.

4.1 Column generation and the Pricing Problem

Consider the *Master Problem* (MP) defined as the linear relaxation of the SP formulation (2.21)–(2.24) obtained by relaxing the integrality constraints:

$$\min_{\lambda} \quad z_{\text{MP}}(\lambda) = \sum_{p \in P} c_p \lambda_p \tag{4.1}$$

$$\sum_{p \in P} \lambda_p = K \tag{4.2}$$

$$\sum_{p \in P} a_{ip} \lambda_p = 1 \qquad \forall i \in V_0 \tag{4.3}$$

$$0 \leq \lambda_p \leq 1 \qquad \forall p \in P. \tag{4.4}$$

The constraint $\lambda_p \leq 1$ in (4.4) is implied from (4.3) and can thus be removed. An optimal solution to the MP provides a valid dual bound at the root node of the search tree. The MP can be extended by adding additional cuts or branching constraints with the aim of improving the dual bounds.

The dual problem associated with (4.1)–(4.4) is:

$$\max_{\pi} \quad z_{\text{DMP}}(\pi) = K\pi_0 + \sum_{i \in V_0} \pi_i \tag{4.5}$$

$$\pi_0 + \sum_{i \in V_0} a_{ip} \pi_i \leq c_p \qquad \forall p \in P \tag{4.6}$$

$$\pi_0 \in \mathbb{R} \tag{4.7}$$

$$\pi_i \in \mathbb{R} \qquad \forall i \in V_0, \tag{4.8}$$

where $\pi_0 \in \mathbb{R}, \pi_i \in \mathbb{R} \quad \forall i \in V_0$ represents the dual variables associated respectively with constraints (4.2) and (4.3). In case the SC formulation (2.26)–(2.29) is used to define the MP, eq. (4.8) in (4.5)–(4.8) can be replaced in favor of $\pi_i \geq 0 \quad \forall i \in V_0$, thus slimming the dual space.

When solving the MP, at each iteration of the *simplex algorithm* (Dantzig et al., 1955) we seek a non-basic variable known as *column*, to price out and enter the basis through evaluation of the dual variables $\pi \in \mathbb{R}^N$. Due to the enormous size of the set of routes P, evaluating the dual variables $\pi \in \mathbb{R}^N$ is computationally intractable.

As a result, in BAP frameworks we consider only a small subset of columns $\mathscr{P} \subseteq P$, yielding the following linear program:

$$\min_{\lambda} \quad z_{\text{RMP}}(\lambda) = \sum_{p \in \mathscr{P}} c_p \lambda_p \tag{4.9}$$

$$\sum_{p \in \mathscr{P}} \lambda_p = K \tag{4.10}$$

$$\sum_{p \in \mathscr{P}} a_{ip} \lambda_p = 1 \qquad \forall i \in V_0 \tag{4.11}$$

$$\lambda_p \geq 0 \qquad \forall p \in \mathscr{P}, \tag{4.12}$$

which takes the name of *Restricted Master Problem* (RMP).

We look for a column to enter the basis, which in turn necessitates the resolution of the following sub-problem:

$$c_p^\star = \min_{p \in P} \left\{ \bar{c}_p = \sum_{e=\{i,j\} \in E} \left(c_e - \frac{\pi_i + \pi_j}{2} \right) a_{ep} \right\}, \tag{4.13}$$

which takes the name of *Pricing Problem* (PP). In (4.13), $\bar{c}_p$ denotes the reduced cost of a route $p \in P$ and $c_p^\star$ is the reduced cost of the optimal route $p^\star \in P$ that leads to the best dual bound improvement. Notice that $\bar{c}_p$ can also be expressed as $\bar{c}_p = c_p - \pi_0 - \sum_{i \in V_0} \pi_i a_{ip}$. For convenience, we define $\bar{c}_e = c_e - \frac{\pi_i + \pi_j}{2}$ as the reduced cost of an edge $e \in E$. The purpose of the RMP is to provide dual variables to be transferred to the pricing sub-problem (Lübbecke et al., 2005).

Any $p \in P$ which satisfies $\bar{c}_p < 0$ is a valid column which can enter the basis of the RMP. The BAP solver must intelligently manage the set $\mathscr{P} \subseteq P$ that it

stores within a pool. During the resolution process, the BAP framework manages which routes to keep or drop from $\mathscr{P}$. Because the set $\mathscr{P}$ is generally sparse, any $p \in \mathscr{P} \mid \lambda_p = 0$ is a valid candidate to be dropped from the pool. A BAP framework must be appropriately engineered to maintain the size of the pool $\mathscr{P} \subseteq P$ manageable.

In the literature, the resolution method, or algorithm, used to solve the PP is known as *oracle* or *pricer*. The oracle needs to solve the pricing sub-problem which, due to the definition of the set P, coincides with solving an *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC) over a **new directed symmetric** network with weights $\bar{c}_{ij} = c_{ij} - \frac{1}{2}\pi_i - \frac{1}{2}\pi_j$. Resources are monotonically increasing quantities, such as capacity or time, which restrict the feasibility of a route. The ESPPRC asks for an elementary route connecting a *source vertex* and a *sink vertex* that satisfies the resource consumption bounds. Because of the elementarity condition, the route can only visit each vertex at most once. In the ESPPRC, the depot node is split into two vertices (one source, one sink) and the newly obtained network is characterized by $N+1$ vertices. Chapter 5 will go over the PP and the ESPPRC in more details.

The *Column Generation* (CG) is an iterative algorithm, successfully applied to a wide variety of problems (Desrosiers et al., 2005), which alternates between two phases (Desaulniers, 2018):

1. The simplex algorithm for solving the RMP, which is characterized by one or more pivot operations.
2. One or several iterations of the Pricing Problem (PP) solved by invoking the pricer algorithm. The simplex algorithm invokes the pricer to verify whether other pivot operations can be performed for improving the dual bound. The pricer algorithm usually lives in a separate external code module from the branch-and-price code.

The column generation may also be interpreted as a cutting-plane strategy for the dual problem. The column generation procedure stops mainly under two scenarios: (i) when no more negative reduced cost routes exist, i.e. the PP outputs a $p^\star \in P$ achieving $c_p^\star \geq 0$ or (ii) the CG procedure tails off, i.e. the

gained dual bound improvements compared to the running time to generate a column become suboptimal (optional).

If after the column generation procedure the candidate solution to the RMP λ^* satisfies the integrality constraints, $\lambda_p^\star \in \{0,1\} \quad \forall p \in \mathscr{P}$, no branching or cutting planes are required and the candidate solution λ^* can update the incumbent. If, instead, $\exists p \in \mathscr{P} \mid \lambda_p^\star \notin \{0,1\}$ then a branching strategy is usually necessary to seek the best integral solution. When all the nodes of the branch-and-bound tree have been explored or pruned, then the CVRP problem is solved to optimality and the incumbent solution becomes the optimal solution $(p_1,\ldots,p_K) \mid \lambda_{p_i}^\star = 1$.

It's also worth noting that the CG doesn't have to solve the PP optimally every time the simplex algorithm advances. Finding any $p \in P$ that achieves $\bar{c}_p < 0$ is usually enough to improve the dual bound. Except for proving the optimality at the last CG iteration, there is commonly no need to solve the pricing problem exactly. Therefore, fast and efficient heuristic algorithms may be employed to tackle the pricing problem, especially during the very first few pricing iterations, where finding reduced cost columns is effortless.

4.2 Branching, Route Enumeration and Cut Generation

Solving the RMP at the root node rarely suffices to satisfy the integrality constraints for the SP formulation (2.21)–(2.24). The CG algorithm may conclude with a non-zero duality gap. As a result, a search tree and a branching scheme are used to find the best integral solution $\lambda_p^\star \in \{0,1\} \quad \forall p \in P$. Modern BAP frameworks employ additional cutting planes to improve the dual bounds and the overall convergence speed. The so-called branch-price-and-cut (BPC) combines common elements from BAP and BAC frameworks. Fukasawa et al. (2006) and Røpke (2012) are two notable works that integrate cutting planes and propose a BCP approach for the VRP. See Sadykov (2019) for a discussion of the cutting-edge components that comprise modern BAP/BPC frameworks.

However, it's important to note that branching isn't a rigidly required oper-

ation. *Route enumeration* (RE), first proposed for the CVRP in Baldacci et al. (2008), is a technique complementary to branching. In RE, all elementary routes that may be part of the VRP optimal solution are identified and later incorporated into an exhaustive SP formulation. The SP formulation is then solved using a standard MIP optimizer. Enumeration is accomplished through a dynamic programming labeling algorithm similar to the one employed for the PP (see the subsequent discussion in section 4.3). The issue with RE is the potentially large number of routes that needs enumeration. This number is directly proportional to the tightness of the duality gap and the average length of a tour N/K (Toth et al., 2014). Route enumeration is an operation that requires exponential space complexity, but it can drastically improve the performance of the BPC in some circumstances. Baldacci et al. (2008) and Baldacci et al. (2011) are two considerable BAP contributions replacing branching entirely in favor of RE.

Modern BPC frameworks instead take a hybrid approach by combining branching and route enumeration (RE) (see Pessoa et al., 2008; Pessoa et al., 2009; Contardo et al., 2014; Pecin et al., 2017b; Pecin et al., 2017a; Pessoa et al., 2020b). Route enumeration is attempted after the column generation has converged. If the number of enumerated routes exceeds a predefined limit, the RE is preemptively aborted, and traditional branching is applied instead. Once the primal-dual bound gap $\bar{z} - \underline{z}$ eventually reaches reasonable levels, the RE strategy will enumerate all routes under the maximum limit. The associated RMP formulation can now be solved efficiently using a standard MIP optimizer.

We will not go into detail about route enumeration in this thesis because its presence has no impact on the pricing problem. Nonetheless, acknowledging its existence and role within a branch-and-price framework is essential.

Due to the PP's existence, branching and cut generation in BCP frameworks are more delicate operations than in traditional BAC frameworks. According to de Aragao et al. (2003), there are two types of inequalities that can occur depending on the branching and cut generation schemes utilized: **robust** and **non-robust** inequalities.

A **robust** inequality is an inequality which can be safely added to the RMP without altering the structure of the set of feasible paths P (Fukasawa et al.,

2006). That is, robust inequalities do not require explicit modeling in the PP formulation and instead manifest their contribution directly in the reduced cost edges $\bar{c}_e$. An oracle, therefore, after the introduction of a robust inequality in the RMP, needs to solve the same ESPPRC problem but with slightly different weights associated with each edge.

A **non-robust** inequality, on the other hand, is significantly harder to cope with because it influences the structure of the set of possible paths P. In general, non-robust inequalities have an enriched potential to lower the integrality gap, but they require explicit modeling and oracle support. These inequalities can significantly complicate the PP by exchanging better dual-bound improvements for longer column generation times. As a result, their inclusion should be evaluated on an individual basis (Desaulniers et al., 2011). Non-robust inequalities can arise during both the branching and cut-generation phases.

These work's contributions will predominantly focus on **robust inequalities** exclusively.

4.2.1 Branching

In this section, we will introduce branching for the SP formulation, under the assumption that a branch-and-price scheme is employed for solving the CVRP.

At first glance, it might seem reasonable to branch directly on the λ_p variables of the SP formulation. Unfortunately, such branching has two significant drawbacks (Vanderbeck et al., 2010). For starters, it is not robust and leads to a harder PP: an ESPPRC with forbidden paths (Villeneuve et al., 2005). Second, it results in an utterly unbalanced search tree.

A vast bulk of CVRP branching approaches investigated in the literature are merely extensions of TSP branching schemes. *Branching on edge*, first applied to the CVRP in Christofides et al. (1969), is one of the most basic branching schemes, but it is also one of the most effective. The branching on edges scheme was refined in diverse forms in Fisher (1994) and Miller (1995). The edge flow decision variables can be calculated from the MP by noting that:

$$x_e = \sum_{p \in P} a_{ep} \lambda_p \quad \forall e \in E, \tag{4.14}$$

where a_{ep} represents the number of times route $p \in P$ crosses an edge $e \in E$.

Given a candidate fractional solution $0 \leq \lambda_p^\star \leq 1 \quad \forall p \in \mathscr{P}$ and a valid edge $e \in E \mid x_e \notin \{0,1,2\}$ to branch on, it is possible to generate two descendant nodes in the search tree to continue scanning for integral solutions:

$$x_e \leq \left\lfloor \sum_{p \in \mathscr{P}} a_{ep} \lambda_p^\star \right\rfloor, \quad x_e \geq \left\lceil \sum_{p \in \mathscr{P}} a_{ep} \lambda_p^\star \right\rceil. \tag{4.15}$$

Branching on edges is an approach that is both extremely simple and robust. Let $\omega_e \in \mathbb{R}$ denote the dual variable associated with edge $e \in E$ and one of the branching polarities in eq. (4.15), then the new reduced cost $\bar{c}_e$ for an edge $e \in E$ after branching is:

$$\bar{c}_e = c_e - \frac{\pi_i + \pi_j}{2} - \omega_e, \tag{4.16}$$

where $\pi \in \mathbb{R}^N$ represents the dual variables associated with eqs. (4.2) and (4.3).

Branching on edges has the disadvantage of achieving only local changes to the fractional solution. For this reason, Augerat et al. (1998) propose an approach to obtain larger perturbations. They achieve this by branching on Rounded Capacity Constraints (RCCs) defined over an arbitrary set $S \subseteq V_0$ with two descendant nodes:

$$\sum_{e \in \delta(S)} x_e \leq 2r(S), \qquad \sum_{e \in \delta(S)} x_e \geq 2r(S) + 2 \tag{4.17}$$

Pecin et al. (2017b) apply a similar branching approach.

Ryan and Foster branching (Ryan et al., 1981) is another viable branching rule which is unfortunately non-robust. *Ryan and Foster branching* complicates the pricing subproblem, but it has the primary benefit of resulting in more balanced search trees. *Branching over the accumulated resource consumption*, proposed in Gélinas et al. (1995), is another viable branching rule that, unlike the previous one, it does not raise the pricing difficulty.

Strong branching is a commonly employed strategy, which evaluates individually potential branching candidates by measuring their impact based on some scoring function. The goal is to choose either the branching candidate that heuristically leads to the best dual bound improvement or the branching candidate that heuristically leads to a more balanced search tree. Strong branching

can reduce the size of the search tree at the expense of a slower branching candidate evaluation. Fukasawa et al. (2006), Pecin et al. (2017c), and Pecin et al. (2017a) are all approaches that opted for the strong branching technique for solving the VRP.

Reduction rules can remove edges verified to not belong to an optimal solution. *Variable fixing* can be used as a reduction rule by utilizing a dual bound $\bar{z}$ and primal bound $\underline{z}$ to eliminate some edges $e \in E \mid \hat{c}_e \geq \bar{z} - \underline{z}$ (Hadjar et al., 2006; Irnich et al., 2010). Reduction rules are a peculiar form of branching where only single descendant nodes are generated. Reduction rules may crop a good portion of the search space.

4.2.2 Cutting Planes

In this section, we will introduce numerous families of inequalities which can be used for cut generation for the 2F (2.2)–(2.7) and the SP (2.21)–(2.24) formulations. See Desaulniers et al. (2011) for a general framework on cutting planes for CG algorithms for arbitrary integer programs.

While the 2F formulation is not the primary focus of this thesis, presenting and studying valid inequalities for such a formulation will be useful for the PP induced during the CG phase. Many 2F valid inequalities can be applied to the ESPPRC if properly rewritten. The early attempts to propose valid inequalities to improve the linear relaxation for the 2F problem were obtained by generalizing the constraints for the traditional TSP problem, see Naddef et al. (1993). Naddef et al. demonstrated that by putting any valid inequality obtained for the TSP into a proper tight triangular form, it is possible to re-adapt it to the CVRP. As an example, the comb inequalities readjusted from the original TSP (Chvátal, 1973; Grötschel et al., 1979; Augerat, 1995).

Valid inequalities for the 2F CVRP formulation can be classified into several groups (Toth et al., 2014). We concentrate on the following groups: capacity constraints and multistar inequalities.

The **capacity constraints** are inequalities which can be expressed in the form $\sum_{e \in \delta(S)} x_e \geq 2r(S) \quad \forall S \subseteq V_0, |S| \geq 2$. This set of constraints may have different names depending on how $r(S)$ is defined. If $r(S) = \sum_{i \in S} q_i / Q$, we obtain the

so-called Fractional Capacity Constraints (FCC). Instead, if we use the typical Bin Packing Problem (BPP) lower bound, $r(S) = \lceil \sum_{i \in S} q_i / Q \rceil$, we obtain the so-called Rounded Capacity Constraints (RCC). The RCC were first presented in Laporte et al. (1983) for the CVRP 2F formulation.

The **multistar inequalities** were initially proposed in Araque et al. (1990) for the *CVRP with unit demands*, then extended to the generic CVRP in Gouveia (1995) and Achuthan et al. (1998), and finally generalized in the so called Generalized Large Multistart (GLM) for the CVRP in Letchford et al. (2002) and Letchford et al. (2006). Given two disjoint sets of customers $A, B \subseteq V_0 \mid A \cap B = \emptyset$, and $\alpha, \beta, \gamma \in \mathbb{R}$, a multistar inequality is described in the 2F formulation using the following template:

$$\alpha \sum_{e \in E(A)} x_e + \beta \sum_{e \in (\delta(A) \cap \delta(B))} x_e \leq \gamma. \tag{4.18}$$

Araque G et al. (1994) were the first to successfully apply multistar inequalities within a branch-and-cut framework. Letchford et al. (2006) demonstrates that GLMs are implied by the SP model, even if the set of feasible paths P is relaxed to contain non-elementary routes.

We will terminate the remainder of this section by introducing some families of inequalities serviceable to improve the linear relaxation for the SP formulation (2.21)–(2.24). Despite the SP formulation generally delivers better dual bounds compared to other CVRP formulations (see discussion in section 2.4), these bounds may still be too weak to obtain an efficient algorithm. Thus, it is common practice to introduce cutting-plane approaches in modern BAP-based VRP solvers.

Cutting planes that only involve edge-flow variables are by nature robust. Because the RCC inequalities only involve the edges $e \in E$ crossing a set $S \subseteq V_0,\ |S| \geq 2$, they trivially can be extended to a robust inequality for the SP formulation:

$$\sum_{p \in P} \sum_{e \in \delta(S)} a_{ep} \lambda_p \geq r(S), \tag{4.19}$$

where $\sum_{e \in \delta(S)} a_{ep}$ counts the number of times a route $p \in P$ enters or leaves a set S and $r(S)$ represents the number of trucks needed to serve each customer

$i \in S$. Lysgaard (2003) proposes a heuristic algorithm for efficiently separating the RCC inequalities.

Robust cuts are moderately effective at closing the integrality gap, but they may not be sufficient for some complicated VRP instances. Consequently, some researchers attempted to apply non-robust cuts to the CVRP with promising success. Non-robust cuts must be separated carefully to use them effectively. Taking into account the aggravation of the pricing problem during the separation of non-robust cuts is of vital importance for the overall performance of the BAP scheme. While we will mostly avoid non-robust cuts in this thesis, it is noteworthy to acknowledge their significance within a BAP framework.

Traditionally, these types of cutting planes have been handled by extending the dynamic programming labeling algorithm through the addition of fictitious resources and modifications to the dominance rules. See discussion in section 4.3 for more details.

Major contributions in the non-robust cutting planes domain for the VRP are:

1. **Strengthened Capacity Cuts (SCCs)**, sometimes also called *Strong Capacity Cuts* were first proposed in Baldacci et al. (2008),. These cutting planes can be considered a non-robust tighter version of the RCCs. Compared to RCCs, they are not affected by routes entering set $S \subseteq V_0$ more than once.
2. **Extended Capacity Cuts (ECCs)** proposed in Pessoa et al. (2008) and Pessoa et al. (2009) are another pertinent generalization of the RCCs.
3. **Subset Row Cuts (SRCs)** proposed in Jepsen et al. (2008a), are non-robust inequalities obtained as a subset of the Chvátal-Gomory rank-1 cuts (Chvátal, 1973) applied to a subset $C \subseteq V_0$ of customers. Their separation is NP-hard. Therefore, the subset C is typically separated through enumeration and for computational tractability the evaluation is restricted solely to $\forall C \subseteq V_0$ that satisfy $|C| \leq 5$. 3-SRCs, namely SRCs where $|C| = 3$, were applied successfully in many works (Desaulniers et al., 2008; Jepsen, 2011; Baldacci et al., 2011; Contardo et al., 2014; Pecin et al., 2017b). The SRCs were later generalized to a more general Chvátal-Gomory Rank-1 cuts (R1Cs) in Petersen et al. (2008). R1Cs have the potential to be ex-

tremely powerful, but they make pricing subproblems significantly more difficult. Baldacci et al. (2011) propose a weakened variant of the 3-SRCs inequalities.

4. **Limited Memory Subset Row Cuts (lm-SRCs)** proposed in Pecin et al. (2017b), are a dynamically controlled weakening of the SRCs that has far less computational impact on the PP. Later, Pecin et al. (2017c) extend this reasoning and develop the so-called Limited Memory Rank-1 Cuts (lm-R1Cs).
5. **Strong Degree Cuts (SDCs)** proposed in Contardo et al. (2011) and Contardo et al. (2014), are cutting planes which can be considered a specialized version of the SCCs. They can enforce partial route elementarity in the RMP formulation and are redundant when the pricer always outputs elementary paths. SDCs can be used to achieve the elementarity bound in the case the pricer produces non-elementary routes (Contardo et al., 2014).
6. **k-Cycle Elimination Cuts (k-CECs)**, proposed in Contardo et al. (2014), are a weakened form of the SDCs, which forbids routes that have cycles of length smaller or equal to k over a customer $i \in V_0$. SDCs may be viewed as a special case of k-CECs with $k = \infty$. The k-CECs have less impact in the pricing problem compared to SDCs.
7. **Clique inequalities** readjusted for the VRPTW problem domain in Spoorendonk et al. (2010).

For a more complete survey on the various non-robust cutting planes proposed over the years for the CVRP refer to Costa et al. (2019).

4.3 Solving the Pricing Problem

The Pricing Problem (PP) asks for the determination of an elementary shortest path, subject to resource constraints, over a directed network formulation where the cost of each edge is dualized. In the ESPPRC problem, the elementarity condition imposes that the optimal route may visit the same vertex at most once. For convenience we define as *Shortest Path Problem with Resource Constraints* (SPPRC) as the non-elementary version of the ESPPRC and the abbreviation

RCSP to denote a *resource constrained shortest path*. Sometimes SPPRC is also abbreviated as RCSPP: *Resource Constrained Shortest Path Problem*. As it was claimed in Dror (1994), since the underlying network may contain negative cost cycles, the ESPPRC is an NP-hard problem.

The q-routes (Christofides et al., 1981) are routes p with total resource consumption $q(p) \leq Q$ that do not necessarily satisfy the elementarity condition and may thus contain cycles. In the q-routes relaxation, each additional visit to already covered customers counts toward the total resource consumption. In the original definition of q-routes cycles of two vertices are prohibited. The q-routes with k-cycle elimination are a natural extension of the q-routes in which multiple visits to the same vertex are allowed only if at least k other vertices are covered in between.

It was discovered that broadening the set of feasible paths P in the SP formulation to include q-routes could result in a more tractable pricing problem by making it weakly NP-hard and thus solvable via a pseudo-polynomial algorithm (Desrochers et al., 1988; Irnich et al., 2005). Notice that the q-routes violates the SP constraints (2.23), thus, they won't be part of a CVRP optimal solution. Many efficient algorithms for the VRP rely on the solution of SPPRC subproblems which make use of q-routes (Desrochers et al., 1992; Fukasawa et al., 2006; Contardo et al., 2011). While this expansion of the set of feasible paths P is still valid, it has the unfortunate side effect of yielding significantly worse dual bounds for the linear relaxations (Feillet et al., 2004). As a result, the definition of the set of feasible paths P is critical in balancing efficiency at the pricing stage with the overall quality of the dual bounds.

The label-correcting algorithm was first proposed in Desrochers et al. (1992) for solving an SPPRC pricing sub-problem induced from the Set Covering (SC) formulation of the VRPTW. The label-correcting algorithm, which could only generate q-routes at the time, is a dynamic programming algorithm that works through label construction, propagation, and correction from node to node. Each partial path is tracked by the algorithm using labels. Each label encodes the current partial resource consumption, among other things. The algorithm's dominance rules allow it to efficiently prune the search space by ignoring dominated

labels, thus vastly speeding up the resolution process. The label-correcting algorithm is a natural extension of the traditional Bellman-Ford algorithm (Bellman, 1958; Ford Jr, 1956), but it takes additional path constraints into account. The Bellman-Ford algorithm is commonly known by all operations research practitioners and it is used to compute resource-unconstrained shortest paths on a negative-cycles-free network. The running time complexity of the label-correcting algorithm for pricing q-routes is $O(N^2Q)$ operations. The number of processed labels scales with the vehicle capacity. By nature of the approach, the applicability of the labeling algorithm is limited to VRPs having stringent vehicle capacity and characterized by routes visiting a mediocre amount of customers each (Jepsen et al., 2008b). On top of that, due to its inherent nature, the labeling algorithm is limited by single-threaded performance and cannot scale to multiple machine cores.

Feillet et al. (2004) extends the label-correcting algorithm to handle elementary paths by introducing additional fictitious resources for each node. As the authors point out, this new algorithm has two advantages. First, the linear relaxation produces a significantly smaller duality gap than treating q-routes. Second, it may allow for the resolution of certain problem classes where relying on column generation schemes based on q-routes is hardly an option. The extended algorithm of Feillet et al. for pricing elementary routes has a running time complexity of $O(2^NQ)$ operations, limiting its applicability to larger problems. Righini et al. (2006) later improve the dynamic programming algorithm of Feillet et al.

Nonetheless, some improvements have been made to the overall running time of the label-correcting algorithm for pricing elementary routes. Some significant contributions in this domain can be attributed to: (i) *bidirectional search* proposed in Righini et al. (2006) and (ii) *decremental state-space relaxation* (DSSR) proposed independently in Boland et al. (2006) and Righini et al. (2008) under different names. The *decremental state-space relaxation* (DSSR) is an approach where elementarity impositions are included lazily during the running time of the column generation phase. In DSSR, the label-correcting algorithm is first asked to generate q-routes, then for each vertex that is visited twice or more, elementarity is enforced at the pricer stage and the procedure is repeated. The first BCP

algorithm based on full elementarity routes was proposed in Chabrier, 2006, but unfortunately, applying arbitrary k-cycle elimination to the pricing problem has not been thoroughly researched since. Despite arbitrary k-cycle elimination can also be enforced by using non-robust cuts (see discussion in section 4.2.2), Feillet et al. demonstrated that avoiding such cycles within the column generation procedure supplied substantially better dual bounds at the cost of increasing the running time of the pricing problem.

Given the complexity of the ESPPRC, many authors inspired by these works have investigated using some form of controlled **partial elementarity** condition. The idea behind partial elementarity is to obtain dual bounds as close to the elementary route bound as possible while still keeping the pricing problem manageable (Contardo et al., 2015).

Irnich et al. (2006) and Fukasawa et al. (2006) have investigated the usage of q-routes with k-cycle elimination with $k \geq 3$ within the labeling algorithm. Fukasawa et al. (2006) tests the q-routes with 4-cycle elimination and shows that such an approach leads to a substantially slower pricing problem with dual bounds that are extremely far from the elementary bound. Instead, Desaulniers et al. (2008) takes a totally different approach. They introduce a new concept of "partial elementarity" obtained by enforcing the elementarity condition only on a small predetermined subset of customers.

The *ng-routes relaxation* developed in Baldacci et al. (2011) is credited as one of the radical innovations in modern pricing efficiency. Baldacci et al. observed that in most cases, q-routes cycles emerged to use only low-cost edges and were confined to comparatively small clusters of the graph. They define the concept of *ng-set* N_i for a vertex i which encodes the $|N_i|$ closest neighbors of the vertex $i \in V$. The quantities $|Ni|$ can be increased on-demand to achieve better dual bounds, but doing so leads to an exponential increase in pricing complexity. The ng-sets can be determined a priori or dynamically, as shown in Roberti et al. (2014). The ng-routes permits cycles over a vertex $i \in V$ only if the route passes through a vertex $j \in V \mid i \notin N_j$. The new novel relaxation of Baldacci et al. is simple to incorporate into the dynamic programming labeling algorithm. Dominance is verified through a limited-memory set that changes as the route

get extended. The limited memory, by its nature, "forgets" old covered vertices, which can then be subsequently revisited as the generated path gets longer. The ng-routes relaxation can be considered a more versatile and dynamic version of q-routes with k-cycle elimination. They use a different definition of cycle length. This definition provides a better measure of a cycle's impact and may be regarded as the primary cause of this relaxation's efficiency. The ng-routes relaxation measures cycle lengths in terms of travel or time distances. In contrast, in classical k-cycle relaxation, they are measured in terms of customers visited (Contardo et al., 2014). This new novel relaxation achieves a bound that is very close to the elementary bound while not being detrimental to the pricing problem. Furthermore, depending on the circumstances, the amount of elementarity can be controlled by selecting the appropriate neighborhood set size $|N_i|$.

We conclude this section by mentioning a few final contributions that are related to the pricing problem.

Fukasawa et al. (2006) modifies the label-correcting algorithm to employ it as a fast pricing heuristic. Heuristics can significantly speed up the column generation process. The invocation of the exact pricer occurs solely when the pricing heuristic fails to determine a valid reduced-cost route. Desaulniers et al. (2008) and Archetti et al. (2011) propose a meta-heuristic based on tabu search to generate routes with a negative reduced cost.

Lozano et al. (2013) propose the *pulse algorithm* to solve the SPPRC. Lozano et al. (2016) extend the pulse algorithm to handle elementary paths.

Desaulniers et al. (2019) propose a new paradigm called *selective pricing* for the ng-SPPRC. The column generation procedure is preemptively aborted in selective pricing if there is proof that no reduced-cost elementary routes exist, regardless of whether the ng-SPPRC admits reduced-cost ng-routes. This method allows for rejecting some non-elementary routes even when they are not dominated.

The dual variables $\pi \in R^N$ tend to oscillate from one CG iteration to the next as the average number of customers per route N/K increases, slowing the convergence speed of the column generation algorithm (Toth et al., 2014). As a result, some dual variable stabilization techniques were proposed to address this

issue and reduce the number of pricing iterations required by the column generation algorithm (see Du Merle et al., 1999; Rousseau et al., 2007; Pessoa et al., 2013; Pessoa et al., 2018b). For a general introductory discussion on column generation stabilization techniques, see Vanderbeck (2005). When the SC formulation is used instead of the SP formulation, the dual variables are typically more stable (Rousseau et al., 2007; Feillet, 2010).

We conclude the section by mentioning additional surveys discussing labeling algorithm for tackling the non-elementary SPPRC: Irnich et al. (2005), Pugliese et al. (2010), and Pugliese et al. (2013).

CHAPTER 5

The Pricing Problem

The *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC) appears as the pricing sub-problem in column generation schemes for vehicle routing problems, as previously discussed in chapter 4. As a result, comprehending it is crucial for developing efficient branch-and-price frameworks. For a literature review on the importance and methodologies for solving the pricing problem, we refer the reader to section 4.3.

While the ESPPRC can be studied independently, we are primarily interested in it in this work from the perspective of pricing for the VRP. The ESPPRC requests the shortest path (or route) between a starting and destination vertex on a directed weighted network, subject to additional resource constraints. The standard version of the ESPPRC allows for the definition of multiple resources with different semantics, such as time, capacity, gasoline, and more. The resources are typically monotonically increasing quantities that increase as the route passes through new edges or vertices (Irnich et al., 2005; Irnich, 2007). Resource consumption can be bounded at various levels of scope, such as global scope, vertex scope, or arc scope.

The *Elementary Shortest Path Problem with Capacity Constraints* (ESPPCC) is a subset of the ESPPRC characterized by a single resource: the number of goods. A global-scoped threshold $Q \in \mathbb{R}_+$, the vehicle capacity, limits the number of goods served during a route. The ESPPCC can be used to model the pricing sub-problem in the column generation schemes for the Capacitated Vehicle

Routing Problem (CVRP).

We will concentrate on the ESPPCC pricing sub-problem in this work. However, it is crucial to acknowledge that the more general ESPPRC is required to model more complex scenarios, such as the pricing sub-problem induced by the Vehicle Routing Problem with Time Windows (VRPTW).

The *Capacitated Profitable Tour Problem* (CPTP) is a combinatorial optimization problem that, like the ESPPCC, can effectively model the CVRP pricing sub-problem. The CPTP asks for a minimum cost circuit subject to capacity constraints and can be regarded as a specific case of ESPPCC where the starting and destination vertices coincide.

In section 5.2 we will formally introduce, discuss, and provide an IP formulation for the ESPPCC. Similarly, in section 5.3, we will introduce the CPTP and provide an IP formulation and extra valid inequalities. We will study the basic version of these two problems, which means that we will not consider non-robust inequalities that can change the structure of these problems. However, before delving into the discussion of these two problems, we will first devote the following section 5.1 to a review of the few previous contributions regarding the use of branch-and-cut approaches for pricing.

5.1 Literature Review on Branch-and-Cut approaches applied to the Pricing Problem

Label-correcting dynamic programming approaches dominate as the primary approach to solve the (E)SPPRC in the context of pricing, see Desrochers et al., 1992; Feillet et al., 2004; Righini et al., 2004; Righini et al., 2006; Boland et al., 2006; Righini et al., 2008; Pugliese et al., 2010; Baldacci et al., 2011; Lozano et al., 2013; Lozano et al., 2016; Sadykov et al., 2021a. For a more complete discussion, see section 4.3.

To our knowledge, branch-and-cut approaches for tackling the (E)SPPRC have received little attention. The few branch-and-cut contributions that we are aware of are: (i) Jepsen et al. (2008b) for the ESPPCC, (ii) Jepsen (2011) and Jepsen et al. (2014) to the CPTP, (iii) Taccari (2016) and Drexl et al. (2014) for

the related non-resource constrained ESPP and (iv) Horváth et al. (2016) for the non-elementary RCSP.

To the best of our knowledge, no branch-and-cut contributions exist for the general ESPPRC version in the literature. This situation is most likely due to the inherent challenges of modeling resource constraints with non-global bounds, such as time window constraints (Jepsen et al., 2008b). First, non-global resource bounds necessitate the usage of directed network-based IP models, which are twice as large as an undirected equivalent solution. Second, non-globally bounded resource constraints can only be modeled as big-M constraints, which are known to be inherently computationally unstable (Jepsen et al., 2008b).

Jepsen et al. (2008b) provide an IP mathematical model for the ESPPCC over an undirected network, along with additional valid inequalities. In their work, they empirically evaluate the effectiveness of their branch-and-cut framework in the context of pricing for the CVRP.

Jepsen et al. (2014) investigates the CPTP problem in the context of pricing for the CVRP. Their work stems from the initial efforts of Jepsen et al. (2008b). Jepsen et al. provide a first tutorial/survey/framework and foundational theory for further development of the CPTP as a standalone problem or in the context of pricing. In addition to discussing valid inequalities and their efficient separation, they propose a model and a branch-and-cut algorithm for solving the CPTP.

Jepsen et al. (2014) conduct a computational study to assess the competitiveness of their approach and the usefulness of the employed cutting planes. Their findings showed that the CPTP-based branch-and-cut framework was not competitive in solving the CVRP's PP. Overall, the dynamic programming algorithm appeared to perform vastly better. Nonetheless, their findings revealed that the branch-and-cut algorithm solved some larger problem instances in which the labeling algorithm failed to provide a solution. More specifically, their results showed that:

1. The branch-and-cut (BAC) algorithm appeared to solve some standalone CPTP instances characterized by 800 customers. Whereas the labeling algorithm ran out of available computation time when solving instances characterized by more than 200 customers.

2. When the weights of the associated ESPPCC/CPTP network were highly negative, the branch-and-cut algorithm outperformed the labeling algorithm significantly.
3. When the optimal objective value of the problem was very close to the zero threshold, the branch-and-cut framework performed worse.

Item 1 relates to standalone CPTP instances and may thus be irrelevant in the pricing context. With regard to pricing, item 2 is of little interest because highly negative weights networks are usually solved through heuristics anyway. Item 3 is probably due to the fact that labeling algorithms are usually optimized to cut off extensive portions of the solution space when the optimal value resides very close to the zero threshold.

Despite their contribution needs to be revisited in light of recent advances on both sides, their findings demonstrated that a branch-and-cut approach could supplement the dynamic programming algorithm for pricing.

5.2 The Elementary Shortest Path Problem with Capacity Constraints

This section will go over the *Elementary Shortest Path Problem with Capacity Constraints* (ESPPCC) in the context of pricing for the CVRP. We've already seen the general ESPPRC case of this problem in sections 4.1 and 4.3 but we've never provided a formal description of the mathematical model. In this section, we consider the basic version of the ESPPCC, not considering non-robust inequalities at the RMP level. When non-robust inequalities are used, ESPPCC is insufficient for modeling the pricing problem. To ensure the correctness of the column generation approach, a slight variation of the ESPPCC with additional constraints is required.

The ESPPCC asks for the determination of the shortest path on a weighted network between two vertices s,t subject to a single global resource restriction characterized by the number of goods available for serving $Q \in \mathbb{R}_+$. The elementarity condition in the ESPPCC problem requires that no optimal path pass

through the same vertices two or more times. In general, the ESPPCC's underlying network may contain negative cost cycles. The presence of negative cost cycles makes solving such a problem NP-hard (Dror, 1994). A dynamic programming algorithm to tackle the elementary version of the problem was proposed in Feillet et al. (2004). In the context of pricing, however, the problem has traditionally been resolved through an SPPRC. Unlike its elementarity version, the SPPRC can be solved in pseudo-polynomial time using a much faster dynamic programming algorithm, see Desrochers et al. (1992). Unfortunately, relaxing the elementarity condition slows down column generation and worsens dual bounds, as discussed previously in section 4.3.

If the weighted network of the ESPPCC contains no negative-cost cycles, we can safely ignore the elementarity restriction without affecting correctness (Beasley et al., 1989). A sufficient but not-necessary condition is when the reduced costs are all positive: $\bar{c}_{ij} \geq 0 \quad \forall i, j \in V_0 \cup \{s,t\}$. In this case, an optimal solution to the associated non-elementary SPPRC is an optimal solution to the original elementary version of the problem (Beasley et al., 1989). The associated SPPRC can then be solved trivially in pseudo-polynomial time. Initial contributions for tackling the ESPPCC exploited such property. The resource consumptions were relaxed through a Lagrangian method. They obtained feasible dual solutions over strictly positive weighted networks using a branch-and-bound scheme and an off-the-shelf standard shortest path algorithm, such as Dijkstra (Sniedovich, 2006). To name a few, see the contributions of Beasley et al. (1989), Dumitrescu et al. (2003), Carlyle et al. (2008), and Muhandiramge et al. (2009).

However, as noted in Righini et al. (2004), Lagrangian relaxation is only effective when the dualized variables are positive for a significant portion of the Lagrange multipliers search space, limiting the effectiveness of these approaches in the context of pricing.

5.2.1 Integer Programming Formulation

While Jepsen et al. (2008b) provides an **undirected** symmetric network-based formulation for the ESPPCC, in our presentation, we chose to re-adjust the more-

general **directed** symmetric network-based ESPPRC formulation provided in the works of Beasley et al. (1989), Toth et al. (2002), and Toth et al. (2014). As a result, we must readjust/extend the mathematical notation provided for the undirected CVRP in section 2.1.

The ESPPCC is defined over a directed symmetric network $G' = (V', A')$, where $V' = V_0 \cup \{s, t\}$ denotes the set of vertices and A' the set of arcs. The vertex set V' has size $|V'| = N' = N_0 + 2$, where N_0 represents the total customers in the original CVRP problem. The vertices $s = 0$, $t = N' - 1$ denote respectively the source and sink versions of the depot node. The arc set A' can be expressed as:

$$\begin{aligned} A' = &\{(i,j) \mid i,j \in V_0,\ i \neq j\} \\ &\cup \{(s,i) \mid i \in V_0\} \\ &\cup \{(i,t) \mid i \in V_0\}. \end{aligned} \tag{5.1}$$

The arc set A' has size $|A'| = N'(N'-1) + 2(N'-1)$. Following the CVRP, we associate a resource consumption or vertex demand, to each node of the network: $q_i \quad \forall i \in V'$. The resource consumption semantics associated with each customer $i \in V_0$ remains unaltered from the CVRP, while for the source and sink vertices we respectively fix $q_s = q_t = 0$. We formally define $\delta^+(S) = \{(i,j) \in A' \mid i \in S, j \notin S\}$ to denote the out-arcs crossing the set $S \subset V'$. Likewise, we define $\delta^-(S) = \{(i,j) \in A' \mid i \notin S, j \in S\}$ to denote the in-arcs crossing the set $S \subset V'$. For brevity's sake, we also define $\delta^+(i) = \delta^+(\{i\})$, $\delta^-(i) = \delta^-(\{i\})$ to denote respectively the singleton version of the in and out arcs for vertices $i \in V'$. Note that the following conditions hold: $\delta^+(s) = V_0$, $\delta^-(t) = V_0$, $\delta^-(s) = \emptyset$, $\delta^+(t) = \emptyset$. We also define $A'(S) = \{(i,j) \in A' \mid i,j \in S\}$ to denote the set of arcs having both end points in set $S \subseteq V'$.

As it was done in section 4.1, for each directed arc we associate a weight $\bar{c}_{ij} \in \mathbb{R} \quad \forall (i,j) \in A'$. The weight $\bar{c}_{ij}$ represents the reduced cost of an arc $(i,j) \in A'$. Its definition is linked to the dual variables $\pi \in \mathbb{R}^N$ associated to the RMP's

constraints of eqs. (4.2) and (4.3) through the following relationship:

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \frac{\pi_i + \pi_j}{2} & \forall i, j \in V_0 \\ c_{si} - \frac{\pi_0 + \pi_j}{2} & \forall j \in V_0 \\ c_{it} - \frac{\pi_i + \pi_0}{2} & \forall i \in V_0 \\ \infty & \texttt{otherwise} \end{cases} \tag{5.2}$$

Additional robust inequalities (cuts or branching) induce supplementary negative term contributions, through further dual variables, directly to the reduced arc cost $\bar{c}_{ij}$. See previous discussion in section 4.2.

A feasible path solution to the ESPPCC is a sequence $p = (p_0, p_1, \ldots, p_u, p_{u+1})$ with $p_0 = s,\ p_{u+1} = t$ in which $\{p_1, \ldots, p_u\} \subseteq V_0$ customers are visited. Observe that there's no restriction in which customers need to be necessarily covered by the path p. Due to the elementarity condition, the path p can cover a vertex at most once. The path's resource consumption $q_p = q(p) = \sum_{j=0}^{u} q_{p_j}$ satisfies $q(p) \leq Q$. The optimal path $p^\star$ minimizes the overall path's reduced cost $\bar{c}(p) = \sum_{i=0}^{u} \bar{c}_{p_i, p_{i+1}}$ across all possible feasible choices for p.

We are now ready to provide a ESPPCC formal description through an Integer Program (IP). Let $x_{ij} \in \{0,1\} \quad (i,j) \in A'$ be a new set of binary decision variables: $x_{ij} = 1$ if arc $(i,j) \in A'$ is picked by the optimal path. The model reads:

$$\min_x \quad z_{\text{ESPPCC}}(x) = \sum_{(i,j)\in A'} \bar{c}_{ij} x_{ij} \tag{5.3}$$

$$\sum_{i \in V_0} \frac{q_i}{2} \left(\sum_{(j,i)\in\delta^-(i)} x_{ji} + \sum_{(i,j)\in\delta^+(i)} x_{ij} \right) \leq Q \tag{5.4}$$

$$\sum_{(j,i)\in\delta^-(i)} x_{ji} = \sum_{(i,j)\in\delta^+(i)} x_{ij} \qquad \forall i \in V_0 \tag{5.5}$$

$$\sum_{(s,i)\in\delta^+(s)} x_{si} = 1 \tag{5.6}$$

$$\sum_{(i,t)\in\delta^-(t)} x_{it} = 1 \tag{5.7}$$

$$\sum_{(i,j)\in A'(S)} x_{ij} \leq |S| - 1 \qquad \forall S \subseteq V_0 \cup \{s,t\},\ |S| \geq 2 \tag{5.8}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A', \tag{5.9}$$

where (5.3) is the objective function to be minimized, i.e. the overall reduced cost of the path. Constraint (5.4) is the resource consumption upper bound. Constraint (5.5) imposes a flow conservation for all vertices except for the source and the sink. Observe that eq. (5.5) does not impose restriction on whether customer $i \in V_0$ needs to be necessarily covered by the path p. Constraints (5.6) and (5.7) respectively impose an out-flow and in-flow of 1 for the source and sink vertices. Constraints (5.8) are the Subtour Elimination Constraints (SEC) and have a two-fold effect. First, they avoid the formation of spurious tours in unconnected regions of the network, such as the situation depicted in fig. 5.1. Second, they impose the elementarity condition, restricting paths from visiting multiple times the same vertices, such as the situation depicted in fig. 5.2. Constraint (5.9) imposes integrality and forces each arc $(i,j) \in A'$ to be traversed at most once. Despite the dual variables $\pi \in \mathbb{R}^N$ contributions in eq. (5.2) are grouped by two and averaged over each arc, thanks to eqs. (5.5)–(5.7) the path reduced cost still sums to $\bar{c}(p) = \sum_{i=0}^{u} c_{p_i,p_{i+1}} - \sum_{i=0}^{u} \pi_{p_i} = c(p) - \sum_{i=0}^{u} \pi_{p_i}$. Therefore, $\pi_i \in R,\ i \in V_0$ can be interpreted as the gained profit obtained in serving customer $i \in V_0$. Similarly, $\pi_0 \in R$ can be interpreted as an additional constant term biasing the reduced cost of a feasible route p.

Notice that, due to the flow conservation constraint and to the bounds on the binary variables of (5.5) and (5.9), we have that the following equality holds:

$$\sum_{(j,i)\in\delta^-(i)} x_{ji} + \sum_{(i,j)\in\delta^+(i)} x_{ij} = \begin{cases} 2 & \texttt{if } i \in p \\ 0 & \texttt{otherwise} \end{cases} \qquad \forall i \in V_0. \tag{5.10}$$

Regardless of a constant factor of 2, the quantity $\sum_{(j,i)\in\delta^-(i)} x_{ji} + \sum_{(i,j)\in\delta^+(i)} x_{ij}$ can be used to determine the number of times a customer $i \in V_0$ is visited.

The resource consumption constraint of eq. (5.4) is expressed at the vertex level. Equivalently, it is possible to express the same constraint by considering the resource consumption at the arc level:

$$\sum_{(i,j)\in A'} \frac{q_i + q_j}{2} x_{ij} \leq Q. \tag{5.11}$$

Due to the structure of the optimal path $p^\star$ the total resource consumption will not change regardless of whether (5.4) or (5.11) is used.

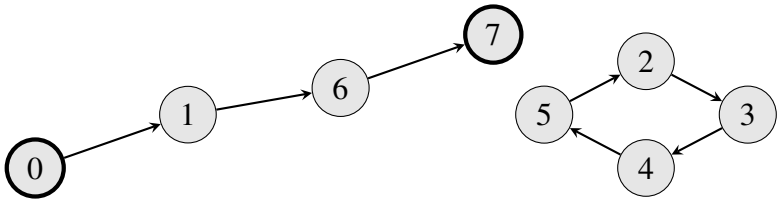


Figure 5.1: An example of a path containing spurious unconnected subtours, over a network with $V' = \{0,\ldots,7\}$, $s = 0$, $t = 7$. The SEC inequalities of eq. (5.8) prohibit the depicted situation from occurring.

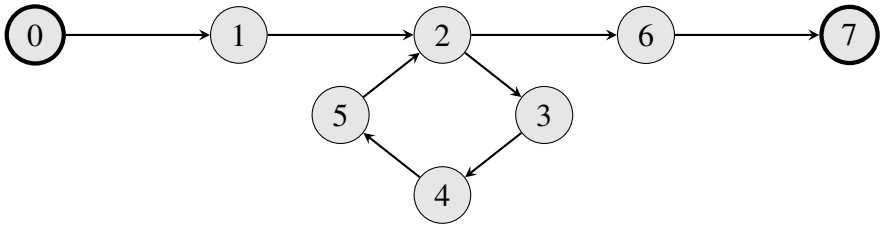


Figure 5.2: An example of a path not satisfying the elementarity constraints, over a network with $V' = \{0,\ldots,7\}$, $s = 0$, $t = 7$. The SEC inequalities of eq. (5.8) prohibit the depicted situation from occurring.

As it is done in Beasley et al. (1989), if one wishes to avoid the formation of spurious unconnected subtours while relaxing the elementarity condition, it can achieve so by substituting eq. (5.8) in favor of a big-M constraint of the form:

$$\sum_{(i,j)\in A'(S)} x_{ij} \leq M \sum_{(i,j)\in \delta^+(S)} x_{ij} \quad \forall S \subseteq V_0 \cup \{s\}, \tag{5.12}$$

where $M \in \mathbb{R}_+$ denotes an arbitrary large positive constant. Equation (5.12) avoids the situation depicted in fig. 5.1 while permitting the situation depicted in fig. 5.2. Modeling the pricing as an SPPCC can significantly reduce column generation running time but at the expense of significantly weaker dual-bound improvements. For more information, refer to the discussion in section 4.3.

Valid inequalities and a branch-and-cut framework for the ESPPCC are discussed in the work of Jepsen et al. (2008b). The CPTP in the context of pricing is the primary focus of this thesis. As a result, we will not spend any more time discussing the ESPPCC.

5.3 The Capacitated Profitable Tour Problem

The *Capacitated Profitable Tour Problem*, abbreviated as CPTP, belongs to the group of the so-called *Travelling Salesman Problems with Profits* (TSPP) (see Feillet et al., 2005) and thus share many similarities with other studied optimization problems such as: (i) the *Orienteering Problem* (OP) (Golden et al., 1987; Laporte et al., 1990) (ii) the *Profitable Tour Problem* (PTP) (Dell'Amico et al., 1995), (iii) the *Prize Collecting Traveling Salesman Problem* (PCTSP) (Balas, 1989; Balas, 1995), (iv) the *Capacitated m-Ring-Star Problem* (CmRSP) (Baldacci et al., 2007). Refer to Letchford et al. (2013) for the Steiner extension of these problems.

The OP, an NP-hard problem (Laporte et al., 1990), asks for a route serving a subset of customers with associated profits. In the OP, the route length is limited by the number of vertices visited, whereas, in CPTP, the route length is constrained by the accumulated demand served. We can conclude that CPTP is NP-hard by applying the same reasoning to prove that the OP is NP-hard in Laporte et al. (1990). The OP is equivalent to an edge-capacitated CPTP with unit demands. For contributions introducing BAC frameworks and valid inequalities for the OP, see Fischetti et al. (1998) and Gendreau et al. (1998). Many of these inequalities (but not all) extend to the CPTP as shown in Jepsen et al. (2014).

The CPTP is a special case of ESPPCC where the source and sink vertices coincide. The CPTP, like the ESPPCC, can model the pricing sub-problem for the CVRP. In contrast to the more general ESPPRC formulation, the CPTP is much easier to describe. However, because of its narrower scope, it is a problem that receives far less attention in the literature. To our knowledge, the few contributions to studying the CPTP in the context of pricing for the CVRP are Jepsen (2011) and Jepsen et al. (2014). not considering non-robust inequalities at the RMP level.

In more detail, the CPTP is a combinatorial optimization delivery problem, in which, given as input a fully connected undirected network where vertices represent the customers, the goal resides in finding a resource-constrained elementary tour (or circuit) starting from a common point called the depot, that minimizes

the overall travel distance while maximizing the profits associated with serving only a subset of the customers. Each customer's profit lowers the tour cost only if the corresponding node is visited. Because of the elementarity constraint, vertices cannot be visited multiple times, implying that the earnable profit per customer is only available once. The profit is encoded at the node level in the traditional CPTP formulation (Jepsen et al., 2014) rather than being included directly in the definition of the reduced cost variable $\bar{c}_{ij} \quad \forall i, j \in V_0 \cup \{0\}$ as it is in the ESPPCC.

The presentation of this section, as well as the implementation discussed in chapter 6, owes a lot to the original efforts of Jepsen et al. (2014).

5.3.1 Integer Programming formulation

We present a CPTP formulation that closely resembles the formulation provided in Jepsen et al. (2014). In terms of mathematical notation, we will use and extend the original mathematical constructs introduced for the CVRP in section 2.1.

Let $p_i \in \mathbb{R}$ denote the profit associated in visiting a vertex $i \in V$. The profit function is directly related to the dual variables $\pi \in \mathbb{R}^N$ associated to the RMP's constraints of eqs. (4.2) and (4.3). Namely, $p_i = \pi_i \quad \forall i \in V$.

Let x_e, $y_i \in \{0,1\}$ be two sets of binary decision variables which respectively model whether an undirected edge $e \in E$ or vertex $i \in V$ is covered by the optimal CPTP route. We can provide a formal mathematical description of the CPTP through an Integer Program (IP) formulation:

$$\min_{x,y} \quad z_{\text{CPTP}}(x,y) = \sum_{i \in E} c_e x_e - \sum_{i \in V} p_i y_i \tag{5.13}$$

$$y_0 = 1 \tag{5.14}$$

$$\sum_{i \in V} q_i y_i \leq Q \tag{5.15}$$

$$\sum_{e \in \delta(i)} x_e = 2y_i \qquad \forall i \in V \tag{5.16}$$

$$\sum_{e \in \delta(S)} x_e \geq 2y_i \qquad \forall i \in S,\ \forall S \subseteq V_0,\ |S| \geq 2 \tag{5.17}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \tag{5.18}$$

$$y_i \in \{0,1\} \qquad \forall i \in V, \quad (5.19)$$

where (5.13) is the objective function to be minimized, i.e. the cost accounting for the total travel distance and the profit in serving a subset of the vertices. Constraint (5.14) ensures that the depot is always covered by the optimal route. Constraint (5.15) is the resource consumption upper bound. Constraint (5.16) has a twofold effect. First, it functions as a form of directed flow conservation. Second, it links the x, y sets of variables together; namely, it ensures that the if a vertex $i \in V$ is covered by an optimal route ($y_i = 1$) then the number of incident edges in that node sums to 2, otherwise to 0. Constraint (5.17) are the so-called Generalized Subtour Elimination Constraints (GSECs), which avoid the formation of spurious tours in unconnected regions of the network. Because there are an exponential number of GSECs, these constraints cannot be inserted statically in a MIP solver and must instead be separated exactly, at least for integral solutions. Constraints (5.18) and (5.19) respectively impose bounds and integrality conditions on the number of traversal for edges $e \in E$ and vertices $i \in V$. The IP model consists of $\frac{N^2+N}{2}$ number of binary variables and an exponential number of constraints. If we ignore the GSECs and variable bounds, the IP model owns a total of $N+2$ constraints. Constraints (5.17) can also be rewritten in a different form (Wolsey, 2020):

$$\sum_{e \in E(S)} x_e \leq \sum_{i \in S \setminus \{j\}} y_i \quad \forall j \in S,\ \forall S \subseteq V_0,\ |S| \geq 2. \quad (5.20)$$

Depending on the size of the subset $S \subseteq V_0, S \geq 2$, (5.20) may be sparser than (5.17). Equivalently, the constraint of eq. (5.15) can be defined at the edge level:

$$\sum_{e=\{i,j\} \in E} \frac{q_i + q_j}{2} x_e \leq Q. \quad (5.21)$$

Note that model (5.13)–(5.19) does not permit for single-customer routes. Instead of modifying the formulation to allow for this edge case, it may be simpler to leave the model unchanged. By employing a brute-force algorithm in $\Theta(N)$ time, we scan for improving single-customer solutions. After the resolution of the IP model, we check for improving single-customer solutions and, if any exist, we update the incumbent as appropriate.

5.3.2 Additional Valid Inequalities

We present additional valid inequalities for the CPTP problem in this section. While these additional inequalities are not strictly required to ensure the correctness of the solution approach, they can massively strengthen the linear relaxation and thus speed up the resolution process when embedded within an efficient branch-and-cut framework. However, devising efficient separation algorithms is critical to make such inequalities efficacious.

The inequalities presented here closely follow the presentation provided in Jepsen et al. (2014). For a complete list of CPTP valid inequalities compared to what we could give, see Jepsen et al. (2014). We will focus primarily on two valid inequalities in this thesis: the *Rounded Capacity Constraints* (RCC) discussed in section 5.3.2.1 and the *Generalized Large Multistars* (GLM) discussed in section 5.3.2.2. We have focused solely on these two sets of inequalities for two reasons. First, Jepsen et al. (2014) found that these two inequalities are the most frequently separated, implying that including them would likely speed up the resolution process. Second, they can be reasonably separated utilizing the same separation procedure employed for the GSEC inequalities. See implementation details later in section 6.4.

5.3.2.1 Rounded Capacity Constraints (RCC)

The *Rounded Capacity Constraints*, or RCC for short, were introduced for the CVRP in Laporte et al. (1983) and were previously introduced in section 2.2.

Recall that $q(S) = \sum_{i \in S} q_i$ represents the total demand in serving all vertices in set $S \subseteq V$. The RCC constraints act as a capacity constraint by requiring that any customer set $S \subseteq V_0$ be crossed by a number of edges that is not less than the required trucks amount to serve all customers in $S \subseteq V_0$. Jepsen et al. (2014) extend the RCC inequalities to the CPTP:

$$\sum_{e \in \delta(S)} x_e \geq 2 \left\lceil \frac{\sum_{i \in S} q_i y_i}{Q} \right\rceil \quad \forall S \subseteq V_0,\ |S| \geq 1. \tag{5.22}$$

Equation (5.22) is very similar to the CVRP's RCC of eq. (2.8), except that the right-hand side is composed of model's variables instead of constants. Notice

that such approach is correct since $\sum_{e\in\delta(S)} x_e \in \mathbb{Z}_+$. Unfortunately, because the ceil function is a non-linear operation, such a constraint cannot be reported to most MIP solvers.

Fortunately, Baldacci et al. (2007) provides a way to combat this undesirable aspect at the cost of obtaining a looser bound of the RCC. Namely, given $\alpha, \beta, \gamma \in Z_+$, with $\alpha > \gamma$ and $mod(\alpha, \gamma) \neq 0$, the following result holds:

$$\left\lceil \frac{\alpha - \beta}{\gamma} \right\rceil \geq \left\lceil \frac{\alpha}{\gamma} \right\rceil - \frac{\beta}{mod(\alpha, \gamma)}. \tag{5.23}$$

Jepsen et al. (2014) claim that eq. (5.23) holds even when $\alpha > \gamma$ is not satisfied (unfortunately the authors don't provide further explanations on why this is the case).

By picking $\alpha = q(S) = \sum_{i\in S} q_i$, $\beta = \sum_{i\in S} q_i(1-y_i)$, $\gamma = Q$, we obtain the following linear, but weaker, RCC:

$$\sum_{e\in\delta(S)} x_e \geq 2\left(\left\lceil \frac{q(S)}{Q} \right\rceil - \frac{\sum_{i\in S} q_i(1-y_i)}{Q_R(S)} \right) \quad \forall S \subseteq V_0,\ |S| \geq 1, \tag{5.24}$$

where $Q_R(S) = mod(q(S), Q)$ is the remainder capacity associated to an arbitrary set $S \subseteq V$. Notice, that eq. (5.23) guarantees correctness of the eq. (5.24) only if $q_i \in Z_+ \quad \forall i \in V_0$ holds. Fortunately, this is true for the vast majority of the routing problems considered in the literature.

As Jepsen et al. (2014) points out, no exact separation procedure is currently known for the RCC inequalities, for this reason, they suggest using the characterizing sets identified by the separation of the multistar inequalities, such as the GLM separation procedure. See the next section 5.3.2.2.

5.3.2.2 Generalized Large Multistar (GLM)

The *Generalized Large Multistar* inequalities, or GLMs for short, were first proposed in Gouveia (1995) for the CVRP. The GLMs were further generalized to the so-called *Knapsack Large Multistar* (KLM) inequalities in Letchford et al. (2002). Letchford et al. (2006) investigates the effectiveness of the multistar family of cuts when applied to vehicle routing problems. The multistar cuts are

a collection of inequalities related to the intersection of the 0–1 knapsack and the CPTP polytopes.

Consider the capacity inequality:

$$\sum_{e \in \delta(S)} x_e \geq \frac{2}{Q} \sum_{i \in S} d_i y_i \quad \forall S \subseteq V_0,\ |S| \geq 2, \tag{5.25}$$

which is a weaker version of the RCC in eq. (5.22), where the ceiling is not applied. Equation (5.25) can be improved to obtain the GLM inequalities by noting that nodes $j \notin S$ are also visited when the crossing edges $\delta(S)$ are used.

The GLM inequalities can be expressed as (Jepsen et al., 2014):

$$\sum_{e \in \delta(S)} x_e \geq \frac{2}{Q} \left(\sum_{i \in S} q_i y_i + \sum_{\substack{e=\{i,j\} \in \delta(S) \\ i \in S,\ j \notin S}} q_j x_e \right) \quad \forall S \subseteq V_0,\ |S| \geq 2. \tag{5.26}$$

The general idea behind multistar inequalities is the following. The vehicle must have sufficient capacity to serve all of the customers covered by S and all other customers outside S who are connected to nodes in S through a crossing edge $e = \{i, j\},\ i \in S,\ j \notin S$. The "capacity" in GLM is measured in terms of the number of vehicles, which is found by dividing by Q the corresponding demand. GLM inequalities are efficiently separable with an exact polynomial-time procedure by finding minimum (u, v)-cuts $\forall u \in V_0,\ v = 0$ in a directed flow network with capacities that depend on the vertex u (Letchford et al., 2006; Jepsen et al., 2014). A simpler approach is to use a heuristic approach by minimizing only the term $\sum_{e \in \delta(S)} x_e$ as to stress the violation. This can be achieved by computing (u, v)-cuts $\forall u \in V_0,\ v = 0$ in a directed flow network where capacities are given exclusively from the fractional solution $x^\star_{\{i,j\}} \quad \forall i, j \in V$. See implementation details later in section 6.4.

CHAPTER 6

Implementation

This chapter introduces the proposed branch-and-cut algorithm (BAC). The BAC algorithm was implemented using the recent *CPLEX 22.1 C callable library*. The proposed branch-and-cut uses the CPTP formulation as the foundational static IP model. Refer to the previous discussion on the CPTP in section 5.3 for more information. By appropriately defining the profit function associated with each vertex $p_i \quad \forall i \in V$, our framework can be utilized as a pricer inside a column generation approach for the standard CVRP. The BAC's source code was developed in *C*. The source code is freely available under a permissive `MIT license` at the following *Github* repository: `https://github.com/dparo/master-thesis`.

IBM ILOG CPLEX Optimizer[1], CPLEX for short, is a commercial optimization software package for solving problems expressed as either: linear programs, mixed-integer programs, quadratic programs, or quadratically constrained programs. For an introduction to CPLEX refer to appendix A.

Our branch-and-cut implementation heavily relies on the CPLEX optimizer to solve the associated CPTP MIP model. The cutting planes separation was implemented using the *CPLEX generic callback functions*, which has the advantage of not disabling the *Dynamic Search algorithm*. The *Dynamic Search* is the CPLEX's internal advanced proprietary branch-and-cut algorithm, as discussed

[1] IBM ILOG CPLEX Optimizer: `https://www.ibm.com/analytics/cplex-optimizer`

in appendix A.

Using a modern MIP solver, such as CPLEX, may provide at least two advantages over tailored pricer algorithms, such as the labeling algorithm (Desrochers et al., 1992; Feillet et al., 2004): (i) parallelization and efficient use of the machine's multiple cores basically for free and (ii) take advantage of all the engineering effort that went into creating an efficient MIP solver (preprocessing, diving, fixing, primal heuristics, and more). However, there are also drawbacks to such an approach. Complicated relaxations or constraints, such as the ng-sets (Baldacci et al., 2011), may be impractical to implement efficiently within a BAC algorithm. On top of that, BAC-based approaches tend to require high amounts of memory for harder problems, leading to tremendous performance losses when the MIP optimizer or the host operating system decide to swap memory to disk.

The chapter's outline is quickly summarized. The implemented static IP model is presented in section 6.1. Section 6.2 describes the implemented primal heuristics for *warm starting* the MIP optimizer. Section 6.3 discusses the implemented branching scheme. Finally, section 6.4 describes the cutting plane strategies employed, as well as a discussion of the separation techniques used.

6.1 Full static model

Our static IP model is based on a minor modification of the CPTP formulation (5.13)–(5.19) by disregarding the GSECs inequalities (5.17). Upon violation, the GSECs will be separated lazily through an exact procedure to ensure the approach's correctness.

In more detail, we've implemented the following static Integer Program (IP) model:

$$\min_{x,y} \quad z_{\text{CPTP}}(x,y) = \sum_{i \in E} c_e x_e - \sum_{i \in V} p_i y_i \tag{6.1}$$

$$y_0 = 1 \tag{6.2}$$

$$B \leq \sum_{i \in V} q_i y_i \leq Q \tag{6.3}$$

$$\sum_{e \in \delta(i)} x_e = 2y_i \qquad \forall i \in V \tag{6.4}$$

$$x_e \in \{0,1\} \qquad \forall e \in E \qquad (6.5)$$

$$y_i \in \{0,1\} \qquad \forall i \in V, \qquad (6.6)$$

where $B \in R_+$ is an additional lower bound on the resource consumption that may slightly improve the linear relaxation. The value of B is computed as $B = q_u + q_v$, where $u, v \in V_0,\ u \neq v$ represent the least two demanding customers in the network: $q_u \leq q_v \leq q_i \quad \forall i \in V_0, i \neq u, v$.

6.1.1 Upper cutoff value

In the context of CVRP pricing, we're only interested in routes achieving a strictly negative cost function. A MIP optimizer can use the *upper cutoff value* to reduce the number of evaluated branch-and-bound nodes. The upper cutoff value acts as a direct constraint on the objective function:

$$z_{\text{CPTP}}(x,y) < 0 - \varepsilon_{\text{ct}}, \qquad (6.7)$$

where $\varepsilon_{\text{ct}} \in \mathbb{R}_+$ denotes the upper cutoff tolerance. Most MIP solvers, including CPLEX, has internal support for specifying cutoff values. However, fixing the cutoff value is not the same as specifying an explicit constraint in the static model. Only when the branch-and-bound procedure is invoked, do cutoff values contribute. An explicit constraint, on the other hand, contributes even at the linear relaxation level. The upper cutoff tolerance ε_{ct} biases the threshold at which a produced route can be considered a valid reduced cost column. A non-zero value for ε_{ct} avoids numerical stability problems caused by the usage of floating-point arithmetic employed in BPC implementations.

To ensure correctness, the value of ε_{ct} must match the value used by the BPC algorithm. In our case, we chose $\varepsilon_{\text{ct}} = 10^{-6}$ because it is the value used by the modern BPC framework described in Sadykov et al. (2021b).

6.2 Warm Starting

Warm starting is a technique that involves feeding a MIP optimizer with (good) feasible solutions before beginning the resolution process. Having a set of (good)

initial feasible solutions can significantly reduce the primal-dual bound gap and, as a result, the amount of time required to achieve optimality. Warm starting can be applied to a wide range of IP problem domains, and it is usually supported by the market's leading MIP optimizers, such as CPLEX.

Primal heuristics offer a quick and practical approach to warm starting. A CPTP problem is very similar to a TSP problem. As a result, TSP's heuristics are fine-tunable to work successfully even with the CPTP. Many contributions were dedicated to researching good heuristics for the TSP, see Rosenkrantz et al., 1977; Johnson et al., 1997; Laporte, 1992; Johnson et al., 2007; Hoffman et al., 2013.

The warm starting procedure that we've implemented is explained in the remainder of this section. The procedure is divided into two stages: a constructive *insertion heuristic* stage (see section 6.2.1) followed by a *2-OPT refinement* stage (see section 6.2.2),

6.2.1 Insertion heuristic

Rosenkrantz et al., 1977 do an excellent job of describing the TSP's insertion heuristic algorithm and various facets in which it can be implemented. In $\Theta(N^2)$ time, the insertion heuristic algorithm can generate a new feasible primal solution. In this section, we modify the TSP insertion heuristic to make it work with the CPTP.

Start by selecting two distinct nodes $u, v \in V,\ q_u + q_v \leq Q$ to form an initial route back-to-back $p = (u, v)$.

The *insertion heuristic* algorithm is an iterative approach in which an almost-feasible CPTP tour is available at each iteration. If the CPTP admits feasible solutions, the final available tour will undoubtedly be feasible when all iterations of the insertion heuristic are exhausted. Let's define $q(p) \leq Q$ as the total demand served by the partial route in this iteration. Choose a vertex $a \in p$ and a vertex $h \notin p$ for each iteration. Let's define b as the successor of node a in the current tour. The goal is to insert h in the tour by deleting edge (a, b) and inserting edges $(a, h),\ (h, b)$. Only perform the insertion operation if the h insertion is required to restore feasibility or if its insertion lowers the route cost. Let us

define the extra mileage more formally as follows:

$$\Delta_m(h,a) = \begin{cases} c_{ah} + c_{hb} - c_{ab} - p_h, & \texttt{if } q(p) + q_h \leq Q \\ \infty, & \texttt{otherwise}, \end{cases} \tag{6.8}$$

which represents the route's delta cost in inserting h as the successor of node a. A vertex $h \in V$ is a good candidate for insertion if at least one of the following conditions is met:

1. $\Delta_m(h,a) < 0$, i.e. inserting h improves over the current route.
2. $h = 0$, i.e. h is the depot node. When the insertion candidate coincides with the depot, regardless of whether its insertion is convenient, it must occur sooner or later. Because we have fixed $q_0 = 0$ by definition, $q(p) + q_0 \leq Q$ is always satisfied for the depot node.
3. The number of visited nodes in the current tour is 2 and $q(p) + q_h \leq Q$.

Prior to insertion the partial route has the form $p = (\ldots, a, b, \ldots)$; whereas after the insertion operation completes: $p = (\ldots, a, h, b, \ldots)$. In our implementation we employed the *cheapest insertion scheme*, which means that the pair (h,a) is chosen to minimize the extra mileage $\Delta_m(h,a)$ across all possible choices for $a \in p,\ h \notin p$. When no more h candidate vertices can be found, the algorithm terminates. At the end of the insertion heuristic, p is a valid route that visits the depot node. The depot node may not necessarily be the first element of the tour $p_0 \neq 0$. A simple circular rotation is all that is required to restore the condition $p_0 = 0$. If no valid h candidate can be found while the tour length is 2, we can conclude that the CPTP formulation does not admit any feasible solution before even solving the IP formulation.

In our implementation we chose $u = 0$ and $v \in V_0$ satisfying $q_u + q_v \leq Q$. This approach allows us to generate $O(N)$ acceptable primal (feasible) solutions in $O(N^3)$ time.

6.2.2 2-OPT refinement

Each solution produced from the insertion heuristic can be further optimized using a *2-OPT refinement procedure*. The 2-OPT algorithm is a heuristic local-

search hill-climbing procedure proposed originally for the TSP in Flood (1956) and Croes (1958) independently.

The 2-OPT algorithm works iteratively in $\Omega(N^2)$ number of iterations. Unfortunately, as Chandra et al. (1999) points out, the 2-OPT procedure may take an exponential number of iterations when fed with purposefully artificially constructed instances. Although this is unfortunate, it is also worth noting that in practice, the probabilistic average number of iterations required for 2-OPT is at most polynomial.

Each iteration of the 2-OPT procedure looks for an existing edge crossing and, if found, performs a 2-OPT exchange to undo it. A 2-OPT exchange is a *primal operation*, which means that its use preserves route feasibility. Because 2-OPT exchange does not add nor remove vertices from the current route p, the route's gained profit does not change during the 2-OPT procedure's execution time. As a result, adapting the original TSP's 2-OPT procedure to the CPTP problem becomes trivial.

Let $a,b \in p$ represent two vertices visited along the current route p. Let $a',b' \in p$ denote respectively the successor of a and b in the current route p. A 2-OPT exchange replaces edges (a,a'), (b,b') with (a,b), (a',b') but only if the delta distance, defined as:

$$\Delta(a,b) = c_{ab} + c_{a'b'} - c_{aa'} - c_{bb'}, \tag{6.9}$$

satisfies $\Delta(a,b) < 0$. In this case, a 2-OPT exchange over vertices (a,b) reduces the route cost. Following a 2-OPT exchange, the portion of the route $[a_s,\ldots,b]$ denoted by head a_s and tail b must be reversed.

In our implementation we look for vertices a,b achieving the cheapest exchange, that is, the delta distance $\Delta(a,b)$ is minimized across all possible valid choices of $a,b \in p$. The local search algorithms terminates when $\nexists(a,b) \mid a,b \in p,\ \Delta(a,b) < 0$.

6.3 Branching

We don't implement any specific branching schemes. We rely on the branching schemes already provided by the CPLEX optimizer.

6.4 Cutting planes and Inequalities separation

Although CPLEX already implements the separation of some families of general cutting planes internally (see appendix A), generating additional inequalities, which aren't deducible from the static model, can significantly improve the running time of the resolution process. Cutting plane separation is a problem that involves finding (strong) violated inequalities and embedding them inside a branch-and-cut framework. Cutting planes improve the linear relaxation by "cutting" fractional points from the convex hull of integer solutions. An efficient cutting planes separation strategy is probably the most important and delicate aspect of any branch-and-cut algorithm (Ralphs et al., 2003). When evaluating the inclusion of a set of inequalities, it's critical to consider the trade-off between the separation cost and the improvements in the linear relaxation. Separation of integral inequalities, on the other hand, can be viewed as a procedure for dynamically generating mandatory constraints that would otherwise be impossible to insert statically into a MIP model.

Additional valid inequalities that are not strictly required to ensure the algorithm's correctness but are computationally expensive to separate exactly, may get included using a heuristic separation procedure.

The parametrization settings of the violated inequalities are another critical aspect when implementing efficient cutting planes procedures. The generation and number of cutting planes alone (without considering separation running time), may significantly impact the BAC algorithm's computation time and memory consumption. Sparse/compact inequalities are usually preferred whenever possible to reduce memory usage. When reporting violated inequalities to the MIP optimizer, it is common practice to set a violation tolerance threshold to limit the number of violated inequalities considered. Inequalities that are not "sufficiently violated" aren't generated nor reported to the MIP optimizer. A low violation threshold results in better dual bounds and fewer branch nodes but slows down the convergence in each node. On the other hand, a high violation threshold results in more branch nodes but faster convergence speed in each node (Jepsen et al., 2008b).

Modern MIP optimizers, such as CPLEX, can filter each user-provided cut by

scoring them, regardless of the violation threshold used by the separation procedure. If CPLEX deems these cuts ineffective, branching may occur prematurely regardless of the presence of violated inequalities (tailing off condition). The process of scoring cuts necessitates even more computation time; as a result, user-cuts filtering can be disabled if so desired. However, if an effective tailing condition is required, the user must implement it within the separation procedure of each cut.

In our implementation, all cutting planes reported to CPLEX are filtered. The MIP optimizer itself makes the final decision on their inclusion.

We are majorly interested in separating only the GSECs (5.17), RCC (5.24) and GLM (5.26) families of inequalities in our branch-and-cut algorithm. All these inequalities have one thing in common: they all require the determination of a subset $S \subseteq V_0$. A subset $S \subseteq V_0$ can be determined using a suitable *labeling algorithm*, which partitions the network by assigning labels (integer) to each vertex.

The GSECs are the only constraints in our model that must be obligatorily included by using an exact separation procedure. Their inclusion must be devised at the very least for integral solutions of the IP model of eqs. (6.1)–(6.6). GSECs can be precisely separated by tracing the *major connected components* induced by the model's integral solution.

Tracing the connected components can be used to determine subsets $T \subseteq V \,|\, \sum_{e \in \delta(T)} x_e = 0$, which constitutes a valid labeling of the network. However, the exact separation of fractional solutions is more complicated. Let $x^\star \in \mathbb{R}^{|E|}$ denote the value of a fractional solution of the linear relaxation in eqs. (6.1)–(6.4), the GSECs can be separated (exactly) by solving (u,v)-min-cuts on a flow network where capacities are given from the fractional solutions' value $x^\star$. The (u,v)-min-cuts outputs a subset $T \subseteq V$ induced from (u,v) by minimizing the term $\sum_{e \in \delta(T)} x^\star_e$. The subset $T \subseteq V$ constitutes a valid labeling of the network for each $u, v \in V,\ u \neq v$.

Instead, the exact separation of the RCC and GLM inequalities is a separate concern; for more info see in sections 5.3.2.1 and 5.3.2.2. There is currently no exact separation procedure for the RCC inequalities (Jepsen et al., 2014). On the

other hand, GLM inequalities can be separated by solving $(u, v = 0)$-min-cuts on a flow network with u-dependent capacities (Letchford et al., 2006; Jepsen et al., 2014).

In our implementation, we chose a different approach for the RCC and GLM inequalities than the one proposed in Jepsen et al. (2014). We reuse the same labeling algorithm employed for the exact fractional and integral separation of the GSECs inequalities. Because the same labeling procedures feed the separation of three different families of inequalities, we can limit the programming effort required to create custom separation procedures while substantially reducing the computational impact. The labeling algorithms and the cutting-plane separation procedures were implemented in user-provided callbacks using the *CPLEX generic callback API*.

We will describe the labeling algorithm used for integral and fractional solutions in the following two sections sections 6.4.1 and 6.4.2. However, note that these two labeling algorithms are only optimal for separating the GSECs inequalities, and they behave suboptimally (heuristically) for separating the RCC and GLM inequalities. In the remaining sections, instead, we will be detailing the separation of each implemented inequality.

6.4.1 Labeling from Integral Solutions

Our integral labeling algorithm is devised to be optimal for detecting violated integral GSECs inequalities of eqs. (6.1)–(6.6), but the same procedure is shared to feed a suboptimal integral separation also for the GLM and RCC inequalities, as shown in the block diagram in fig. 6.1.

Let $x^\star \in \{0,1\}^{|E|}$, $y \in \{0,1\}^{|V|}$ denote the current integral solution of eqs. (6.1)–(6.6). If $x^\star, y^\star$ violates a GSEC constraint, it is simple to show that the solution contains at least a spurious unconnected subtour. Subtours do not satisfy the original CPTP model of eqs. (5.13)–(5.19) and should thus be removed. Unconnected subtours can be detected by computing the *major connected components* via a Depth-First Search (DFS) traversal of the network induced from the integral solution's covered edges $\{e \in E \mid x^\star_e = 1\}$.

More formally, let $C = \{-1, 0, \dots, n_c - 1\}$ denote the set of connected components resulting from the integral solution $x^\star \in \{0,1\}^{|E|}$, $y \in \{0,1\}^{|V|}$. A vertex $i \in V$ is said to belong to a singleton connected component, if the connected component contains only i itself, namely $y_i^\star = 0$. We are only interested in modeling **major** connected components, which have at least two vertices. We ignore singleton connected components and use the sentinel value -1 to encode all the vertices $i \in V$ belonging to singletons. As a result, the number $n_c \in \mathbb{Z}$ frankly represents the number of major connected components and satisfies $n_c \geq 1$. Equivalently, n_c represents the number of subtours formed in the current integral solution. Let $cc(i) \in C$ be an array that encodes in which connected component each vertex $i \in V$ belongs. Because the depot node always belongs to a connected component, we fix $cc(0) = 0$ by definition. Instead, for singletons we fix $cc(i) = -1 \quad \forall i \in V_0 \mid y_i^\star = 0$. The connected components can be computed by a simple DFS traversal which pseudocode is provided in algorithm 1.

Consider the sets $T_k = \{i \in V \mid cc(i) = k\}$ with $k = 0, \dots, n_c$ computed from the recently devised labeling algorithm 1. Due to the construction of such sets, by letting $k = 0, \dots, n_c$, it holds that $|T_k| \geq 2$, $\sum_{e \in \delta(T_k)} x_e^\star = 0$ and $\exists i \in T_k \mid y_i^\star = 1$. Thus ensuring the correctness of the entire branch-and-cut procedure.

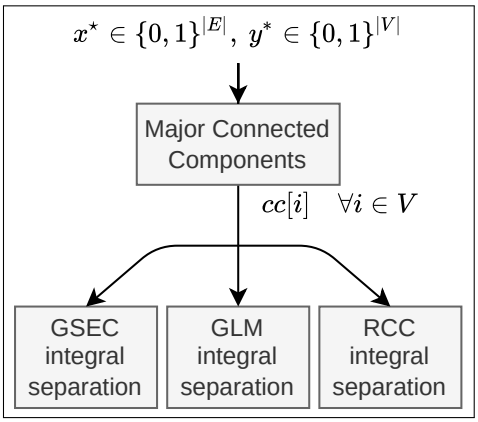


Figure 6.1: A block diagram illustrating the structure of the employed separation for integral solutions along with the shared labeling algorithm.

Algorithm 1: An algorithm for computing the major connected components through a Depth-First Search (DFS) traversal

Data: $x^\star \in \{0,1\}^{|E|}$, $y^\star \in \{0,1\}^{|V|}$: current integral solution of eqs. (6.1)–(6.6)
Result: n_c: number of subtours formed
Result: $cc[i] \in C$: connected component of $\forall i \in V$

```
1  proc cc_dfs(x*, y*)
2      let cc[i] ← −1   ∀i ∈ V;
3      let nc ← 0;
4      for i ← 0 to N do
5          if cc[i] < 0, y*_i = 1 then
               /* Found a non visited subtour                    */
6              cc[i] ← n_c;
7              let u ← −1;
               /* Traverse the subtour                           */
8              do
9                  let v ← −1;
10                 for j ∈ V | e = {u, j} ∈ E, cc[j] < 0, x*_e = 1 do
                       /* The body of this loop will execute only
                          once                                   */
11                     cc[j] = n_c;
12                     v ← j;
13                 end
14                 u ← v
15             while u ≥ 0;
16             n_c ← n_c + 1;
17         end
18     end
       /* Validate some invariants                               */
19     assert n_c ≥ 0;
20     assert c[0] = 0;
21     assert c[i] = −1   ∀i ∈ V_0 | y*_i = 0;
22     assert c[i] ≥ 0   ∀i ∈ V_0 | y*_i = 1;
       /* Done:  terminate                                       */
23     return n_c, cc
24 end
```

6.4.2 Labeling from Fractional Solutions

Our fractional labeling algorithm is devised to be optimal for detecting violated fractional GSECs inequalities of eqs. (6.1)–(6.4), but the same procedure is shared to feed a suboptimal fractional separation also for the GLM and RCC inequalities, as shown in the block diagram in fig. 6.2. This algorithm is inherently more difficult, and computationally expensive than the integral labeling algorithm.

A valid set $T \subseteq V$ can be separated for a fractional $x^\star \in \mathbb{R}^{|E|},\ y^\star \in \mathbb{R}^{|V|}$, by solving a maxflow problem between two arbitrary source and sink vertices $u, v \in V,\ u \neq v$ on a fully connected directed graph. The maxflow problem is represented by a directed symmetric flow network, where the capacities $w_{ij} \in \mathbb{R} \quad \forall i, j \in V$ are derived from the current fractional solution $w_{ij} = x^\star_{\{i,j\}} \quad \forall i, j \in V$.

A solution to the maxflow problem produces a maxflow $f_{\max}(u,v)$ and a bipartition, also known as *"binary coloring"*, of the set V, namely two complementary sets $F_1(u,v),\ F_2(u,v)$ such that $F_1(u,v) \cup F_2(u,v) = V$, $F_1 \cap F_2 = \emptyset$ and $u \in F_1(u,v),\ t \in F_2(u,v)$. The quantity $\delta^+(F_1(u,v))$ constitutes the min-cut induced from solving the maxflow problem over (u,v). It is well understood that solving a maxflow problem assures the following two properties:

1. Arcs $\{(i,j) \mid i \in F_1(u,v),\ j \in F_2(u,v)\}$ are saturated
2. Arcs $\{(j,i) \mid i \in F_1(u,v),\ j \in F_2(u,v)\}$ are drained.

As a result, the following statements hold:

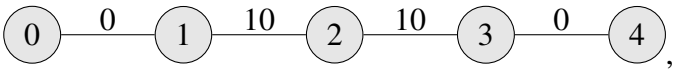
$$\sum_{(i,j)\in\delta^+(F_1(u,v))} f_{ij} = \sum_{(i,j)\in\delta^+(F_1(u,v))} x^\star_{\{i,j\}} = f_{\max}(u,v) \tag{6.10}$$

$$\sum_{(i,j)\in\delta^-(F_1(u,v))} f_{ij} = \sum_{(i,j)\in\delta^+(F_2(u,v))} f_{ij} = 0, \tag{6.11}$$

where f_{ij} denotes the flow in the corresponding arc $(i,j),\ i,j \in V$ of the flow network.

Because we are dealing with a **symmetrical** flow network, solving the maxflow problem between pairs (u,v) and pair (v,u) will yield the same maxflow value, i.e. $f_{\max}(u,v) = f_{\max}(v,u)$. However, the induced bipartitions

are not guaranteed to be symmetric in the general case. In general, we have that $F_1(u,v) \neq F_2(v,u)$ and $F_2(u,v) \neq F_1(v,u)$. To demonstrate this, consider the following simple flow network:



,

produces the same maxflow value when solving for $(0,4)$ and $(4,0)$ pairs and produces $F_1(0,4) = \{0\}$, $F_2(0,4) = \{1,2,3,4\}$, $F_1(4,0) = \{4\}$, $F_2(4,0) = \{0,1,2,3\}$, clearly indicating a nonsymmetric coloring. This behavior is a direct consequence of the fact that flow networks do not guarantee unique min-cuts.

We used the push relabel max flow algorithm first developed in Goldberg, 1997 in this thesis. The push relabel algorithm, takes $O(N^4)$ time to complete and, in practice, is typically faster than commoner approaches like the Ford-Fulkerson algorithm. When combined with an exhaustive enumeration of all possible $u,v \in V,\ u \neq v$ choices, the Goldberg's algorithm can take up to $O(N^6)$ time.

The Gomory-Hu tree, first presented in Gomory et al., 1961, allows us to compute all pairs of (u,v) max flows with only N main maxflow computations. A Gomory-Hu tree, in essence, is a data structure that represents a simpler reduced flow network in which maxflow computations become trivially solvable with a single iteration of the Ford-Fulkerson algorithm in $\Theta(N^2)$. An exhaustive enumeration of all possible (u,v) pairs using the Gomory-Hu tree and the Goldberg's algorithm takes up to $O(N^5)$ time to build the tree and further $\Theta(N^4)$ to query for all the possible $F_1(u,v), F_2(u,v)$ bipartitions.

We used the Gomory-Hu tree in our implementation to thoroughly enumerate all possible (u,v) max flows. For the sake of brevity, we will not include the pseudocode for the maxflow and Gomory-Hu tree algorithms. For their respective implementations, please see the Github repository[2].

We conclude the section with a suggestion for implementing reliable maxflow algorithms. Because of the accumulation of accuracy errors, implementing fast and correct maxflow algorithms that operate directly with floating-point arithmetic can be very problematic. Any trivial implementation can "get stuck" in

[2]Github repository: https://github.com/dparo/master-thesis

massively long loops pushing atomically small amounts of flows. Special care must be taken when developing this kind of algorithm. Unit testing and other solid software engineering practices can aid in identifying problematic implementations.

A cleaner approach is to implement a maxflow algorithm that uses integer arithmetic. Because $x^\star$ is bounded in the $[0,1]$ range, we can multiply the fractional solution value $x^\star$ by a big-constant $M \in \mathbb{R}_+$ (e.g. $M = 10^6$) and truncate its value when defining the flow network's capacities. Truncation may however hurt the BAC algorithm's precision if done recklessly. After that, the value $f_{\max}(u,v)$ must be remapped on the original scale by dividing it by M.

Our maxflow implementation follows the latter approach: it uses integer arithmetic for simplicity and stability.

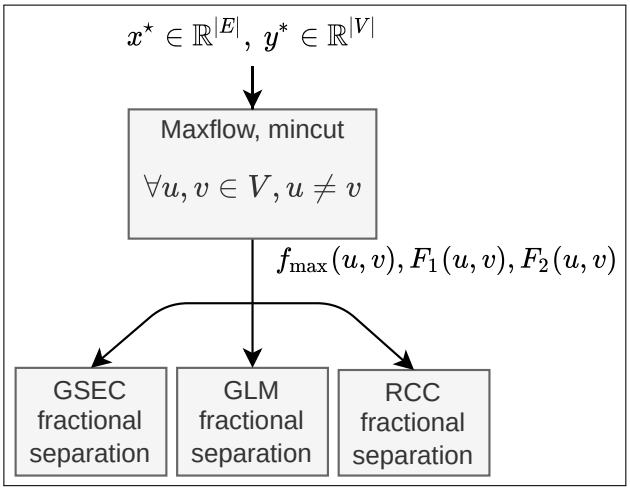


Figure 6.2: A block diagram illustrating the structure of the employed separation for fractional solutions along with the shared labeling algorithm.

6.4.3 GSEC Separation

6.4.3.1 GSEC Integral Separation

The GSEC integral separation procedure is the most important separation procedure in our implementation, since it is mandatory to ensure the correctness of the branch-and-cut algorithm. When an integral solution violates a GSEC inequal-

ity, the latter must be separated exactly and reported to the MIP optimizer as is (without employing violation thresholds).

The GSEC inequalities require for the identification of a subset $S \subseteq V_0,\ |S| \geq 2$. Let $T_k = \{i \in V \mid cc(i) = k\},\ |T_k| \geq 2 \quad \forall k \in \{1,\dots,n_c\}$ represent the labeling outputted from the major connected components as discussed in section 6.4.1. By construction $n_c \geq 1,\ 0 \in T_0$. Then a valid subset $S \subseteq V_0$ can be picked as $S = T_k \quad \forall k \in \{1,\dots,n_c\}$. This means that any major connected component (i.e. subtour) that does not include the depot node, can be used as a valid set $S \subseteq V_0$ for separating GSEC inequalities from integral solutions. If $n_c = 1$ then no $S \subseteq V_0$ can be separated, implying that the current integral solution represents an optimal solution for the entire CPTP formulation (5.13)–(5.19).

Let $x^\star \in \{0,1\}^{|E|},\ y \in \{0,1\}^{|V|}$ denote the current integral solution of eqs. (6.1)–(6.6). Remembering the GSEC inequality definition in eq. (5.17), it is simple to verify that $\sum_{e \in \delta(S)} x^\star_e = 0,\ y_i = 1 \quad \forall i \in S$, implying that eq. (5.17) cannot be satisfied by any $i \in S$.

We can thus separate $\sum_{k=1}^{n_c} |\{i \mid cc(i) = k \quad \forall i \in V\}|$ violated GSEC inequalities in a single integral separation iteration. The violated inequalities can be promptly reported to the MIP optimizer to reject the candidate integral solution. Algorithm 2 contains the complete pseudocode for the GSEC integral separation procedure.

6.4.3.2 GSEC Fractional Separation

While it is not strictly necessary for our implementation, separating GSEC inequalities for fractional solutions can significantly reduce the number of branch nodes.

Let $f_{\max}(u,v),\ F_1(u,v),\ F_2(u,v)$ represent the maxflow value and bipartitions produced by the labeling algorithm described in section 6.4.2. A valid $S \subseteq V_0, |S| \geq 2$ for separating GSEC inequalities from fractional solutions is:

$$S \subseteq V_0 = \begin{cases} F_1(u,v), & \texttt{if } 0 \notin F_1(u,v) \\ F_2(u,v), & \texttt{otherwise} \end{cases}, \qquad |S| \geq 2. \tag{6.12}$$

Algorithm 2: An algorithm for separating GSEC integral inequalities for the CPTP

Result: n_c: number of major connected components
Data: $cc[i] \in C$: connected component $\forall i \in V$, see section 6.4.1
Data: $x^\star \in \{0,1\}^{|E|}$, $y^\star \in \{0,1\}^{|V|}$: current integral solution of eqs. (6.1)–(6.6)

```
1  proc gsec_integral_sep(n_c, cc, x*, y*)
2      const sense ← '≥';
3      const rhs ← 0;
4      for k ← 1 to n_c do
5          let S ← {i | cc[i] = k};
6          assert |S| ≥ 2;
           /* Build the cut                                   */
7          let nnz ← 0, index ← [], value ← [];
8          for e = {i, j} ∈ E | i ∈ S, j ∉ S do
9              index[nnz] ← x_mip_var_idx(e);
10             value[nnz] ← 1;
11             nnz ← nnz + 1;
12         end
13         for i ∈ S do
14             index[nnz] ← y_mip_var_idx(i);
15             value[nnz] ← −2.0;
               /* Reject and report cut to MIP                */
16             mip_add_user_cut(sense, rhs, nnz, index, value);
17         end
18     end
19 end
```

This approach can feed up to $N^2 - N$ distinct sets $S \subseteq V_0$, one for each possible pair $(u,v) \in V^2$, $u \neq v$

Let $x^\star \in \mathbb{R}^{|E|}$, $y \in \mathbb{R}^{|V|}$ denote the current fractional solution of eqs. (6.1)–(6.4). Remembering the GSEC inequality definition in eq. (5.17), it is simple to verify that $\sum_{e \in \delta(S)} x^\star_{ij} = f_{\max}(u,v)$ holds due to the symmetry property of the flow network.

Any $i \in S$ violating $f_{\max}(u,v) \geq 2y_i$ can thus be used to separate a violated GSEC inequality. We can separate $O(N^3)$ GSECs per fractional solution by using this method. A quick empirical evaluation revealed that generating and

reporting all violated GSEC inequalities significantly slowed down the MIP optimizer.

We took a different approach in our implementation by relaxing the conditions under which these cuts are reported. We report only the single most violated GSEC associated with the customer $i \in V_0$ that maximizes the $f_{\max}(u,v) - 2y_i$ violation for each subset $S \subseteq V_0,\ |S| \geq 2$. Furthermore, to limit the number of weak GSEC inequalities that are reported to the MIP optimizer, we define a violation threshold $\varepsilon_{\text{GSEC}}$ and report the associated cut only if:

$$f_{\max}(u,v) \geq 2y_i - \varepsilon_{\text{GSEC}} \tag{6.13}$$

is violated. We chose $\varepsilon_{\text{GSEC}} = 0.01$ in our implementation.

Algorithm 3 contains the complete pseudocode for the GSEC fractional separation procedure.

6.4.4 RCC separation

The RCC inequalities, defined in eq. (5.24), necessitate for the identification of a subset $S \subseteq V_0, |S| \geq 1$. Such inequalities can be rewritten as follows:

$$\sum_{e \in \delta(S)} x_e - \sum_{i \in S} y_i \frac{2q_i}{Q_{\text{R}}(S)} \geq 2\left(\left\lceil \frac{q(S)}{Q} \right\rceil - \frac{q(S)}{Q_{\text{R}}(S)} \right) \quad \forall S \subseteq V_0,\ |S| \geq 1. \tag{6.14}$$

We begin by discussing the separation of RCC inequalities for integral solutions. This procedure follows roughly the same reasoning as the GSEC integral separation discussed in section 6.4.3.1. Specifically, let $T_k = \{i \in V \mid cc(i) = k\},\ |T_k| \geq 2 \quad \forall k \in \{1,\dots,n_c\}$ be the network labeling output by the major connected components as discussed in section 6.4.1. Then a valid subset $S \subseteq V_0,\ |S| \geq 1$ can be picked as $S = T_k \quad \forall k \in \{1,\dots,n_c\}$. We test for inequality violation after selecting the appropriate subset S. Only when the associated RCC inequality is violated is the MIP optimizer notified. Algorithm 4 contains a complete pseudocode.

Following that, we describe the separation of the RCC inequalities for fractional solutions. This separation procedure follows roughly the same reasoning as the GSEC fractional separation discussed in section 6.4.3.2.

Algorithm 3: An algorithm for separating GSEC fractional inequalities for the CPTP

Data: $f_{\max}(u,v), F_1(u,v), F_2(u,v)$: maxflow and bipartitions induced from (u,t)-min-cut, see section 6.4.2
Data: $x^\star \in \mathbb{R}^{|E|}$, $y^\star \in \mathbb{R}^{|V|}$: current fractional solution of eqs. (6.1)–(6.4)

proc *gsec_frac_sep*($f_{\max}(u,v)$, $F_1(u,v)$, $F_2(u,v)$, $x^\star$, $y^\star$)
  const sense $\leftarrow$ '$\geq$';
  const rhs $\leftarrow 0$;
  const $\varepsilon_{\text{GSEC}} \leftarrow 0.01$;
  let S;
  if $0 \notin F_1(u,v)$ **then**
    $S \leftarrow F_1(u,v)$;
  else
    $S \leftarrow F_2(u,v)$;
  end
  `/* Scan for the most violated customer */`
  let $c \leftarrow -1$, $m \leftarrow \infty$;
  for $i \in V \mid i \in S$ **do**
    let $v \leftarrow f_{\max}(u,v) - 2y_i^*$;
    if $(f_{\max}(u,v) < 2y_i^* - \varepsilon_{\text{GSEC}})$ **and** $v < m$ **then**
      $c \leftarrow i$;
      $m \leftarrow v$;
    end
  end
  if $c \geq 0$ **and** $|S| \geq 2$ **then**
    `/* Build the cut */`
    let nnz $\leftarrow 0$, index $\leftarrow$ [], value $\leftarrow$ [];
    for $e = \{i,j\} \in E \mid i \in S,\ j \notin S$ **do**
      index[nnz] $\leftarrow$ x_mip_var_idx(e);
      value[nnz] $\leftarrow 1.0$;
      nnz $\leftarrow$ nnz + 1;
    end
    index[nnz] $\leftarrow$ y_mip_var_idx(c);
    value[nnz] $\leftarrow -2.0$;
    nnz $\leftarrow$ nnz + 1;
    `/* Reject and report cut to MIP */`
    mip_add_user_cut(sense, rhs, nnz, index, value);
  end
end

Let $f_{\max}(u,v)$, $F_1(u,v)$, $F_2(u,v)$ denote the maxflow value and bipartitions produced by the network labeling algorithm described in section 6.4.2. A suitable $S \subseteq V_0, |S| \geq 2$ for separating RCC inequalities from fractional solutions can be chosen as:

$$S \subseteq V_0 = \begin{cases} F_1(u,v), & \texttt{if } 0 \notin F_1(u,v) \\ F_2(u,v), & \texttt{otherwise.} \end{cases} \tag{6.15}$$

After selecting the appropriate subset S we test for inequality violation, but, as with fractional separation, we employ a non-zero violation threshold ε_{RCC}. We set $\varepsilon_{\text{RCC}} = 0.01$ in our implementation. Using this method, we can separate $O(N^2)$ RCCs per fractional solution. Algorithm 5 contains the complete pseudocode.

6.4.5 GLM separation

The GLM inequalities, defined in eq. (5.26), require for the identification of a subset $S \subseteq V_0, |S| \geq 2$. Such inequalities can be rewritten as follows:

$$\sum_{\substack{e=\{i,j\}\in\delta(S) \\ i\in S,\ j\notin S}} x_e \left(1 - 2\frac{q_j}{Q}\right) - \sum_{i\in S} \frac{2q_i}{Q} y_i \geq 0 \quad \forall S \subseteq V_0,\ |S| \geq 2. \tag{6.16}$$

We begin by discussing the separation of the GLM inequalities for integral solutions. This procedure follows roughly the same reasoning as the GSEC integral separation discussed in section 6.4.3.1. Specifically, let $T_k = \{i \in V \mid cc(i) = k\},\ |T_k| \geq 2 \quad \forall k \in \{1,\ldots,n_c\}$ be the network labeling output by the major connected components as discussed in section 6.4.1. Then a valid subset $S \subseteq V_0,\ |S| \geq 2$ can be picked as $S = T_k \quad \forall k \in \{1,\ldots,n_c\}$. We test for inequality violation after selecting the appropriate subset S. Only when the associated GLM inequality is violated is the MIP optimizer notified. Algorithm 6 contains a complete pseudocode.

Following that, we describe the separation of the GLM inequalities for fractional solutions. This separation procedure follows roughly the same reasoning as the GSEC fractional separation discussed in section 6.4.3.2. Let $f_{\max}(u,v)$, $F_1(u,v)$, $F_2(u,v)$ denote the maxflow value and bipartitions

Algorithm 4: An algorithm for separating RCC integral inequalities for the CPTP

Result: n_c: number of major connected components
Data: $cc[i] \in C$: connected component $\forall i \in V$, see section 6.4.1
Data: $x^\star \in \{0,1\}^{|E|}$, $y^\star \in \{0,1\}^{|V|}$: current integral solution of eqs. (6.1)–(6.6)

proc *rcc_integral_sep*(n_c, cc, $x^\star$, $y^\star$)
  const sense $\leftarrow$ '$\geq$';
  for $k \leftarrow 1$ **to** n_c **do**
    const $S \leftarrow \{i \mid cc[i] = k\}$;
    const $Q_S \leftarrow \sum_{i \in S} q_i$;
    const $Q_R \leftarrow \mod(Q_S, Q)$;
    const rhs $\leftarrow 2\left(\left\lceil \frac{Q_S}{Q} \right\rceil - \frac{Q_S}{Q_R}\right)$;
    if $Q_R > 0$ **and** $|S| \geq 1$ **then**
      if $-2 \sum_{i \in S} \frac{q_i}{Q_R} y_i^* < rhs$ **then**
        /* Build the cut */
        let nnz $\leftarrow 0$, index $\leftarrow []$, value $\leftarrow []$;
        for $e = \{i, j\} \in E \mid i \in S,\ j \notin S$ **do**
          index[nnz] $\leftarrow$ x_mip_var_idx(e);
          value[nnz] $\leftarrow 1$;
          nnz $\leftarrow$ nnz + 1;
        end
        for $i \in S$ **do**
          index[nnz] $\leftarrow$ y_mip_var_idx(i);
          value[nnz] $\leftarrow -2q_i/Q_R$;
          nnz $\leftarrow$ nnz + 1;
        end
        /* Reject and report cut to MIP */
        mip_add_user_cut(sense, rhs, nnz, index, value);
      end
    end
  end
end

Algorithm 5: An algorithm for separating RCC fractional inequalities for the CPTP

Data: $f_{\max}(u,v), F_1(u,v), F_2(u,v)$: maxflow and bipartitions induced from (u,t)-min-cut, see section 6.4.2
Data: $x^\star \in \mathbb{R}^{|E|}$, $y^\star \in \mathbb{R}^{|V|}$: current fractional solution of eqs. (6.1)–(6.4)

proc *rcc_frac_sep*($f_{\max}(u,v)$, $F_1(u,v)$, $F_2(u,v)$, $x^\star$, $y^\star$)
  const sense $\leftarrow$ '$\geq$';
  const $\varepsilon_{\text{RCC}} \leftarrow 0.01$;
  let S;
  if $0 \notin F_1(u,v)$ **then**
    $S \leftarrow F_1(u,v)$;
  else
    $S \leftarrow F_2(u,v)$;
  end
  const $Q_S \leftarrow \sum_{i \in S} q_i$;
  const $Q_R \leftarrow \mod(Q_S, Q)$;
  const rhs $\leftarrow 2\left(\left\lceil \frac{Q_S}{Q} \right\rceil - \frac{Q_S}{Q_R}\right)$;
  if $Q_R > 0$ **and** $|S| \geq 1$ **then**
    if $f_{\max}(u,v) - 2\sum_{i \in S} \frac{q_i}{Q_R} y_i^* < rhs$ - ε_{RCC} **then**
      `/* Build the cut */`
      let nnz $\leftarrow 0$, index $\leftarrow$ [], value $\leftarrow$ [];
      for $e = \{i,j\} \in E \mid i \in S,\ j \notin S$ **do**
        index[nnz] $\leftarrow$ x_mip_var_idx(e);
        value[nnz] $\leftarrow 1$;
        nnz $\leftarrow$ nnz + 1;
      end
      for $i \in S$ **do**
        index[nnz] $\leftarrow$ y_mip_var_idx(i);
        value[nnz] $\leftarrow -2q_i/Q_R$;
        nnz $\leftarrow$ nnz + 1;
      end
      `/* Reject and report cut to MIP */`
      mip_add_user_cut(sense, rhs, nnz, index, value);
    end
  end
end

produced by the network labeling algorithm described in section 6.4.2. A suitable $S \subseteq V_0, |S| \geq 2$ for separating GLM inequalities from fractional solution can be chosen as:

$$S \subseteq V_0 = \begin{cases} F_1(u,v), & \texttt{if } 0 \notin F_1(u,v) \\ F_2(u,v), & \texttt{otherwise} \end{cases}, \qquad |S| \geq 2. \tag{6.17}$$

After selecting the appropriate subset S we test for inequality violation, but, as with fractional separation, we employ a non-zero violation threshold ε_{GLM}. We set $\varepsilon_{\text{GLM}} = 0.01$ in our implementation. Using this method, we can separate $O(N^2)$ GLMs per fractional solution. Algorithm 7 contains the complete pseudocode.

Algorithm 6: An algorithm for separating GLM integral inequalities for the CPTP

Result: n_c: number of major connected components
Data: $cc[i] \in C$: connected component $\forall i \in V$, see section 6.4.1
Data: $x^\star \in \{0,1\}^{|E|}$, $y^\star \in \{0,1\}^{|V|}$: current integral solution of eqs. (6.1)–(6.6)

proc *glm_integral_sep(n_c, cc, $x^\star$, $y^\star$)*
  const sense $\leftarrow$ '$\geq$';
  const rhs $\leftarrow 0$;
  for $k \leftarrow 1$ **to** n_c **do**
    let $S \leftarrow \{i \mid cc[i] = k\}$;
    assert $|S| \geq 2$;
    `/* Build the cut */`
    let lhs $\leftarrow 0$;
    let nnz $\leftarrow 0$, index $\leftarrow$ [], value $\leftarrow$ [];
    for $e = \{i, j\} \in E \mid i \in S,\ j \notin S$ **do**
      index[nnz] $\leftarrow$ x_mip_var_idx(e);
      value[nnz] $\leftarrow 1 - 2q_j/Q$;
      lhs $\leftarrow$ lhs + $x^\star_{ij} \cdot (1 - 2q_j/Q)$;
      nnz $\leftarrow$ nnz + 1;
    end
    for $i \in S$ **do**
      index[nnz] $\leftarrow$ y_mip_var_idx(i);
      value[nnz] $\leftarrow -2q_i/Q$;
      lhs $\leftarrow$ lhs + $y^*_i \cdot (2q_i/Q)$;
      nnz $\leftarrow$ nnz + 1;
    end
    if not $lhs \geq rhs - \varepsilon_{\text{GLM}}$ **then**
      `/* Reject and report cut to MIP */`
      mip_add_user_cut(sense, rhs, nnz, index, value);
    end
  end
end

Algorithm 7: An algorithm for separating GLM fractional inequalities for the CPTP

Data: $f_{\max}(u,v), F_1(u,v), F_2(u,v)$: maxflow and bipartitions induced from an arbitrary $s \neq t, s \in V, t \in V$ pair
Data: $x^\star \in \mathbb{R}^{|E|}$, $y^\star \in \mathbb{R}^{|V|}$: current fractional solution of eqs. (6.1)–(6.4)

proc *glm_frac_sep*($f_{\max}(u,v)$, $F_1(u,v)$, $F_2(u,v)$, $x^\star$, $y^\star$)
  const sense $\leftarrow$ '$\geq$';
  const rhs $\leftarrow 0$;
  const $\varepsilon_{\text{GLM}} \leftarrow 0.01$;
  let S;
  if $0 \notin F_1(u,v)$ **then**
    $S \leftarrow F_1(u,v)$;
  else
    $S \leftarrow F_2(u,v)$;
  end
  if $|S| \geq 2$ **then**
    /* Build the cut */
    let lhs $\leftarrow 0$;
    let nnz $\leftarrow 0$, index $\leftarrow$ [], value $\leftarrow$ [];
    for $e = \{i,j\} \in E \mid i \in S,\ j \notin S$ **do**
      index[nnz] $\leftarrow$ x_mip_var_idx(e);
      value[nnz] $\leftarrow 1 - 2q_j/Q$;
      lhs $\leftarrow$ lhs + $x^\star_{ij} \cdot (1 - 2q_j/Q)$;
      nnz $\leftarrow$ nnz + 1;
    end
    for $i \in S$ **do**
      index[nnz] $\leftarrow$ y_mip_var_idx(i);
      value[nnz] $\leftarrow -2q_i/Q$;
      lhs $\leftarrow$ lhs + $y^*_i \cdot (2q_i/Q)$;
      nnz $\leftarrow$ nnz + 1;
    end
    if not $lhs \geq rhs - \varepsilon_{\text{GLM}}$ **then**
      /* Reject and report cut to MIP */
      mip_add_user_cut(sense, rhs, nnz, index, value);
    end
  end
end

CHAPTER 7

Results

This chapter will present the empirical results of the branch-and-cut algorithm, which implementation was discussed in chapter 6. We will evaluate its performance as a pricer for the CVRP by comparing it with the state-of-the-art labeling algorithm based on dynamic programming discussed in the works of Pessoa et al. (2020a) and Sadykov et al. (2021a).

7.1 CVRP Benchmark Instances

Several CVRP benchmark instances are used to measure the competitiveness of our branch-an-cut pricer in an accurate manner. The CVRPLIB website[1] is an online database for the vehicle routing problem that includes several downloadable test instances for free, among other things. It is a valuable resource for all practitioners interested in the VRP. It includes interactive plots of various routing problem instances and the optimal (or best known) solution discovered by the best scholars over time. Each CVRP test instance is stored in a file with the following filename template:

`<F>-n<N>-k<K>.vrp`

where `.vrp` is the file extension, `<N>` is the number of vertices in the instance, and `<K>` is the number of (maximum or exact) available vehicles. Finally, `<F>`

[1] CVRPLIB website: http://vrp.galgos.inf.puc-rio.br/index.php/en/

represents the set instance family. `<F>` is a single letter that uniquely identifies the instance set and the authors who published such a set. `P-n40-k5.vrp`, for example, denotes a test instance made up of 40 nodes (39 customers) and 5 vehicles. The `P` in the name refers to the instance set family, which was published in Augerat et al. (1995).

Each CVRP instance file's contents adhere to the `TSPLIB95` file format (Reinelt, 1995). Distances between pairs of nodes are computed using the 2D Euclidean distance function rounded to the nearest integer, as became standard for the TSP in Reinelt (1991). Rounding is performed to stabilize the optimal values, allowing for an accurate comparison of different contributions. However, rounding introduces issues when comparing heuristic contributions; for more information, see Uchoa et al. (2017).

We used some of the most historic and well-known CVRP instances to evaluate the performance of our pricer: set `A`, `B`, `P` (Augerat et al., 1995), set `E` (Dantzig et al., 1959; Christofides et al., 1969; Gaskell, 1967; Gillett et al., 1974), and finally set `F` (Fisher, 1994). If the reader is wondering how these test sets were generated in the first place, they can consult the CVRPLIB website or the work of Uchoa et al. (2017).

The employed test instances are summarized in tables 7.1–7.5.

Instance	K	Q	**Optimal Value**
E-n51-k5	5	160	521
E-n76-k7	7	220	682
E-n76-k8	8	180	735
E-n76-k10	10	140	830
E-n76-k14	14	100	1021
E-n101-k8	8	200	815
E-n101-k14	14	112	1067

Table 7.1: Table listing the employed instances of the set `E` (in total 7) for the empirical evaluation. The set `E` was proposed in Dantzig et al. (1959), Christofides et al. (1969), Gaskell (1967), and Gillett et al. (1974). The node locations were chosen at random from a uniform distribution (Uchoa et al., 2017).

Instance	K	Q	Optimal Value
F-n45-k4	4	2010	724
F-n72-k4	4	30000	237
F-n135-k7	7	2210	1162

Table 7.2: Table listing the employed instances of the set F (in total 3) for the empirical evaluation. The set F was proposed in Fisher (1994). The instances come from an actual distribution problem involving grocery deliveries in Ontario (Uchoa et al., 2017).

7.1.1 Inflation of the CVRP Test Instances

We created new artificial instances based on the presented sets in tables 7.1–7.5. The idea is to generate new CVRP instances characterized by longer routes so that we can evaluate the behavior of the proposed branch-and-cut pricer and labeling algorithm as the routes they need to produce grow longer (see previous discussions in sections 1.2 and 4.3). The new artificial instances were created as follows. Let $Q \in R_+$, $K \in Z_+$ respectively denote the vehicle capacity and the number of trucks of the unmodified CVRP instance. Define a scale factor $s \in R_+$. For each unmodified CVRP instance, we generate a new artificial instance characterized by the vehicle capacity $Q' \in R_+$ and by the number of vehicles $K' \in Z_+$. We've inflated the total number of available CVRP instances by using the following relationship:

$$Q' = Q \times s, \quad K' = \left\lceil \frac{K}{s} \right\rceil$$

Each unmodified CVRP instance is inflated through multiple values for the scale factor $s \in R_+$. As $s \in R_+$ grows, the inflated CVRP instances will look more and more like a conventional TSP. If $K' = 1$, the CVRP decades to a TSP. As a result, the BAP frameworks may become a suboptimal approach for dealing with such a problem.

7.2 *BaPCod*

BaPCod (Sadykov et al., 2021b) is a software package developed in France at the Bordeaux University and Bordeaux Research Center that embeds a sophisticated

Instance	K	Q	Optimal Value
A-n37-k5	5	100	669
A-n37-k6	6	100	949
A-n38-k5	5	100	730
A-n39-k5	5	100	822
A-n39-k6	6	100	831
A-n44-k6	6	100	937
A-n45-k6	6	100	944
A-n45-k7	7	100	1146
A-n46-k7	7	100	914
A-n48-k7	7	100	1073
A-n53-k7	7	100	1010
A-n54-k7	7	100	1167
A-n55-k9	9	100	1073
A-n60-k9	9	100	1354
A-n61-k9	9	100	1034
A-n62-k8	8	100	1288
A-n63-k9	9	100	1616
A-n64-k9	9	100	1401
A-n65-k9	9	100	1174
A-n69-k9	9	100	1159
A-n80-k10	10	100	1763

Table 7.3: Table listing the employed instances of the set `A` (in total 21) for the empirical evaluation. The set `A` was proposed in Augerat et al. (1995). The node locations were randomly chosen from a square grid (Uchoa et al., 2017).

column generation approach embedded in a generic and modern branch-price-and-cut (BPC) algorithm. *BaPCod* takes a compact mixed-integer programming model as input and solves it using a Dantzig-Wolfe reformulation (Dantzig et al., 1960). See the previous discussion on section 2.4 and chapter 4. This modern BPC solver automatically applies the Dantzig-Wolfe reformulation. In this case, a default generic pricer based on a MIP optimizer is employed during the column generation phase. *BaPCod* uses an automatic dual price smoothing stabilization, as discussed in Pessoa et al. (2018b), to improve the convergence speed of the column generation. *BaPCod* supports direct branching on the master formulation's arc-flow variables as well as Vanderbeck branching (Vanderbeck,

Instance	K	Q	Optimal Value
B-n38-k6	6	100	805
B-n39-k5	5	100	549
B-n41-k6	6	100	829
B-n43-k6	6	100	742
B-n44-k7	7	100	909
B-n45-k6	6	100	678
B-n50-k7	7	100	741
B-n50-k8	8	100	1312
B-n51-k7	7	100	1032
B-n52-k7	7	100	747
B-n56-k7	7	100	707
B-n57-k7	7	100	1153
B-n57-k9	9	100	1598
B-n63-k10	10	100	1496
B-n64-k9	9	100	861
B-n66-k9	9	100	1316
B-n67-k10	10	100	1032
B-n68-k9	9	100	1272
B-n78-k10	10	100	1221

Table 7.4: Table listing the employed instances of the set `B` (in total 19) for the empirical evaluation. The set `B` was proposed in Augerat et al. (1995) The node locations were randomly chosen from a square grid (Uchoa et al., 2017).

2011). The latter is a non-robust branching scheme imposing bounds modifications on the sub-problem variables. Because *BaPCod* is generic, it supports user-developed extensions which can alter the BPC algorithm's behaviour. These extensions can provide the BPC framework with custom-defined cutting planes (robust and non-robust), custom branching decisions (robust and non-robust), or even ad-hoc pricer implementations.

The *VRPSolver* extension (Pessoa et al., 2020a), is a *BaPCod* extension distributed by the same authors. This extension includes an advanced implementation of a bidirectional dynamic programming labeling algorithm (Sadykov et al., 2021a) for solving the pricing problem. The included labeling algorithm can be used as an exact or heuristic pricer. The labeling algorithm contains two successively lighter heuristic implementations; for more information, see Sadykov

Instance	K	Q	Optimal Value
P-n16-k8	8	35	450
P-n19-k2	2	160	212
P-n20-k2	2	160	216
P-n21-k2	2	160	211
P-n22-k2	2	160	216
P-n22-k8	8	3000	603
P-n23-k8	8	40	529
P-n40-k5	5	140	458
P-n45-k5	5	150	510
P-n50-k7	7	150	554
P-n50-k8	8	120	631
P-n50-k10	10	100	696
P-n51-k10	10	80	741
P-n55-k7	7	170	568
P-n55-k8	8	160	588
P-n55-k10	10	115	694
P-n55-k15	15	70	989
P-n60-k10	10	120	744
P-n60-k15	15	80	968
P-n65-k10	10	130	792
P-n70-k10	10	135	827
P-n76-k4	4	350	593
P-n76-k5	5	280	627
P-n101-k4	4	400	681

Table 7.5: Table listing the employed instances of the set `P` (in total 24) for the empirical evaluation. The set `P` was proposed in Augerat et al. (1995). The instances were created by changing the capacities of a few occurrences of the `A`, `B` and `E` sets (Uchoa et al., 2017).

et al. (2021a). The labeling algorithm makes use of a generalized ng-sets definition (Baldacci et al., 2011) defined through the *packing* and *elementarity* sets, as described in Pessoa et al. (2020a). When the solution obtained after the CG convergence is fractional, the ng-sets are dynamically augmented, see Pessoa et al. (2020a) for more details. The *VRPSolver* extension, also, includes the implementation of some specific cutting planes and branching decisions aimed at efficiently solving routing-like problems (or problems that exhibit similar struc-

tures) such as the CVRP, VRPTW, and also others (see Pessoa et al., 2020a). The *VRPSolver* implements the robust RCC cuts of Laporte et al. (1983), and the non-robust limited-memory rank-1 cuts of Pecin et al. (2017b). The *VRPSolver* also implements the non-robust Ryan&Foster branching (Ryan et al., 1981). The *VRPSolver* extension was also used successfully in Pessoa et al. (2020b) to solve the Bin Packing Problem (BPP).

Currently, *BaPCod* and its *VRPSolver* extension are the leading cutting-edge technologies for solving vehicle routing problems (Pessoa et al., 2020a). The *BaPCod* source code is available for free, for academic purposes only, by issuing a form request to this URL: `https://bapcod.math.u-bordeaux.fr/`. The *VRPSolver* extension, on the other hand, is only available in compiled form, and must be explicitly requested via email by contacting one of the original authors: `mailto:ruslan.sadykov@inria.fr`. The official technical report presented in Sadykov et al. (2021b) contains instructions for building, configuring, and using *BaPCod* and its *VRPSolver* extension. To round things out, a Julia language interface to the *VRPSolver* extension can be found at this Github repo[2].

For a comprehensive technical discussion of the components of modern and advanced BPC frameworks, see Sadykov (2019).

7.3 Evaluation Setup

The goal of this thesis, as previously stated, is to determine whether a branch-and-cut framework can be competitive in solving the pricing problem, particularly as the routes grow in length. We hypothesized that, since the labeling algorithm's performance degrades as the vehicle capacity increases, a branch-and-cut framework could aid in solving the more demanding pricing problems. Refer back to the discussion in sections 1.2 and 4.3 for additional details.

However, the two frameworks operate in very different ways. Our branch-and-cut framework does not support non-robust inequalities or variable fixing but produces stronger dual bounds. Similarly, the labeling algorithm instead generates weaker dual bounds but allows for multiple column generation per

[2]Github repo: `https://github.com/inria-UFF/BaPCodVRPSolver.jl`

pricing invocation, non-robust cut generations and branching. As a result, comparing the performance of the two schemes is not an easy task. So a natural question arises. What is the best approach to compare the performance of the two frameworks?

When it came to developing our branch-and-cut pricer, we had two options: (i) developing our pricer in tight integration with the BPC framework, by coding it as a *BaPCod* plugin, or (ii) developing the pricer in a standalone executable, separate from the BPC framework.

The first option has two primary benefits. For starters, it enables our branch-and-cut pricer to influence the BPC algorithm's execution flow. Second, it empowers more diversified ways of measuring the pricers' effectiveness. We can decide to compare the running time required for solving the root node, the running time for solving the entire CVRP instance or the running time of each single pricing iteration. The disadvantage of the first approach is its implementation complexity and constraint in tooling or programming languages employed. Programming the BAC pricer as plugin requires heavy knowledge of the *BaPCod* framework's intricacies.

The second option, on the other hand, has one significant advantage: it allows for greater implementation flexibility, simplifying development efforts. We can develop the BAC pricer in complete isolation, using whatever programming language or external library we want. However, the branch-and-cut pricer cannot change the BPC algorithm's execution flow. As a result, the branch-and-cut pricer follows the same execution flow as the BPC algorithm using the labeling algorithm. This scenario led to two primary downsides. For starters, improved dual-bounds from the BAC algorithm will not affect the number of pricing iterations. As a result, the BAC algorithm's solution to the more difficult elementary SPPRC has no practical way to pay off its high computation times. Secondly, the time required to solve each pricing iteration is the only viable metric to measure the pricers' competitiveness.

In the end, we decided on the second option. We chose simplicity over complexity and built our pricer as a standalone executable. Measuring the running time of each pricing iteration is neither an appropriate nor a poor approach.

There are both benefits and downsides to this approach. First *BaPCod* must be configured to operate in the same domain as our pricer, thereby disabling many features that could potentially result in high efficiency in the broader scheme. Second, our branch-and-cut pricer generates elementary bounds, naturally solving a harder problem. While solving a single pricing iteration may take longer, stronger dual-bounds may skip some pricing iterations in the long run, thus resulting in an overall faster CVRP resolution. On the other hand, measuring the pricers' efficiency per each single pricing iteration, it is straightforward to assess compared to the alternative approaches described.

Summing up, it is clear that the performance comparison that we've settled on is just a mere indicator of the genuine efficiency of the two approaches and must therefore be taken cautiously. We believe, however, there is still value in performing such a comparison as it provides a first proof-of-concept/direction regarding the feasibility of the effectiveness of a branch-and-cut scheme for addressing the pricing problem.

We conclude this section by introducing the *BaPCod* parametrization, which we used to bend the labeling algorithm to operate in a compatible environment with the BAC algorithm.

The configuration parameters are specified separately in an appropriately placed configuration file. The BPC framework's branching and cut-generation schemes have been disabled, thus asking the algorithm to halt at the root node of the branch-and-bound tree. We enabled the ng-sets' augmentation and set the maximum ng-set threshold as high as possible, therefore putting pressure on the labeling algorithm to generate dual bounds as close to the elementary bound as possible at the end of each CG iteration. The pricer's tailing-off condition was disabled. We've also asked the labeling algorithm to generate a single column per pricing invocation. Refer to appendix B for a complete list of the employed configuration parameters.

7.4 Evaluation Process

We employed *BaPCod* version 0.66 (released in November 2021) and the `libRCSP v0.5.12`. The latter library contains the *VRPSolver* extension's implementation. The goal is to use *BaPCod* to solve many CVRP instances while simultaneously measuring the labeling algorithm's running time at each pricing invocation.

We re-adapted one of the VRPTW examples (included in the distribution) to model a Capacitated Vehicle Routing Problem by following the *BaPCod* technical document of Sadykov et al. (2021b). The master formulation that we've implemented follows the two-index arc flow model presented in eqs. (2.2)–(2.7), excluding the Rounded Capacity Constraints (RCC) of eq. (2.5).

To use the *VRPSolver* extension's labeling algorithm, the associated Resource Constrained Shortest Path (RCSP) sub-problem must be defined. The RCSP sub-problem is formulated on a complete directed network by linking the new RCSP modeling variables to the master problem formulation. The RCSP sub-problem necessitates the correct definition of the following: the source/sink vertices, as well as the so-called *packing* and *elementarity* sets (see Pessoa et al., 2020a for more details). The correct definition of these generalized sets is required for specific components of the *VRPSolver* extension to function properly (ng-sets, RCC separation, etc). For each customer, a unique *packing* set and *elementarity* set are created. The *VRPSolver* extension also necessitates the explicit definition of an additional distance matrix encoding the distance between pairs of elementarity sets. The elementarity sets and the distance matrix specified by the user let *BaPCod* compute the ng-sets automatically.

Due to internal implementation details of *BaPCod*, the RCSP sub-problem requires the user to specify the consumptions (or demand) for each network's arc. Resource consumptions aren't definable at the vertex level. Therefore, as suggested in Pessoa et al. (2020a), we have defined the resource consumption on the arcs as $q_{ij} = \frac{q_i+q_j}{2} \quad \forall i, j \in V$. This definition is valid and it leads to a symmetric resource consumption ($q_{ij} = q_{ji}$). The symmetric resource consumption property improves the efficiency in pricing by eliminating the need for backward labeling (Pessoa et al., 2020a).

We implemented a custom pricing functor to extract the information required for the performance evaluation. The pricing functor is essentially a glorified callback that can be used to solve the pricing problem through user-defined custom code. Our pricing functor is a simple stubbed implementation that ultimately invokes the pricing functor of the standard *VRPSolver* extension. However, before calling the labeling algorithm, we first do the following. First, we assess the labeling algorithm's performance by measuring its running time. Second, at each pricing invocation, we record the dual variables $\pi \in \mathbb{R}^N$ of the RMP to be later dumped into a dedicated file.

Because of the implementation details of *BaPCod*, the reduced cost of an edge does not follow the simple relationship $\bar{c}_{ij} = c_{ij} - \frac{\pi_i + \pi_j}{2} \quad \forall i, j \in V,\ i \neq j$. By accessing an appropriate field on the MP formulation structure, the user can retrieve the dual variable $\pi_0 \in R$. The remaining dual variables $\pi_i \quad \forall i \in V_0$ must be calculated from the reduced cost of each edge $\bar{c}_{ij}$. Internally, *BaPCod* encodes the reduced cost of an edge using the following relationship:

$$\bar{c}_{ij} = \begin{cases} c_{ij} - \frac{\pi_j}{2} & \texttt{if } \ i = 0, j \in V_0 \\ c_{ij} - \frac{\pi_i + \pi_j}{2} & \texttt{if } \ i, j \in V_0,\ i \neq j. \end{cases} \tag{7.1}$$

By exploiting eq. (7.1), we've implemented a trivial recursive algorithm to extract all the necessary dual variables $\pi \in \mathbb{R}^N$.

In conclusion, we save two files to the hard drive for each pricing invocation of the labeling algorithm. The first file is a JSON file that contains metadata information about the labeling algorithm, such as the instance name, the instance size, the total running time for the labeling, the reduced cost of the generated route and the column generation iteration. The second file, instead, encodes a CPTP instance including: the dual variables (the vertices' profit), the original demands, the original node positions, and the vehicle capacity. The CPTP instance is encoded in a slightly modified `TSPLIB95` file format. In the file format, we've added a new dedicated section, called `PROFIT_SECTION`, to hold the extracted dual variables.

Only the last 10 of all the pricing problems dumped to disk are kept for empirical evaluation. As a result, for a fixed inflation scale factor s, we totaled 70

pricing instances from the `E` test set, 30 from the `F` test set, 210 from the `A` test set, 190 from the `B` test set and 240 from the `P` test set.

7.4.1 Performance profiles

Performance profiles, first introduced in Dolan et al. (2002), are a plot representation used to benchmark optimization software. These novel plot representations are easy to interpret and less susceptible to personal interpretation. In a nutshell, performance profiles represent the cumulative distribution function of a performance metric. We put H algorithms to the test by running them through M problem instances. The metric under consideration in *performance profiles* is typically the overall running time of an algorithm. When the metric under measurement is the running time, we obtain the so-called *time profiles*.

In *time profiles*, the time ratio relative to a baseline is plotted on the X-axis. The baseline is calculated as the best performance obtained from all the algorithms under consideration. Instead, the Y-axis depicts the likelihood of being within an X ratio of the baseline. In *time profiles*, the ideal solver appears as a straight vertical line on the entire left of the plot. An example of a *time profile* is provided in fig. 7.1.

In our case, we also employ the so-called *cost profiles* to plot either the optimal value or the best dual-bound found from each algorithm. Instead of computing its ratio w.r.t. a baseline, the raw value of the metric is plotted directly. Recording the dual bounds inside a *cost profile* allows us to select the best algorithm based on its convergence speed over a limited time frame (time-limit). When plotting the optimal value in the *cost profiles*, on the other hand, we can validate the pricer that would feed the BPC framework with tighter dual bounds. Because the optimal value z always satisfies $z \leq 0$, we used two sentinel values to accentuate some specific circumstances. When a pricer successfully determines that no reduced cost route exists ($z \geq 0 - \varepsilon_{\text{ct}}$), we output $z = 1.0$ in the associated *cost profile*. Instead, if the pricer fails to solve the PP optimally within the timelimit, we output $z = 2.0$. For additional output clarity, in *cost profiles* we tint the plot's background color in green and red for $z < 0 - \varepsilon_{\text{ct}}$ and $z \geq 0 - \varepsilon_{\text{ct}}$ respectively.

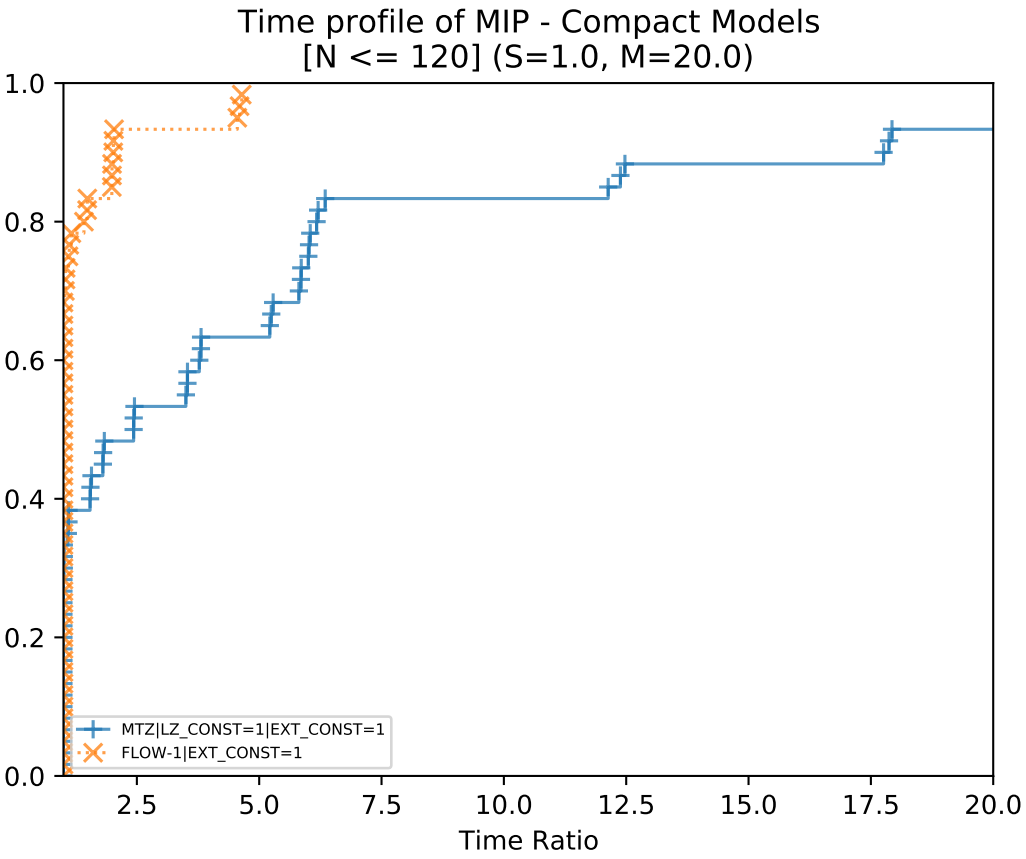


Figure 7.1: This *time profile* example depicts the running time analysis of two exact algorithms for the TSP. A vertical straight line on the entire left side of the plot would represent the best scenario. In the situation depicted in the figure, the algorithm named `FLOW-1|EXT_CONST=1`, the orange line, outperforms the other algorithm in the vast majority of cases.

In this thesis, we evaluated the efficiency of our proposed BAC-based pricer using both the *cost* and *time profiles*. Our pricer's efficiency is compared to the RCSP dynamic programming algorithm included in the *VRPSolver* extension (Pessoa et al., 2020a) of *BaPCod*.

7.4.2 Performance Variability

Performance variability (Lodi et al., 2013) can be defined as an unexpected change in performance that is not the result of a genuine improvement. There are numerous cases of variability, but one of the most common is randomness,

specifically the initial state of random generators. Resolution methods that rely on randomness may perform well by chance, and the effect is magnified for the so-called *chaotic systems*. A slight perturbation to the initial conditions has a tremendous impact on the evolution of a chaotic system.

Branch-and-cut algorithms are chaotic systems by nature. Unlucky branching decisions, appearing at high levels of the branching tree, have an exponential impact on the size of the tree itself.

Performance profiles should filter out the noise and measure the genuine algorithmic improvements of a resolution method. To avoid performance variabilities, we can upsample each test-set instance using a different initial random seed each time. The greater the number of seeds considered, the more likely the performance profile captures a genuine algorithmic improvement rather than variability.

Although using multiple seeds is generally good practice when testing chaotic systems, we used a single seed per pricing instance to substantially decrease the computation time required for the empirical evaluation.

7.5 Empirical Results

The empirical results are presented in this section. The results were obtained on a desktop computer equipped with an AMD Ryzen 7 1700X 3.4GHz with 8 cores (16 virtual cores), 2x8 GBs of memory @ 3200 MHz, running a GNU/Linux operating system with kernel v5.18.3 and Glibc v2.35. For our BAC pricer's implementation, we used CPLEX version 22.1.

We divide the empirical results into two groups.

The first group is committed to testing various versions of the implemented BAC algorithm based on the aggressiveness used for the fractional labeling discussed in section 6.4.2. In the first version of the algorithm, we perform cutting-plane separation aggressively on each possible occasion. We call this version of the algorithm `BAC EFL`, where `EFL` stands for Exhaustive Fractional Labeling. `BAC EFL` computes the Gomory-Hu tree on each fractional solution. The second version of the algorithm is known as `BAC AFL` (Amortized Fractional Label-

ing). BAC AFL amortizes the cost of fractional labeling over multiple iterations. Namely, fractional separations iterations that are not multiple of N become a no-op, no min-cuts computations are performed, and thus no cutting-planes are separated. In the last version of the algorithm, which we call BAC NFL, we perform no fractional labeling (NFL stands for No Fractional Labeling). The cost of computing Gomory-Hu trees becomes zero, but cutting planes will never be separated for fractional solutions.

The computational evaluation of the first group is provided in fig. 7.2, which contain solely cost profiles depicting the obtained dual bounds. The results in fig. 7.2 were obtained using a single core and a time limit of 60 seconds. The winner will be the BAC-pricer version that produces tighter dual bounds within the time limit. According to fig. 7.2, the BAC AFL version produces tighter dual bounds in the majority of cases. As a result, we will only compare the pricing efficiency of the BAC AFL version to the labeling algorithm in the second group of empirical results.

The second group, on the other hand, compares our BAC AFL algorithm to the *VRPSolver*'s labeling algorithm (Pessoa et al., 2020a). The computational evaluation of the second group is provided in figs. 7.3–7.12. The BAC-pricer was run on 16 threads (corresponding to the virtual cores of the host machine) with a time limit of 20 minutes. The empirical results of figs. 7.3–7.12 will be discussed in the next dedicated section 7.5.1.

7.5.1 Discussion of the Empirical Results

In this section we discuss the empirical results depicted in figs. 7.3–7.12. The *VRPSolver*'s labeling algorithm in the empirical results is denoted as libRCSP DP pricer, while our implemented BAC-pricer is denoted as BAC MIP Pricer (AFL).

We tested the performance of the two pricer approaches for different test sets under progressively higher vehicle capacities. We tested for different inflation scale factors s, namely $s \in \{1,2,4,5,8,10,20\}$. See section 7.1.1 for a discussion on how the inflated CVRP instances were obtained.

The empirical results of figs. 7.3–7.12 are subdivided as follows:

- The empirical evaluation for the `E` test set is contained in figs. 7.3 and 7.4.
- The empirical evaluation for the `F` test set is contained in figs. 7.5 and 7.6.
- The empirical evaluation for the `A` test set is contained in figs. 7.7 and 7.8.
- The empirical evaluation for the `B` test set is contained in figs. 7.9 and 7.10.
- The empirical evaluation for the `P` test set is contained in figs. 7.11 and 7.12.

The situation depicted by the empirical results is summarized in the remainder of this section.

In the case of the `E` test set, we can see that at $s \leq 4$, the BAC-pricer is not sufficiently competitive but was able to feed slightly better dual bounds to the BPC framework. The BAC-pricer begins to compete with the labeling algorithm at $s = 5$. For $s \in \{8, 10, 20\}$, not only is the BAC-pricer faster than the labeling algorithm but feeds substantially better dual bounds to the BPC framework. At $s = 20$, the labeling algorithm is 20 times slower than the BAC-pricer in 42% of the cases.

In the case of the `F` test set, we can see that at $s \in \{1, 2\}$, the BAC-pricer is not competitive and occasionally fails to complete the pricing problem within the time limit of 20 minutes. This scenario is most likely caused by the `F-n135-k7` instance, for which we have empirically observed to require a massive amount of cutting planes, resulting in memory exhaustion for the host machine. At $s \in \{4, 5, 8, 10, 20\}$, the BAC-pricer performs progressively better and feeds tremendously tighter dual bounds to the BPC framework.

In the case of the `A` test set, the BAC-pricer is not competitive at $s \in \{1, 2, 4\}$ and occasionally fails to complete the pricing problem within the time limit. At $s = 5$, the BAC-pricer begins to be competitive in terms of running time. The BAC approach yields no discernible dual-bound improvements to the BPC framework.

In the case of the `B` test set, the BAC-approach becomes competitive at $s = 5$. At $s \leq 4$, it fails to complete the pricing problem within the time limit. At $s = 20$, the BAC-pricer outperforms the labeling algorithm. When $s = 10$, the BAC-pricer produces slightly tighter dual-bounds, but significantly better ones when $s = 20$.

In the case of the P test set, the BAC-approach becomes competitive at $s = 5$. At $s = 8$, the BAC approach outperforms the labeling algorithm while yielding significantly better dual bounds to the BPC framework.

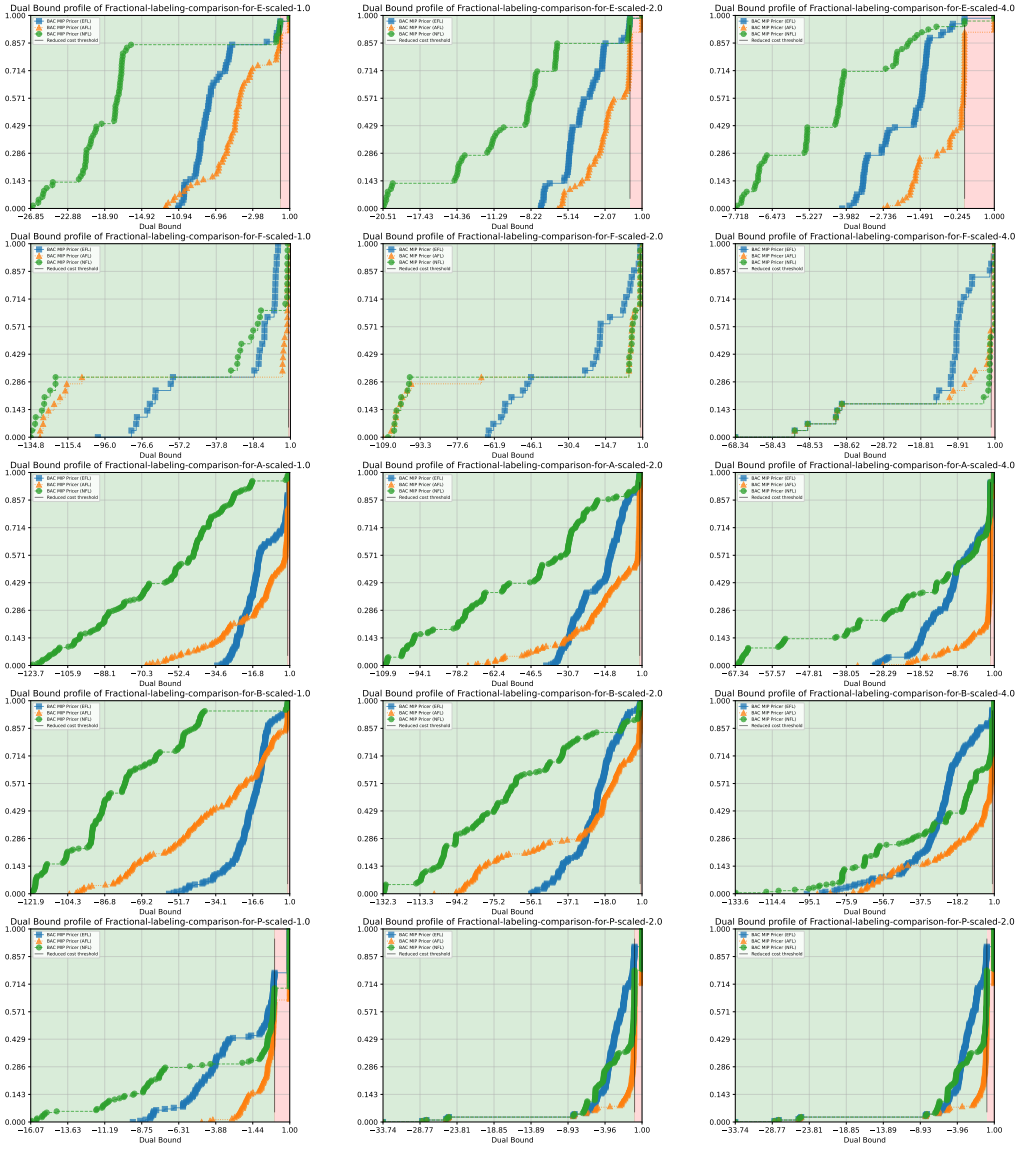


Figure 7.2: This empirical study investigates the tightness of the dual bound for various versions of the BAC algorithm. The plots are laid out in a matrix format. Each row corresponds to a unique CVRP test set. From top to bottom: `E`, `F`, `A`, `B` and finally `P`. Each column represents a different inflation scale factor s. From left to right: $s = 1$, $s = 2$ and finally $s = 4$. The version that uses amortized fractional labeling (`AFL`), represented here by the orange line, appears to produce tighter dual bounds in the majority of cases.

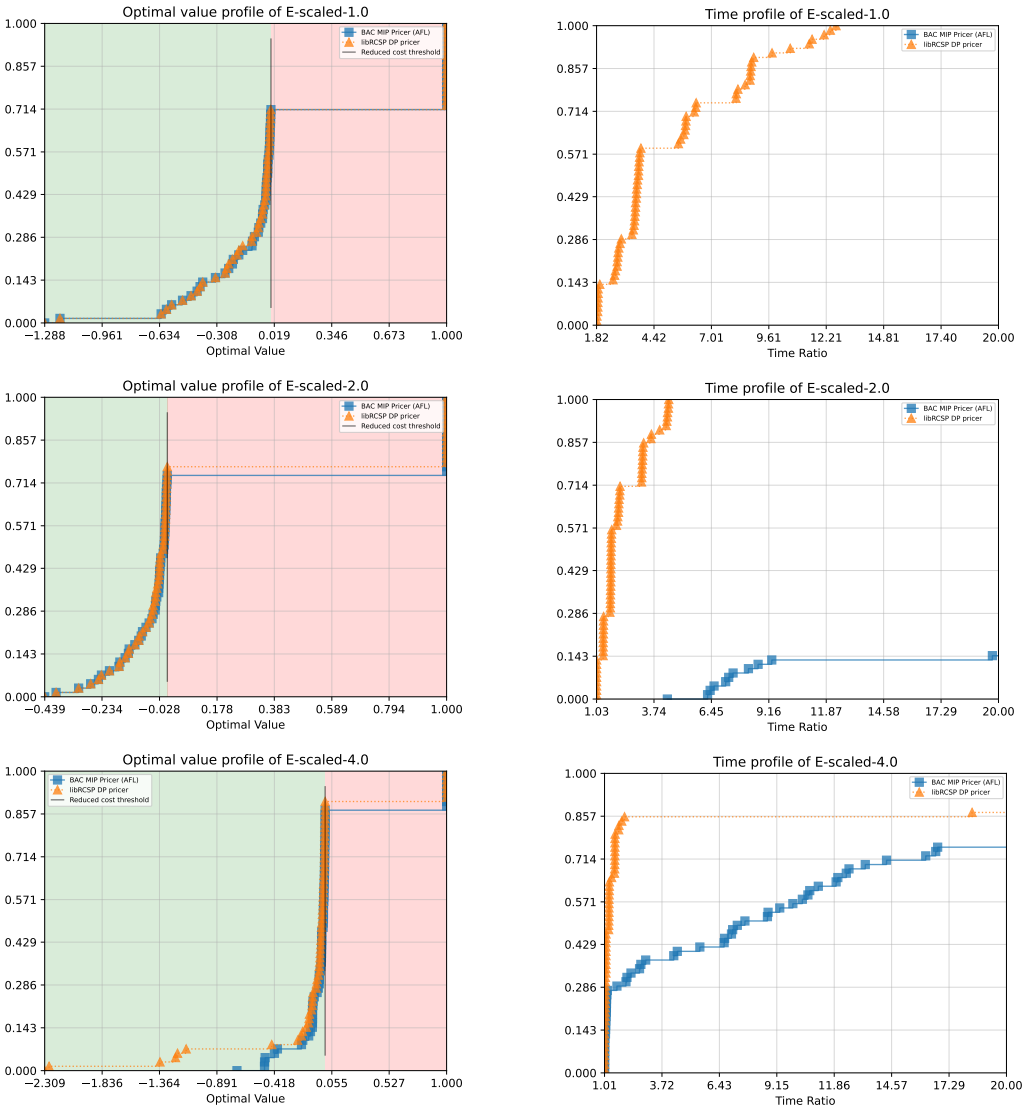


Figure 7.3: Empirical study comparing the pricing performance of the proposed BAC-pricer to the labeling algorithm of Pessoa et al. (2020a) on the E test set. On the left, the cost profile representing the optimal value found by each method. On the right, the time profile measuring the time ratio of the two approaches. Each row represents a different inflation scale factor s. From top to bottom: $s = 1, 2$ and 4.

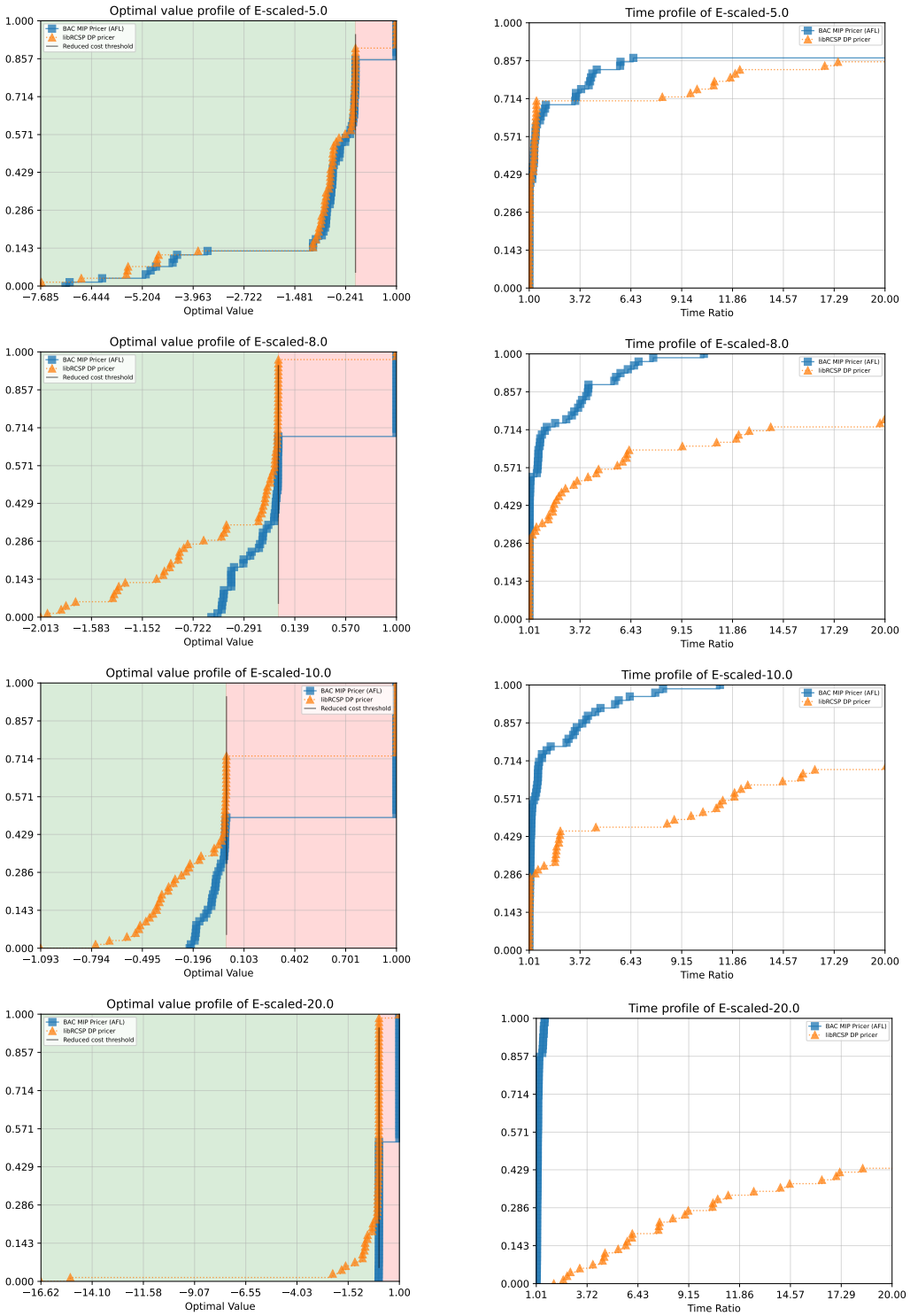


Figure 7.4: Continuation of the empirical study on the E test set from fig. 7.3 testing even further inflation scale factors: $s = 5, 8, 10$ and 20.

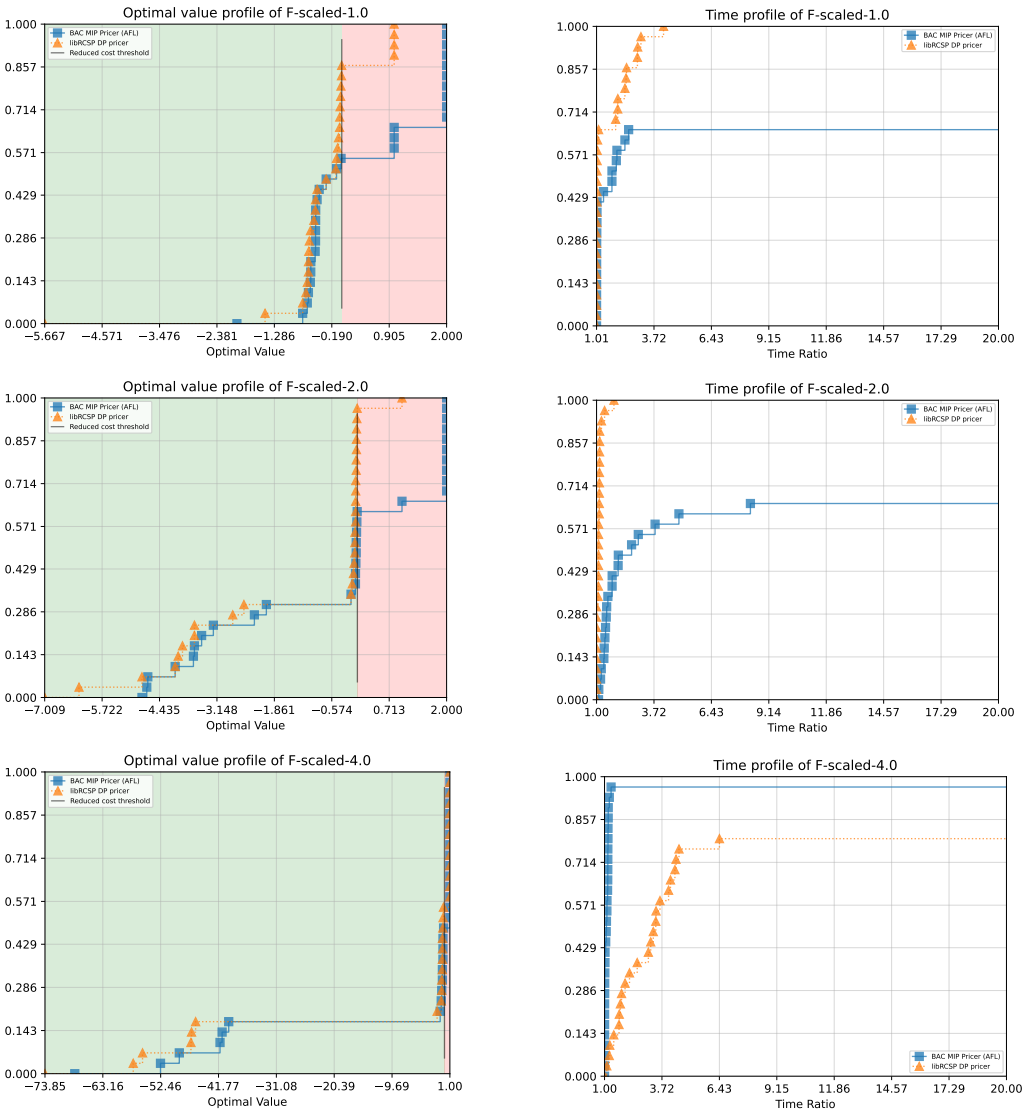


Figure 7.5: Empirical study comparing the pricing performance of the proposed BAC-pricer to the labeling algorithm of Pessoa et al. (2020a) on the F test set. On the left, the cost profile representing the optimal value found by each method. On the right, the time profile measuring the time ratio of the two approaches. Each row represents a different inflation scale factor s. From top to bottom: $s = 1, 2$ and 4.

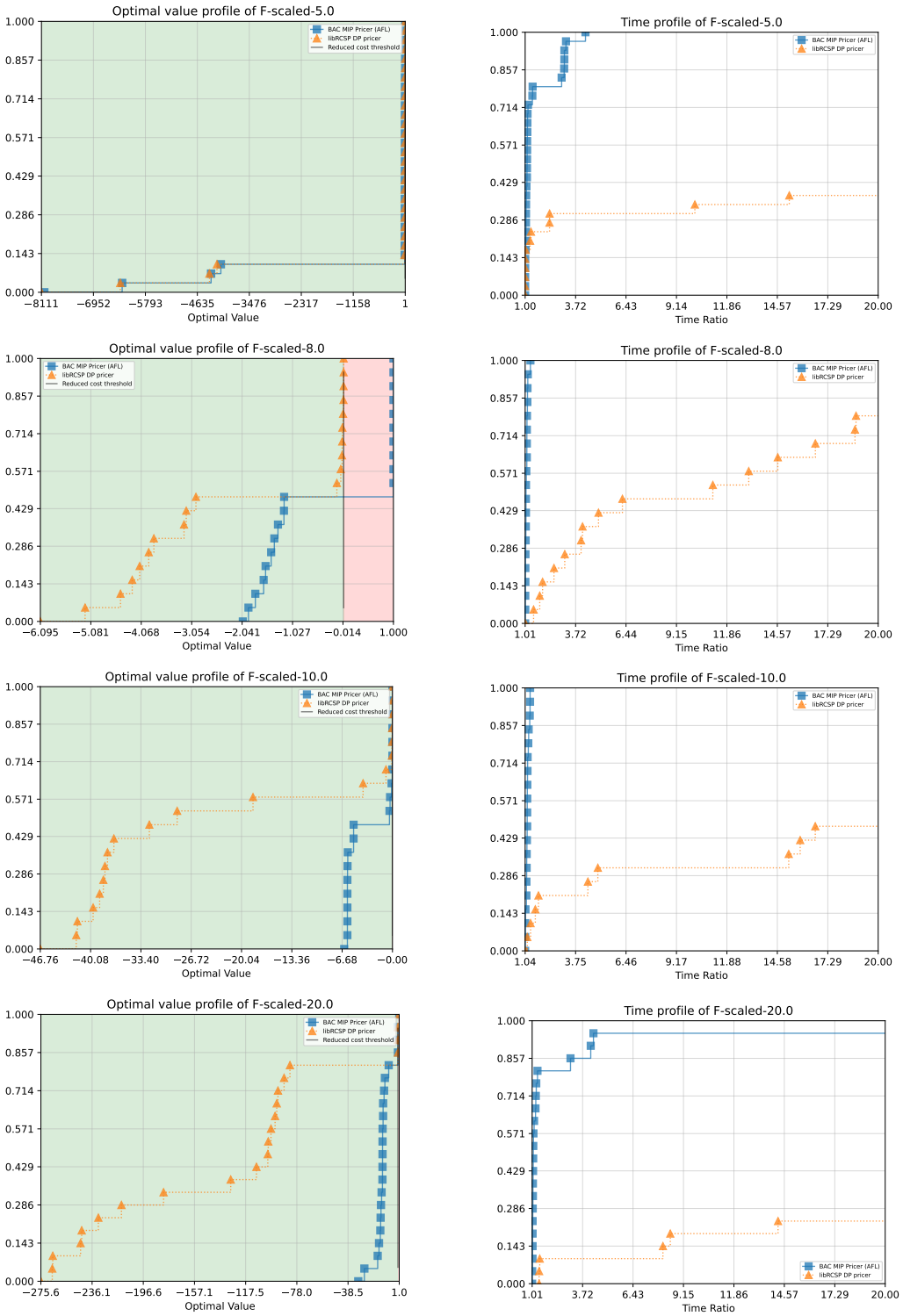


Figure 7.6: Continuation of the empirical study on the F test set from fig. 7.5 testing even further inflation scale factors: $s = 5, 8, 10$ and 20.

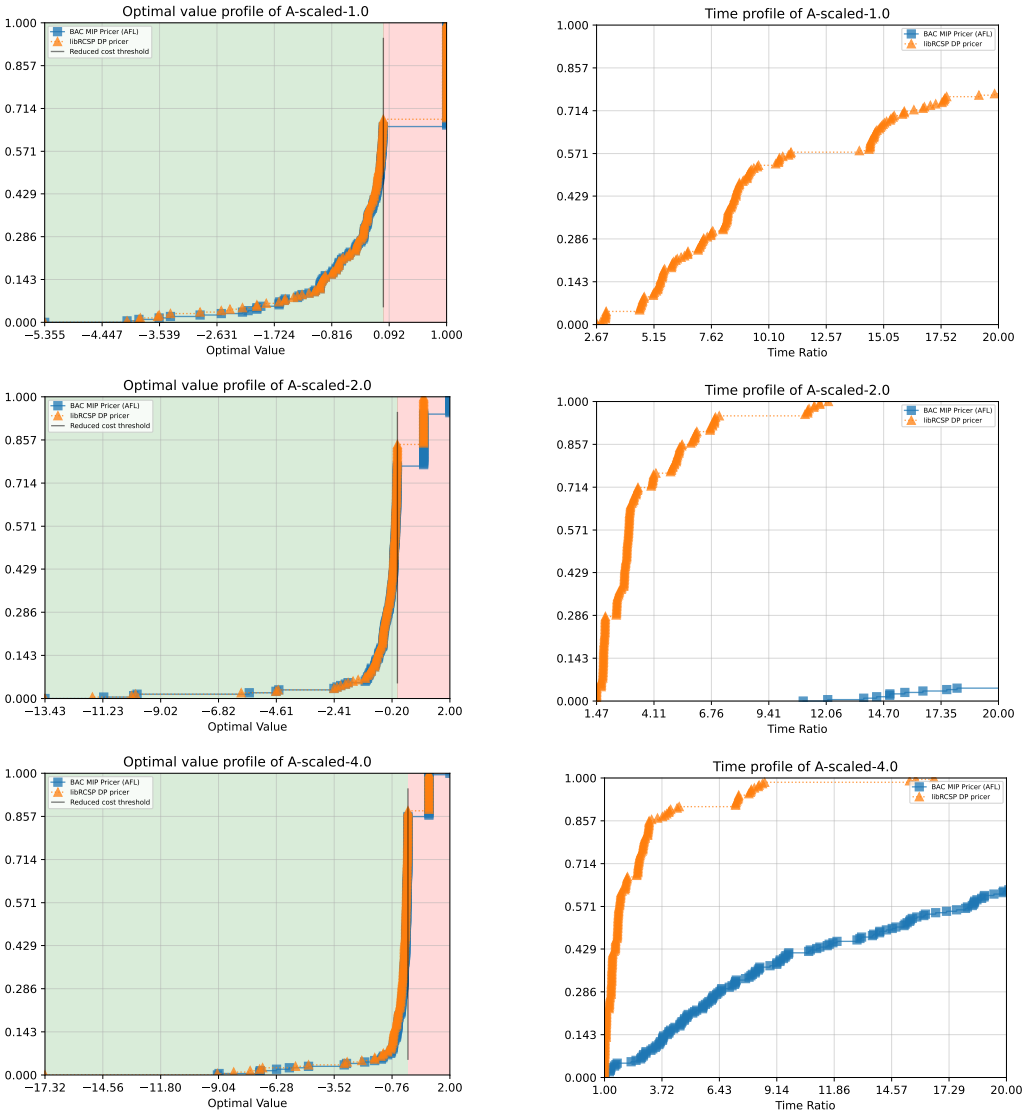


Figure 7.7: Empirical study comparing the pricing performance of the proposed BAC-pricer to the labeling algorithm of Pessoa et al. (2020a) on the `A` test set. On the left, the cost profile representing the optimal value found by each method. On the right, the time profile measuring the time ratio of the two approaches. Each row represents a different inflation scale factor s. From top to bottom: $s = 1, 2$ and 4.

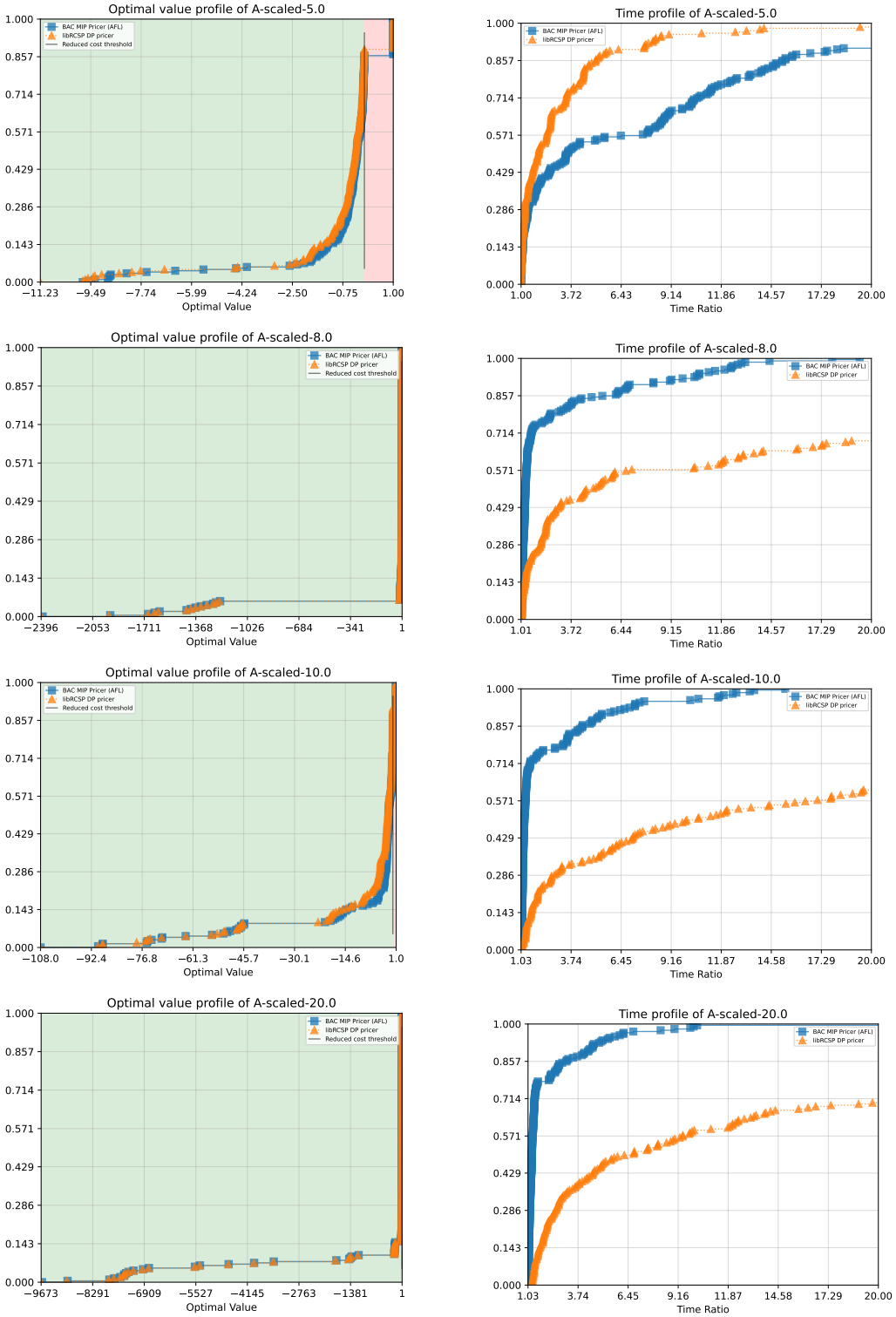


Figure 7.8: Continuation of the empirical study on the A test set from fig. 7.7 testing even further inflation scale factors: $s = 5, 8, 10$ and 20.

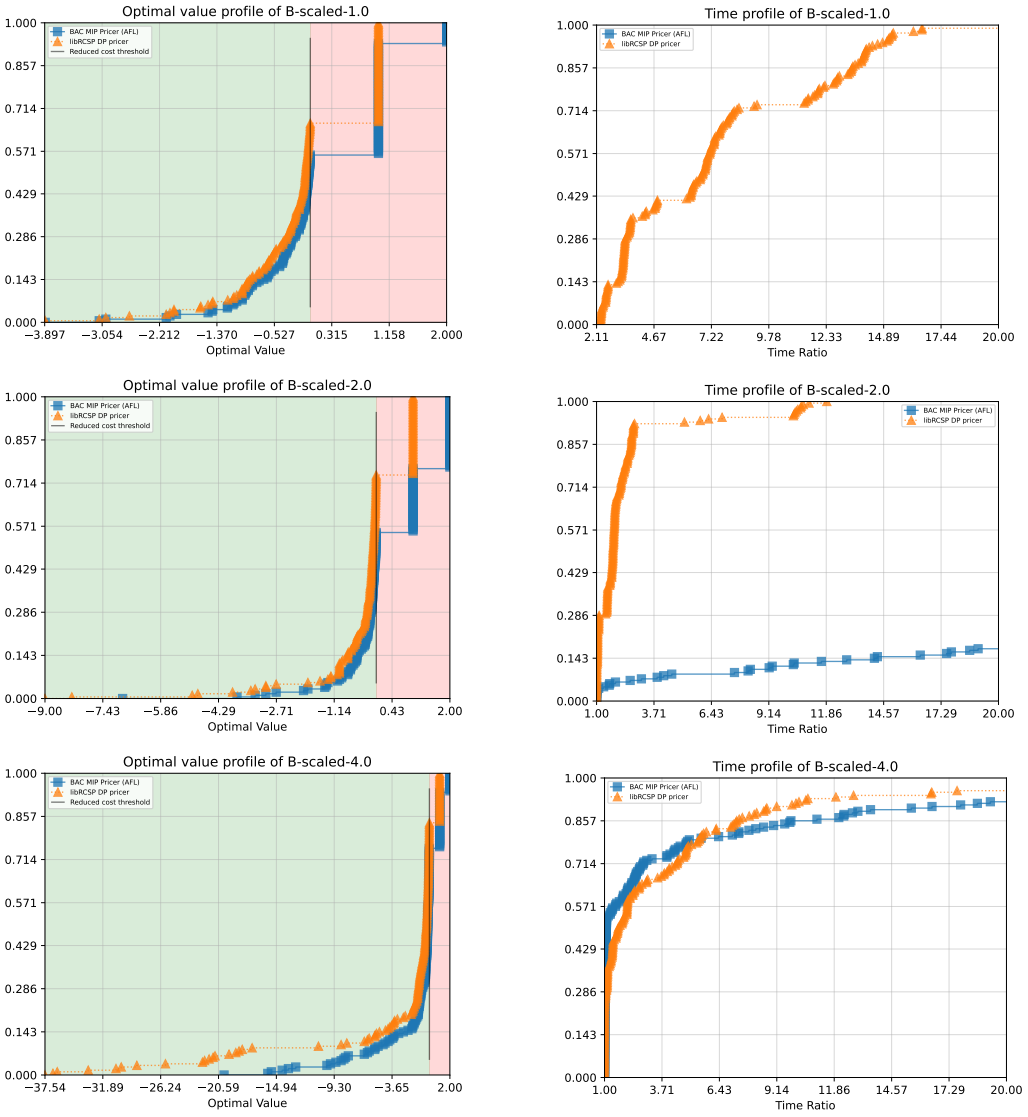


Figure 7.9: Empirical study comparing the pricing performance of the proposed BAC-pricer to the labeling algorithm of Pessoa et al. (2020a) on the B test set. On the left, the cost profile representing the optimal value found by each method. On the right, the time profile measuring the time ratio of the two approaches. Each row represents a different inflation scale factor s. From top to bottom: $s = 1, 2$ and 4.

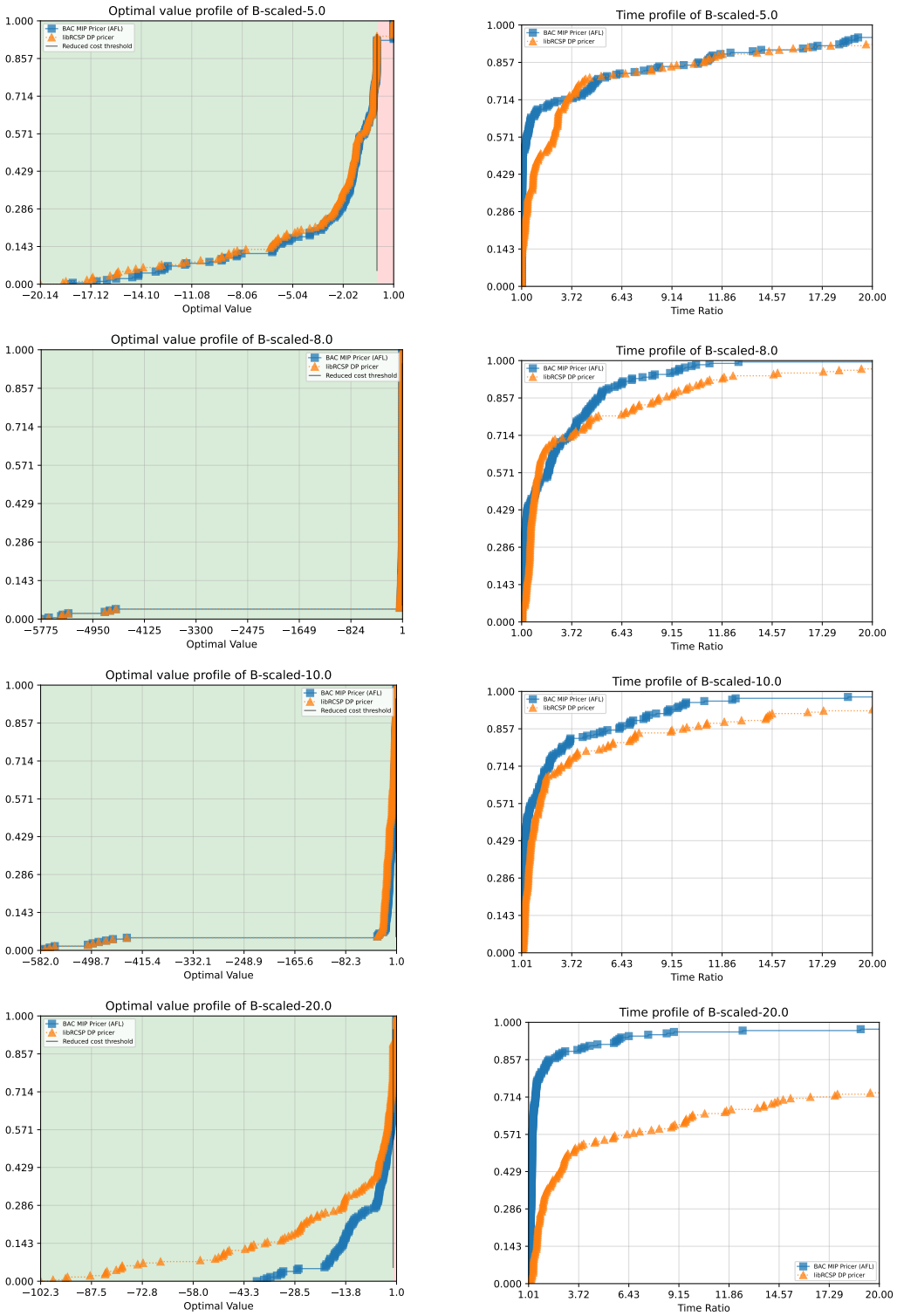


Figure 7.10: Continuation of the empirical study on the B test set from fig. 7.9 testing even further inflation scale factors: $s = 5, 8, 10$ and 20.

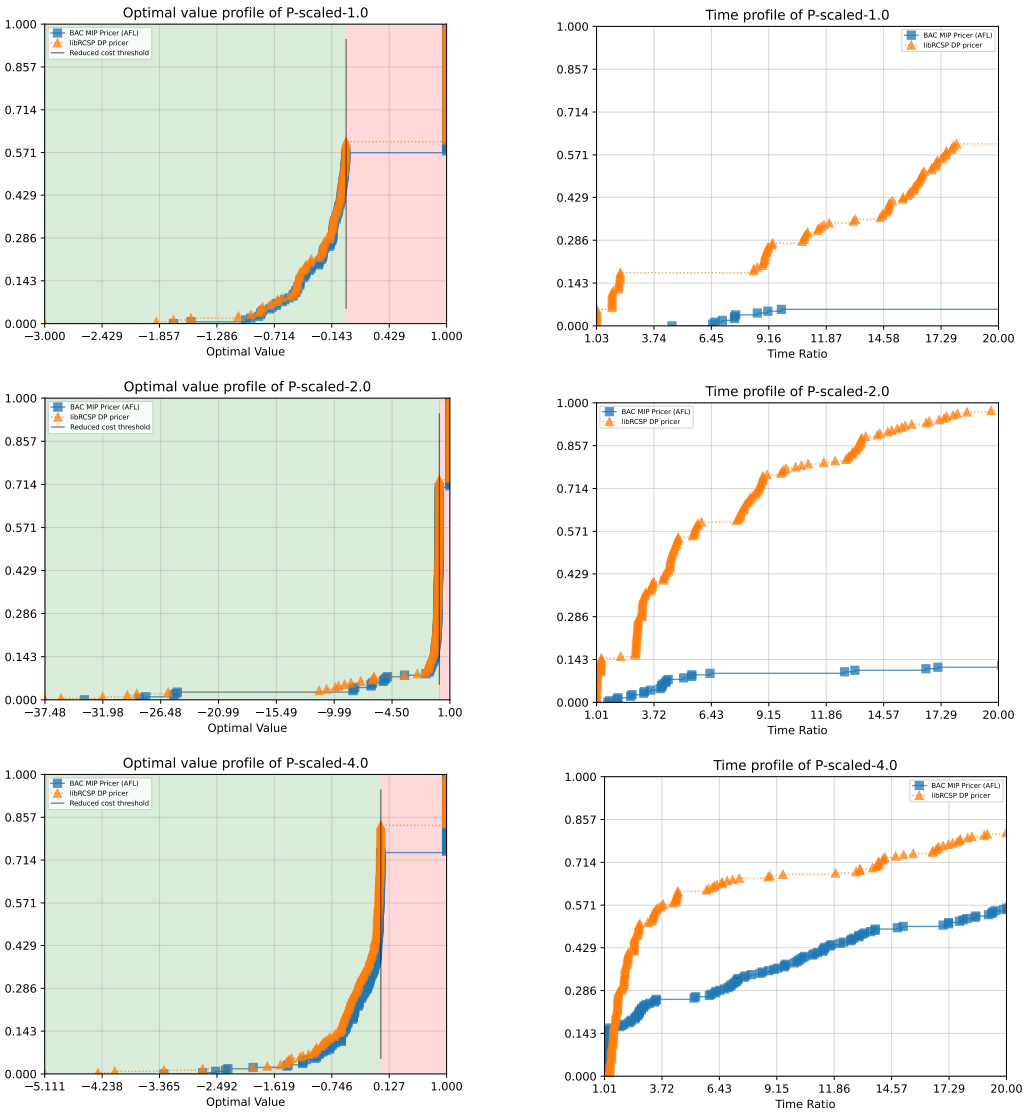


Figure 7.11: Empirical study comparing the pricing performance of the proposed BAC-pricer to the labeling algorithm of Pessoa et al. (2020a) on the P test set. On the left, the cost profile representing the optimal value found by each method. On the right, the time profile measuring the time ratio of the two approaches. Each row represents a different inflation scale factor s. From top to bottom: $s = 1, 2$ and 4.

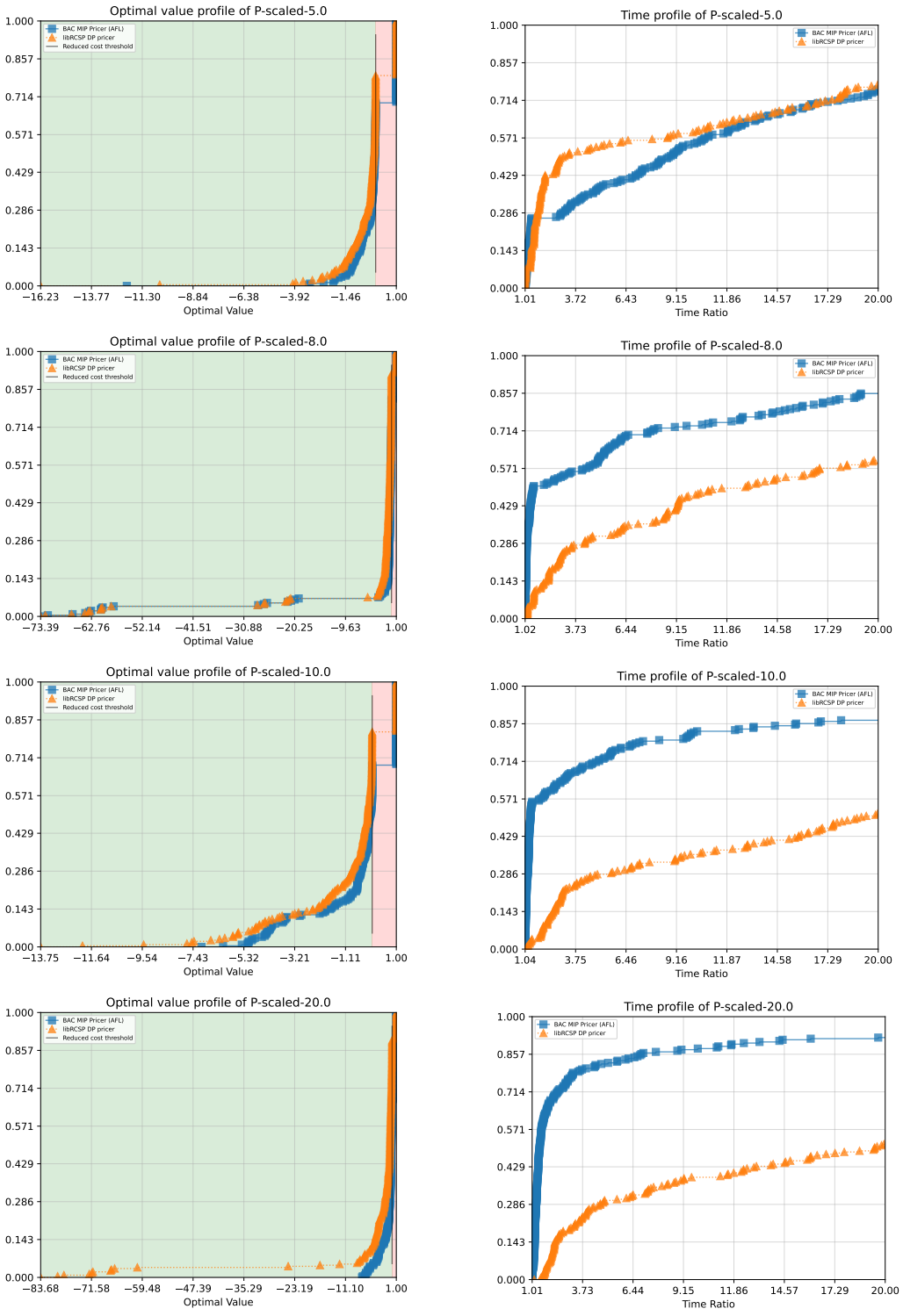


Figure 7.12: Continuation of the empirical study on the P test set from fig. 7.11 testing even further inflation scale factors: $s = 5, 8, 10$ and 20.

CHAPTER 8

Conclusions

We proposed a branch-and-cut framework for solving the pricing problem induced by Column Generation (CG) approaches when applied to Capacitated Vehicle Routing Problems (CVRPs).

The first portion of the thesis provided a theoretical foundation for the CVRP while examining notable contributions to CVRP's exact algorithms. The dynamic-programming-based label-correcting algorithm, proposed in Desrochers et al. (1992) and Feillet et al. (2004), is yet to this day a fundamental component for solving the pricing problem in contemporary VRP solvers. We examined the label-correcting algorithm and discussed additional contributions in the pricing problem domain.

We identified the labeling algorithm's limitations in solving pricing problems with non-stringent vehicle capacities and its inability to scale to multiple machine cores. As a solution, we proposed a BAC algorithm to address the pricing problem, the implementation of which was provided in chapter 6. The BAC algorithm was built on top of the CPLEX MIP optimizer, with the added benefit of scaling to multiple machine cores effortlessly.

Despite the inherent operational differences between the two approaches, we conducted an empirical study in chapter 7 to assess their performance as the associated CVRP vehicle capacity increases. Our analysis revises and vastly supplements the previous study published in Jepsen et al. (2014).

The empirical results were discussed in sections 7.5 and 7.5.1. While we

did not achieve outstanding results in all cases, we demonstrated that the BAC framework outperforms the labeling algorithm as the vehicle capacity bound increases. Our results imply that further research on BAC-based pricer approaches may provide additional benefits to modern CVRP solvers. BAC-based pricers may be integrated within branch-and-price (BAP) frameworks supplementing the traditional dynamic programming labeling algorithm.

8.1 Improvements and Future Work

In the section section 8.1, we discuss improvements and future work regarding our BAC-based pricer.

The labeling algorithm for pricing q-routes of Desrochers et al. (1992) could be integrated to provide sensible dual bounds to the CPLEX MIP optimizer in the early stages. Preprocessing algorithms may also be studied and implemented to reduce the search-space size of the CPTP.

Mathheuristics (Fischetti et al., 2018), such as *local branching* (Fischetti et al., 2003) or *hard-fixing*, could be used to transform an exact BAC algorithm into a heuristic one. Mathheuristics shrinks the search space by imposing additional invalid constraints. These constraints typically limit the search space to solutions resting "closely" to a candidate point. The utility of mathheuristic approaches should be investigated further in the context of pricing.

Readjusting the *decremental state-space relaxation* (DSSR) of Boland et al. (2006), Righini et al. (2008), and Martinelli et al. (2014) to a BAC approach, may also be a viable option for accelerating convergence towards optimal solutions. Initially, the BAC-pricer could be asked to generate non-elementary q-routes. The elementarity condition could then be gradually enforced over time using an integral cutting-planes approach by strengthening the bounds associated with the $y_i \quad \forall i \in V$ decision variables that violated the elementarity condition.

Porting the ng-routes relaxation (Baldacci et al., 2011) to a BAC framework could be an intriguing research topic, though it may be unrealistic to accomplish.

It will be interesting to see if paradigm shifts from the employed CPTP MIP model could provide performance boosts to the BAC pricer. Compact

formulations (polynomial number of constraints) could be studied in the context of pricing: such as sequential formulation (MTZ) (Miller et al., 1960), single-commodity flow formulations (FLOW-1) (Gavish et al., 1978) or multi-commodity flow formulations (MCF) (Wong, 1980; Claus, 1984). Compact formulations, on the other hand, were empirically proven to be unsatisfactory in Taccari (2016) for the ESPP. Nonetheless, hybridizations of these models, combining decision variables and cutting planes strategies, could be a topic of future evaluation.

The BAC pricer could benefit from novel cutting planes with shorter separation times that achieve reasonable dual-bound improvements. We did not use any custom branching strategies in our implementation. Other branching schemes, such as branching over cut-sets of Rounded Capacity Constraints (RCC), could be advantageous to test. To reduce memory consumption and improve efficiency, we could also tune the cutting planes violation thresholds and use the GSEC sparser formulation of (5.20) whenever possible.

We saw the impact of fractional labeling and fractional separation in sections 7.5 and 7.5.1 and how these two components are critical to the BAC pricer's efficiency. The speed of fractional labeling could be increased by using heuristics to compute reasonable min-cuts. The *Lin-Kernighan heuristic* (Kernighan et al., 1970), for example, could be used to compute reasonable min-cuts in $O(N^{2.2})$ time. Instead, a simpler heuristic alternative is to perform a DFS traversal in the support graph generated by the non-zero fractional solutions $x^\star_e > 0 \quad \forall e \in E$. If the heuristic fractional labeling cannot identify any violated inequality, then the exact fractional labeling of section 6.4.2 can be used instead.

In the future, it would be interesting to investigate the competitiveness of our approach by instead measuring the time required to solve the entire CVRP instance with the BPC algorithm using the BAC algorithm as a pricer. Doing so would require additional implementation efforts to port the BAC-pricer to *BaPCod*, requiring the development of a C++ translation layer to route the *BaPCod* pricing requests to our BAC pricer.

As a final side-note, it would be interesting to see if the same results obtained for the CVRP also apply to the VRPTW. However, this scenario would neces-

sitate radical modifications to the pricer implementation, the MIP model, and the cutting planes. In the VRPTW, the MIP model needs to be modified into an ESPPRC to account for time window slots. Time window slots must be modeled as big-M constraints, which are known to be computationally unstable (Jepsen et al., 2008b). Despite the VRPTW is beyond the scope of this thesis, it could be an interesting subject for studying BAC-based pricers in the future.

APPENDIX A

Introduction to CPLEX

IBM ILOG CPLEX Optimizer[1], CPLEX for short, is a commercial optimization software package for solving problems expressed as either: linear programs, mixed-integer programs, quadratic programs, or quadratically constrained programs. ILOG CPLEX was purchased by IBM in 2009 and is still maintained by IBM. CPLEX is available in two flavors: a standalone executable and a callable library. The executable can solve problems defined interactively or defined inside a custom file format. The CPLEX Callable library is a C dynamic library linkable with custom user code. CPLEX also includes language bindings for C++, Java, Python, and other popular programming languages. CPLEX uses the simplex algorithm (both primal and dual) and an advanced proprietary algorithm based on a branch-and-cut technique called *Dynamic Search*. CPLEX applies pre-processing at the root to improve the formulation and reduce the problem size. It also features a probing technique for analyzing logical implications by setting binary variables to 0 or 1. It applies several heuristics at each node of the tree: diving heuristics, Local Branching (LB) (Fischetti et al., 2003), Feasibility pump (Fischetti et al., 2005), Relaxation Induced Neighborhood (Danna et al., 2005), Evolutionary genetic algorithms for polishing the solutions (Rothberg, 2007), and more. CPLEX includes efficient implementations of several cut templates. Among the most notable: Gomory cuts (Chvátal, 1973), Knapsack covers

[1] IBM ILOG CPLEX Optimizer: https://www.ibm.com/analytics/cplex-optimizer

(Letchford et al., 2020), GUB covers (Wolsey, 1990), Local Branching (Fischetti et al., 2003), Flow covers (Padberg et al., 1985), Clique covers (Brigham et al., 1983), 0-1 half cuts (Caprara et al., 1996), and more. Moreover, the node-selection and branching decision strategies in CPLEX are top-notch. The node-selection and branching decision strategies are the most delicate component of a branch-and-cut algorithm because they can substantially alter the whole running time of the procedure (Lodi et al., 2013). They received many years of development and tuning to make them top-notch. For more details about CPLEX refer to Lima et al. (2010) and Lodi (2013).

MIP solvers are rather general and can be used to solve a wide range of problems from various fields (Bixby et al., 2007). MIP models are, in spirit, a way to mathematically program a solver to achieve the desired solution. A MIP solver can solve a mixed-integer linear programming formulation expressed as (Wolsey et al., 1999):

$$\max_{x,y} \quad c^T x + d^T y \tag{A.1}$$

$$\text{s.t.} \quad Ax + By \leq b \tag{A.2}$$

$$x \in \mathbb{R}^n \tag{A.3}$$

$$y \in \mathbb{Z}^k_+, \tag{A.4}$$

where $A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times k}$ are matrices and $c \in \mathbb{R}^n, d \in R^k, b \in \mathbb{R}^m$ are vector coefficients. The bound in eq. (A.2) can also be rewritten in equality and/or greater form.

MIP programs are solvable in various ways, but the most common is to use the simplex algorithm (for solving the continuous relaxation) and a branch-and-cut algorithm (to handle the integrality constraints). The branch-and-cut algorithm, first introduced in Padberg et al. (1991), is a hybridization between a cutting plane method and a branch-and-bound procedure (Land et al., 2010). The branch-and-bound algorithm is a divide-and-conquer algorithm that manipulates a search tree to explore the solution space. Each node of the tree represents a sub-problem of the feasible region. The continuous relaxation was proven to be solvable in polynomial time in Khachiyan (1979). Even though the simplex algorithm works well in practice, it could rarely take an exponential amount of

time to complete. Regardless, the simplex algorithm works flawlessly in practice.

MIP solvers are exact by nature: given enough running time, they can compute a proven optimal solution by exhibiting both a lower and upper bound to the objective function. They tend to be less effective than an ad-hoc heuristic algorithms since they work with a generic abstract mathematical model and cannot anticipate all the nuances that the model itself represents. As Bixby et al. (1999) points out, MIP modeling is a powerful and convenient technique that wasn't practical 40 years ago due to limited hardware and inadequate implementations. MIP models are now quite manageable and solvable in a reasonable time on consumer hardware, thanks to numerous reasons: advances in processor speed, the development of faster algorithms, embedded heuristics, better preprocessing, post-processing, and polishing techniques.

Unfortunately, MIP solvers are frequently quite complex. Lots of efforts are needed to develop an efficient MIP solver. Unlike a simple heuristic whose implementation spans a few lines of code, a MIP solver cannot be developed in "house" using limited resources within a reasonable time. The most efficient MIP solvers on the market have licenses that cost tens of thousands of dollars.

Fortunately, CPLEX is available freely through an academic license at this URL[2].

[2]URL: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.5.1/ilog.odms.cplex.help/CPLEX/GettingStarted/topics/preface/preface_synopsis.html

APPENDIX B

BaPCod Parametrization

This section exhaustively lists all of the configuration parameters for respectively the *BaPCod* and *VRPSolver* extension. These parameters were employed to obtain the results in chapter 7. We will not provide an exhaustive list of each parameter; instead, we will focus on the significant ones employed to achieve the desired intent. The *BaPCod* technical report (Sadykov et al., 2021b) provides a more comprehensive list and description of the available parameters. The technical report alone is not enough to comprehend the purpose of each parameter, therefore it is highly suggested to supplement the read with the scientific document of Pessoa et al. (2020a).

The parameters we've used will be summarized in the following sections of this appendix, along with their associated values, meaning, and justification for their use. First, in the remainder of this section, we want to provide a summary of the guiding decisions that influenced the choice of the parameters.

Non-robust inequalities (branching and cutting planes) are necessarily disabled since our pricer does not support them. While robust branching and cut-generation are technically doable, we chose to disable these two features entirely to simplify the BAP algorithm and reduce the computation time. Evaluating the column generation iterations issued at the root node of the branch-and-bound tree without additional cutting planes or branching is more than sufficient to stress the labeling algorithm.

We want to apply as much pressure on the *VRPSolver*'s pricer as possible. The

ng-sets augmentation (Pessoa et al., 2020a) is enabled, and the ng-sets maximum set size is raised as high as possible. Furthermore, the *VRPSolver* pricer's tailing-off condition was disabled. The labeling algorithm will work harder to produce dual bounds closer to the elementary bound as the ng-sets extend. As a result, the dual bound at the root node will improve significantly.

The *VRPSolver* extension's labeling algorithm can generate multiple routes per pricing iteration, providing the BPC with more diverse paths. We disabled such a feature to produce a fair comparison because our BAC framework can only output a single column per pricing iteration.

B.1 Misc parameters

These parameters control numerous aspects of *BaPCod*, such as the logging verbosity. These aren't especially significant parameters, but they can help with debugging/understanding what's going on inside the BPC algorithm's guts.

- `DEFAULTPRINTLEVEL = 0`. Controls the verbosity of the *BaPCod*'s logging system by setting it to a low level. This value strikes an acceptable balance between the amount of information emitted and the number of characters printed per second.

- `printMasterPrimalSols = 2`. By setting this parameter to 2, *BaPCod* will print the fractional solution after each column generation convergence. Useful for debugging.

B.2 Core parameters

These parameters govern the BPC algorithm's overall behavior, such as total running time, which pricer to use, enabling specific components and performing validation checks.

- `GlobalTimeLimit = 3600`. The maximum allowed solving time is set to one hour. This global timelimit affects the entire BPC algorithm from the

beginning to the resolution process at the root node until the discovery of an integral optimal solution.

- `colGenSubProbSolMode = 3`. Tells *BaPCod* to use the user-supplied custom pricing functor. In our case, the pricing functor is a stubbed implementation that wraps the original *VRPSolver* extension pricing functor and times it (see section 7.4 for more details). Instead, by setting this parameter to the value 2, we can instruct *BaPCod* to use the default generic MIP pricer for solving the pricing sub-problems.

- `ApplyPreprocessing = true`. Tells *BaPCod* to utilize pre-processing to adjust bounds and remove redundant constraints. We enabled this feature because it can potentially improve the model without affecting the pricing sub-problem.

- `PreprocessVariablesLocalBounds = false`. Ensures that *BaPCod* does not change the variable bounds of sub-problems after pre-processing. We disable this feature because our pricer does not support modifying variables' bounds.

- `CheckSpOracleFeasibility = false`. This feature is helpful for debugging and when developing custom pricing functors. When this parameter is set to `true`, an expensive MIP resolution process validates the pricer algorithm's correctness. The MIP determines whether the generated columns satisfy all the MP's constraints. It is disabled in our case.

- `CheckOracleOptimality = false`. Similar to the `CheckSpOracleFeasibility` parameter but it also verifies the optimality of the generated columns.

B.3 Column Generation parameters

These parameters control the column generation framework.

- `GenerateProperColumns = false`. When this parameter is set to `true`, *BaPCod* will throw an error if the pricer produces a non-proper column,

that is, a column that violates the sub-problem variable bounds. In our case, we set this parameter to `false` because it did not appear to cooperate with the *VRPSolver* pricing functor in our tests.

- `MaxNbOfStagesInColGenProcedure = 3`. Sets the number of stages that the pricer may use. When the stage iteration is greater than one, the pricer is permitted to use progressively lighter heuristics for column determination. The pricing problem should be solved exactly when the stage iteration reaches stage zero. When the previous stage converges, namely when it fails to find any reduced cost column, *BaPCod* automatically lowers the current stage number. When using the *VRPSolver* extension, the technical documentation recommends setting this parameter to 3. The *VRPSolver* extension implements two heuristic stages using a modified bidirectional labeling algorithm. See the *"Pricing heuristics"* and *"Column and cut generation"* sections of Sadykov et al. (2021a).

- `ReducedCostFixingThreshold = 0.0`. Setting it to zero disables reduced cost fixing.

B.4 Cut Generation parameters

These parameters govern the generation of cutting planes. We've decided to turn off cut generation entirely in the BPC framework.

- `MasterCuttingPlanesDepthLimit = -1`. When this parameter is set to `-1`, core cut generation is disabled. Core cuts are problem-independent cutting planes generated directly from the BPC algorithm. Because this parameter does not affect user-defined cuts, they must be disabled manually. Explicit usage of user-defined cuts is requested explicitly by registering the associated functors. Consequentially in our *BaPCod* model, we do not register any additional user-defined cuts. As a result, all cutting planes, including the separation of the RCC inequalities eq. (2.5), are disabled.

- `MaxNbOfCutGeneratedAtEachIter = 0`. Sets the maximum number of core cuts added per cut round to zero, effectively enforcing the core cuts generation's deactivation.

B.5 Branching parameters

These parameters govern the BPC algorithm's branching scheme and branching priorities. We've decided to turn off all forms of branching.

- `StrongBranchingPhaseOne=""`, `StrongBranchingPhaseTwo=""`, `StrongBranchingPhaseThree=""`, `StrongBranchingPhaseFour=""` These parameters govern the strong branching behavior of *BaPCod*, comprised of several phases. We disable each phase's strong branching by setting the corresponding parameter to an empty string.

- `SimplifiedStrongBranchingParameterisation = false`. When this parameter is set to true, *BaPCod* populates the parameters `StrongBranchingPhaseOne`, `StrongBranchingPhaseTwo`, `StrongBranchingPhaseThree`, `StrongBranchingPhaseFour` with sane default values. We set `SimplifiedStrongBranchingParameterisation` to `false` because populating these aforementioned parameters would effectively re-enable branching.

B.6 *VRPSolver* extension parameters

These settings affect the labeling algorithm, the size of the ng-sets, the route enumeration procedure and a few other minor aspects of the *VRPSolver* extension. Recall that the *VRPSolver* extension was introduced in Pessoa et al. (2020a), where it quickly became one of the most advanced strategies for solving routing-like problems. Refer to the scientific paper of Pessoa et al. (2020a) for additional details.

- `RCSPuseBidirectionalSearch = 2`. This parameter, known as ϕ^{bidir} in the scientific paper, instructs the RCSP pricer to use bidirectional search to solve the pricing problem.

- `RCSPapplyReducedCostFixing = 1`. This parameter, known as ϕ^{elim} in the scientific paper, instructs the pricer to use the standard bucket arc elimination discussed in Sadykov et al. (2021a).

- `RCSPmaxNumOfColsPerExactIteration = 1`, `RCSPmaxNumOfColsPerIteration = 1`. These parameters, known as $\gamma^{\text{exact}}, \gamma^{\text{heur}}$ in the scientific paper, control the number of generated columns per iteration for the heuristic and exact stages, respectively. Because our pricer can only output a single column per pricing iteration, we force the labeling algorithm to operate similarly to ensure a fair comparison.

- `RCSPallowRoutesWithSameVerticesSet = false`. This parameter tells the labeling algorithm whether it is permissible to output multiple routes that pass through the same vertices. When disabled, it allows the pricer to generate more diversified paths at the expense of increasing the pricing run time. This parameter has no effect because we're forcing the labeling algorithm to output a single route regardless.

- `RCSPmaxNumOfEnumSolutionsForMIP = 1`. This parameter, known as ω^{MIP} in the scientific paper, controls the *route enumeration* (Baldacci et al., 2008) threshold, at which the number of enumerated paths can directly trigger a solution of the SP formulation via a MIP optimizer. Because we want to measure pricing complexity, we disable the route enumeration feature by setting the corresponding parameter to `1`.

- `RCSPstopCutGenTimeThresholdInPricing = 1e21`, `RCSPhardTimeThresholdInPricing = 1e21`. These parameters, known as τ^{soft} and τ^{hard} in the scientific paper, affect the soft and hard time thresholds. These thresholds, applicable only during the exact pricing stage, modify the tailing-off condition of the pricer. If the pricer's running time exceeds one of these thresholds, column generation is preemptively interrupted

in favor of cut generation or branching. We want to measure the label setting algorithm's performance even when it struggles, so we set those parameters to high values to disable the tailing-off condition.

- `RCSPdynamicNGmode = 1`. This parameter tells *BaPCod* to scale dynamically the ng-sets size based on the fractional solution obtained by the column generation. Note that ng-set augmentation **is not** based on the solution received from the pricing problem. See Pessoa et al. (2020a) for more details.

- `RCSPinitNGneighbourhoodSize = 8`, `RCSPmaxNGneighbourhoodSize = 63`. These parameters, known as η^{init} and η^{max} in the scientific paper, limit the the ng-set size during augmentation. Raising η^{init} can improve the dual bound generated by the pricer, but it usually results in an explosion of computation times. As a result, if one wants to improve the dual bound at each branch-and-bound node, keeping η^{init} small and increasing η^{max} is the preferred approach. The ng-sets size starts at the lower value η^{init} and it is later increased (if `RCSPdynamicNGmode` is enabled) after the column generation convergence and only if the MP contains a fractional solution (see Pessoa et al., 2020a). Since our pricer produces the best dual-bound improvement possible (elementary routes), we want to stress the labeling algorithm to try as hard as possible to achieve similar results. Therefore, by setting $\eta^{\text{init}} = 8$ and $\eta^{\text{max}} = 63$ we force the labeling algorithm to produce better dual bounds towards the end of the column generation convergence. Note that, due to *BaPCod*'s implementation details $\eta^{\text{max}} \leq 63$ and thus elementary routes can be guaranteed in the column generation only for instances having less than 64 nodes.

- `RCSPmaxNGaverNeighbourhoodSize = 63`. This parameter is very similar to the η^{max} parameter, but for the average case. In our case we set this parameter to the same value of η^{max} as it was suggested from Ruslan Sadykov, a researcher who worked on *BaPCod* and *VRPSolver* extension implementations.

- `CutTailingOffThreshold = 0.0000001`, `CutTailingOffCounterThreshold = 999999`. These parameters, known as δ^{gap} and δ^{num} in the scientific paper, control the tailing off condition of cut-separation; namely, the threshold at which the BPC deems cuts ineffective and switches to lower priority (and usually more computationally expensive) cuts. Despite their name and application to cut separation, these two critical parameters also affect the column generation procedure: the augmentation of the ng-sets. As stated in the original paper, increasing the ng-sets size is a form of "cut generation" because it improves the dual bound. Ng-sets augmentation has the highest priority, followed by cut-generation and branching in this order. We want to disable the cut-tailing off condition because we always prefer to augment the ng-sets as much as possible. Setting these parameters to the above values allows us to disable the cut-tailing off condition.

Bibliography

[1] George B Dantzig and John H Ramser. “The Truck Dispatching Problem”. In: *Management science* 6.1 (1959), pp. 80–91 (cit. on pp. 1, 4, 17, 80).

[2] Merrill M Flood. “The Traveling-Salesman Problem”. In: *Operations research* 4.1 (1956), pp. 61–75 (cit. on pp. 1, 60).

[3] Michael R Garey, David S. Johnson, and R Endre Tarjan. “The Planar Hamiltonian Circuit Problem Is NP-complete”. In: *SIAM Journal on Computing* 5.4 (1976), pp. 704–714 (cit. on p. 1).

[4] Georges A Croes. “A Method for Solving Traveling-Salesman Problems”. In: *Operations research* 6.6 (1958), pp. 791–812 (cit. on pp. 1, 60).

[5] Gilbert Laporte. “The Traveling Salesman Problem: An Overview of Exact and Approximate Algorithms”. In: *European Journal of Operational Research* 59.2 (1992), pp. 231–247 (cit. on pp. 1, 58).

[6] David S Johnson and Lyle A McGeoch. “The Traveling Salesman Problem: A Case Study in Local Optimization”. In: *Local search in combinatorial optimization* 1.1 (1997), pp. 215–310 (cit. on pp. 1, 58).

[7] David L Applegate, Robert E Bixby, Vasek Chvatal, and William J Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton university press, 2006 (cit. on p. 1).

[8] Gregory Gutin and Abraham P Punnen. *The Traveling Salesman Problem and Its Variations*. Vol. 12. Springer Science & Business Media, 2006 (cit. on p. 1).

[9] Karla L Hoffman, Manfred Padberg, Giovanni Rinaldi, et al. "Traveling Salesman Problem". In: *Encyclopedia of operations research and management science* 1 (2013), pp. 1573–1578 (cit. on pp. 1, 58).

[10] Paolo Toth and Daniele Vigo. *Vehicle Routing: Problems, Methods, and Applications*. SIAM, 2014 (cit. on pp. 1–3, 26, 29, 36, 44).

[11] Paolo Toth and Daniele Vigo, eds. *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, Jan. 2002. ISBN: 978-0-89871-498-2 978-0-89871-851-5. DOI: 10.1137/1.9780898718515. URL: http://epubs.siam.org/doi/book/10.1137/1.9780898718515 (visited on 03/21/2022) (cit. on pp. 2, 15, 44).

[12] Linus Schrage. "Formulation and Structure of More Complex/Realistic Routing and Scheduling Problems". In: *Networks. An International Journal* 11.2 (1981), pp. 229–232 (cit. on p. 2).

[13] Burak Eksioglu, Arif Volkan Vural, and Arnold Reisman. "The Vehicle Routing Problem: A Taxonomic Review". In: *Computers & Industrial Engineering* 57.4 (Nov. 2009), pp. 1472–1483. ISSN: 03608352. DOI: 10.1016/j.cie.2009.05.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360835209001405 (visited on 03/21/2022) (cit. on p. 3).

[14] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. "The Vehicle Routing Problem: State of the Art Classification and Review". In: *Computers & Industrial Engineering* 99 (Sept. 2016), pp. 300–313. ISSN: 03608352. DOI: 10.1016/j.cie.2015.12.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0360835215004775 (visited on 03/21/2022) (cit. on p. 3).

[15] Geoff Clarke and John W Wright. "Scheduling of Vehicles from a Central Depot to a Number of Delivery Points". In: *Operations research* 12.4 (1964), pp. 568–581 (cit. on pp. 3, 17).

[16] Martin Desrochers and TW Verhoog. "A Matching Based Savings Algorithm for the Vehicle Routing Problem". In: *Cahiers du GERAD* (1989) (cit. on p. 3).

[17] Heinrich Paessens. "The Savings Algorithm for the Vehicle Routing Problem". In: *European Journal of Operational Research* 34.3 (1988), pp. 336–344 (cit. on p. 3).

[18] Brian A Foster and David M Ryan. "An Integer Programming Approach to the Vehicle Scheduling Problem". In: *Journal of the Operational Research Society* 27.2 (1976), pp. 367–384 (cit. on p. 3).

[19] Michel Gendreau, Alain Hertz, and Gilbert Laporte. "A Tabu Search Heuristic for the Vehicle Routing Problem". In: *Management science* 40.10 (1994), pp. 1276–1290 (cit. on p. 3).

[20] Jean-François Cordeau and Mirko Maischberger. "A Parallel Iterated Tabu Search Heuristic for Vehicle Routing Problems". In: *Computers & Operations Research* 39.9 (2012), pp. 2033–2050 (cit. on p. 3).

[21] Paolo Toth and Daniele Vigo. "The Granular Tabu Search and Its Application to the Vehicle-Routing Problem". In: *Informs Journal on computing* 15.4 (2003), pp. 333–346 (cit. on p. 3).

[22] Feiyue Li, Bruce Golden, and Edward Wasil. "Very Large-Scale Vehicle Routing: New Test Problems, Algorithms, and Results". In: *Computers & Operations Research* 32.5 (2005), pp. 1165–1179 (cit. on p. 3).

[23] David Pisinger and Stefan Ropke. "A General Heuristic for Vehicle Routing Problems". In: *Computers & operations research* 34.8 (2007), pp. 2403–2435 (cit. on p. 3).

[24] Jari Kytöjoki, Teemu Nuortio, Olli Bräysy, and Michel Gendreau. "An Efficient Variable Neighborhood Search Heuristic for Very Large Scale Vehicle Routing Problems". In: *Computers & operations research* 34.9 (2007), pp. 2743–2757 (cit. on p. 3).

[25] Yuichi Nagata and Olli Bräysy. "Edge Assembly-Based Memetic Algorithm for the Capacitated Vehicle Routing Problem". In: *Networks* 54.4 (Dec. 2009), pp. 205–215. ISSN: 00283045. DOI: 10.1002/net.20333. URL: https://onlinelibrary.wiley.com/doi/10.1002/net.20333 (visited on 03/25/2022) (cit. on p. 3).

[26] Thibaut Vidal, Teodor Gabriel Crainic, Michel Gendreau, Nadia Lahrichi, and Walter Rei. "A Hybrid Genetic Algorithm for Multidepot and Periodic Vehicle Routing Problems". In: *Operations Research* 60.3 (June 2012), pp. 611–624. ISSN: 0030-364X, 1526-5463. DOI: 10.1287/opre.1120.1048. URL: http://pubsonline.informs.org/doi/abs/10.1287/opre.1120.1048 (visited on 03/25/2022) (cit. on p. 3).

[27] Anand Subramanian, Eduardo Uchoa, and Luiz Satoru Ochi. "A Hybrid Algorithm for a Class of Vehicle Routing Problems". In: *Computers & Operations Research* 40.10 (Oct. 2013), pp. 2519–2531. ISSN: 03050548. DOI: 10.1016/j.cor.2013.01.013. URL: https://linkinghub.elsevier.com/retrieve/pii/S030505481300021X (visited on 03/25/2022) (cit. on p. 3).

[28] Bruce L Golden, Edward A Wasil, James P Kelly, I Chao, et al. "The Impact of Metaheuristics on Solving the Vehicle Routing Problem: Algorithms, Problem Sets, and Computational Results". In: *Fleet Management and Logistics*. Springer, 1998, pp. 33–56 (cit. on p. 3).

[29] Michel Gendreau, Gilbert Laporte, and Jean-Yves Potvin. "Metaheuristics for the Capacitated VRP". In: *The Vehicle Routing Problem*. SIAM, 2002, pp. 129–154 (cit. on p. 3).

[30] Michel Gendreau, Jean-Yves Potvin, Olli Bräumlaysy, Geir Hasle, and Arne Løkketangen. "Metaheuristics for the Vehicle Routing Problem and Its Extensions: A Categorized Bibliography". In: *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008, pp. 143–169 (cit. on p. 3).

[31] Gilbert Laporte, Stefan Ropke, and Thibaut Vidal. "Chapter 4: Heuristics for the Vehicle Routing Problem". In: *Vehicle Routing: Problems, Methods, and Applications, Second Edition*. SIAM, 2014, pp. 87–116 (cit. on p. 3).

[32] Raafat Elshaer and Hadeer Awad. "A Taxonomic Review of Metaheuristic Algorithms for Solving the Vehicle Routing Problem and Its Variants". In: *Computers & Industrial Engineering* 140 (2020), p. 106242 (cit. on p. 3).

[33] Jean-François Cordeau, Gilbert Laporte, Martin WP Savelsbergh, and Daniele Vigo. "Vehicle Routing". In: *Handbooks in operations research and management science* 14 (2007), pp. 367–428 (cit. on p. 3).

[34] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. "Recent Exact Algorithms for Solving the Vehicle Routing Problem under Capacity and Time Window Constraints". In: *European Journal of Operational Research* 218.1 (2012), pp. 1–6 (cit. on p. 3).

[35] Jose Caceres-Cruz, Pol Arias, Daniel Guimarans, Daniel Riera, and Angel A. Juan. "Rich Vehicle Routing Problem: Survey". In: *ACM Computing Surveys* 47.2 (Jan. 8, 2015), pp. 1–28. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/2666003. URL: https://dl.acm.org/doi/10.1145/2666003 (visited on 03/21/2022) (cit. on p. 3).

[36] Luciano Costa, Claudio Contardo, and Guy Desaulniers. "Exact Branch-Price-and-Cut Algorithms for Vehicle Routing". In: *Transportation Science* 53.4 (2019), pp. 946–985. ISSN: 15265447. DOI: 10.1287/trsc.2018.0878 (cit. on pp. 3, 20, 32).

[37] Nicos Christofides and Samuel Eilon. "An Algorithm for the Vehicle-Dispatching Problem". In: *Journal of the Operational Research Society* 20.3 (1969), pp. 309–318 (cit. on pp. 4, 17, 27, 80).

[38] TJ Gaskell. "Bases for Vehicle Fleet Scheduling". In: *Journal of the Operational Research Society* 18.3 (1967), pp. 281–295 (cit. on pp. 4, 80).

[39] Billy E Gillett and Leland R Miller. "A Heuristic Algorithm for the Vehicle-Dispatch Problem". In: *Operations research* 22.2 (1974), pp. 340–349 (cit. on pp. 4, 80).

[40] Nicos Christofides. "The Vehicle Routing Problem". In: *Combinatorial optimization* (1979) (cit. on pp. 4, 11).

[41] Marshall L. Fisher. "Optimal Solution of Vehicle Routing Problems Using Minimum K-Trees". In: *Operations Research* 42.4 (1994), pp. 626–642. JSTOR: 171617 (cit. on pp. 4, 17, 18, 27, 80, 81).

[42] P Augerat, J M Beleaguer, E Benavent, A Corberan, D Naddef, and G Rinaldi. "Computational Results with a Branch and Cut Code for the Capacitated Vehicle Routing Problem". In: (1995), p. 33 (cit. on pp. 4, 12, 18, 80, 82–84).

[43] Eduardo Uchoa, Diego Pecin, Artur Pessoa, Marcus Poggi, Thibaut Vidal, and Anand Subramanian. “New Benchmark Instances for the Capacitated Vehicle Routing Problem”. In: *European Journal of Operational Research* 257.3 (2017), pp. 845–858 (cit. on pp. 4, 20, 80–84).

[44] Gabriel Gutiérrez-Jarpa, Guy Desaulniers, Gilbert Laporte, and Vladimir Marianov. “A Branch-and-Price Algorithm for the Vehicle Routing Problem with Deliveries, Selective Pickups and Time Windows”. In: *European Journal of Operational Research* 206.2 (2010), pp. 341–349 (cit. on p. 5).

[45] Claudia Archetti, Mathieu Bouchard, and Guy Desaulniers. “Enhanced Branch and Price and Cut for Vehicle Routing with Split Deliveries and Time Windows”. In: *Transportation Science* 45.3 (2011), pp. 285–298 (cit. on pp. 5, 36).

[46] Andrea Bettinelli, Alberto Ceselli, and Giovanni Righini. “A Branch-and-Cut-and-Price Algorithm for the Multi-Depot Heterogeneous Vehicle Routing Problem with Time Windows”. In: *Transportation Research Part C: Emerging Technologies* 19.5 (2011), pp. 723–740 (cit. on p. 5).

[47] Claudio Contardo and Rafael Martinelli. “A New Exact Algorithm for the Multi-Depot Vehicle Routing Problem under Capacity and Route Length Constraints”. In: *Discrete Optimization* 12 (2014), pp. 129–146 (cit. on pp. 5, 19, 26, 31, 32, 36).

[48] Claudio Contardo, Guy Desaulniers, and François Lessard. “Reaching the Elementary Lower Bound in the Vehicle Routing Problem with Time Windows”. In: *Networks. An International Journal* 65.1 (2015), pp. 88–99 (cit. on pp. 5, 35).

[49] Diego Pecin, Claudio Contardo, Guy Desaulniers, and Eduardo Uchoa. “New Enhancements for the Exact Solution of the Vehicle Routing Problem with Time Windows”. In: *INFORMS Journal on Computing* 29.3 (2017), pp. 489–502 (cit. on pp. 5, 20, 26, 29).

[50] Diego Pecin, Artur Pessoa, Marcus Poggi, and Eduardo Uchoa. “Improved Branch-Cut-and-Price for Capacitated Vehicle Routing”. In: *Mathematical Programming Computation* 9.1 (2017), pp. 61–100. ISSN:

18672957. DOI: 10.1007/s12532-016-0108-8 (cit. on pp. 5, 20, 26, 28, 31, 32, 85).

[51] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. "A Generic Exact Solver for Vehicle Routing and Related Problems". In: *Mathematical Programming* 183.1-2 (2020), pp. 483–523. ISSN: 14364646. DOI: 10.1007/s10107-020-01523-z (cit. on pp. 5, 20, 79, 83–85, 88, 91, 93, 97, 99, 101, 103, 105, 115, 116, 119, 121).

[52] Mads Kehlet Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. "A Branch-and-Cut Algorithm for the Capacitated Profitable Tour Problem". In: *Discrete Optimization* 14 (2014), pp. 78–96. ISSN: 15725286. DOI: 10.1016/j.disopt.2014.08.001. URL: http://dx.doi.org/10.1016/j.disopt.2014.08.001 (cit. on pp. 5, 40, 41, 48, 49, 51–53, 62, 63, 107).

[53] G. Laporte and Y. Nobert. "A Branch and Bound Algorithm for the Capacitated Vehicle Routing Problem". In: *OR Spektrum* 5.2 (1983), pp. 77–85. ISSN: 01716468. DOI: 10.1007/BF01720015 (cit. on pp. 7, 10, 17, 30, 51, 85).

[54] Gilbert Laporte, Yves Nobert, and Martin Desrochers. "Optimal Routing under Capacity and Distance Restrictions". In: *Operations research* 33.5 (1985), pp. 1050–1073 (cit. on pp. 7, 10, 11, 17).

[55] Gilbert Laporte, Hélène Mercure, and Yves Nobert. "An Exact Algorithm for the Asymmetrical Capacitated Vehicle Routing Problem". In: *Networks* 16.1 (1986), pp. 33–46. ISSN: 10970037. DOI: 10.1002/net.3230160104 (cit. on pp. 7, 10, 17).

[56] Bruce L Golden, Thomas L Magnanti, and Hien Q Nguyen. "Implementing Vehicle Routing Algorithms". In: *Networks. An International Journal* 7.2 (1977), pp. 113–148 (cit. on pp. 7, 12).

[57] Michel L Balinski and Richard E Quandt. "On an Integer Program for a Delivery Problem". In: *Operations research* 12.2 (1964), pp. 300–304 (cit. on pp. 7, 14).

[58] Artur Pessoa and Eduardo Uchoa. "Solving Bin Packing Problems Using VRPSolver Models". In: *Operations Research Forum*. Vol. 2. Springer, 2020, pp. 1–25 (cit. on pp. 8, 26, 85).

[59] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. "An Exact Algorithm for the Capacitated Vehicle Routing Problem Based on a Two-Commodity Network Flow Formulation". In: *Operations Research* 52.5 (2004), pp. 723–738. ISSN: 0030364X. DOI: 10.1287/opre.1040.0111 (cit. on pp. 8, 18).

[60] Silvano Martello and Paolo Toth. "Lower Bounds and Reduction Procedures for the Bin Packing Problem". In: *Discrete applied mathematics* 28.1 (1990), pp. 59–70 (cit. on p. 9).

[61] Silvano Martello and Paolo Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990 (cit. on pp. 9, 11).

[62] Gerard Cornuejols and Farid Harche. "Polyhedral Study of the Capacitated Vehicle Routing Problem". In: *Mathematical Programming* 60.1 (1993), pp. 21–52 (cit. on pp. 11, 12).

[63] Clair E Miller, Albert W Tucker, and Richard A Zemlin. "Integer Programming Formulation of Traveling Salesman Problems". In: *Journal of the ACM (JACM)* 7.4 (1960), pp. 326–329 (cit. on pp. 11, 109).

[64] Martin Desrochers and Gilbert Laporte. "Improvements and Extensions to the Miller-Tucker-Zemlin Subtour Elimination Constraints". In: *Operations Research Letters* 10.1 (1991), pp. 27–36 (cit. on p. 11).

[65] Martin Grötschel and Manfred W Padberg. "Partial Linear Characterizations of the Asymmetric Travelling Salesman Polytope". In: *Mathematical Programming* 8.1 (1975), pp. 378–381 (cit. on p. 12).

[66] Vincente Campos, Angel Corberan, and Enrique Mota. "Polyhedral Results for a Vehicle Routing Problem". In: *European Journal of Operational Research* 52.1 (1991), pp. 75–85 (cit. on p. 12).

[67] Philippe Augerat, José-Manuel Belenguer, Enrique Benavent, Angel Corbéran, and Denis Naddef. "Separating Capacity Constraints in the CVRP Using Tabu Search". In: *European Journal of Operational Research* 106.2-3 (1998), pp. 546–557 (cit. on pp. 12, 28).

[68] T.K. Ralphs, L. Kopman, W.R. Pulleyblank, and L.E. Trotter. "On the Capacitated Vehicle Routing Problem". In: *Mathematical Programming* 94.2-3 (Jan. 1, 2003), pp. 343–359. ISSN: 0025-5610, 1436-4646. DOI:

10.1007/s10107-002-0323-0. URL: http://link.springer.com/10.1007/s10107-002-0323-0 (visited on 03/25/2022) (cit. on pp. 12, 18, 61).

[69] Jens Lysgaard. "CVRPSEP: A Package of Separation Routines for the Capacitated Vehicle Routing Problem". In: (2003) (cit. on pp. 12, 31).

[70] Marshall L Fisher and Ramchandran Jaikumar. "A Generalized Assignment Heuristic for Vehicle Routing". In: *Networks. An International Journal* 11.2 (1981), pp. 109–124 (cit. on p. 13).

[71] George B Dantzig and Philip Wolfe. "Decomposition Principle for Linear Programs". In: *Operations research* 8.1 (1960), pp. 101–111 (cit. on pp. 14, 82).

[72] Guy Desaulniers, Jacques Desrosiers, Irina loachim, Marius M. Solomon, François Soumis, and Daniel Villeneuve. "A Unified Framework for Deterministic Time Constrained Vehicle Routing and Crew Scheduling Problems". In: *Fleet Management and Logistics*. Ed. by Teodor Gabriel Crainic and Gilbert Laporte. Boston, MA: Springer US, 1998, pp. 57–93. ISBN: 978-1-4613-7637-8 978-1-4615-5755-5. DOI: 10.1007/978-1-4615-5755-5_3. URL: http://link.springer.com/10.1007/978-1-4615-5755-5_3 (visited on 11/02/2021) (cit. on p. 14).

[73] François Vanderbeck. "Implementing Mixed Integer Column Generation". In: *Column Generation*. Springer, 2005, pp. 331–358 (cit. on pp. 14, 21, 37).

[74] Julien Bramel and David Simchi-Levi. "On the Effectiveness of Set Covering Formulations for the Vehicle Routing Problem with Time Windows". In: *Operations Research* 45.2 (1997), pp. 295–301 (cit. on p. 16).

[75] John Franklin Pierce. "Direct Search Algorithms for Truck-Dispatching Problems". In: *Transportation Research* 3.1 (1969), pp. 1–42 (cit. on p. 17).

[76] N. Christofides, A. Mingozzi, and P. Toth. "Exact Algorithms for the Vehicle Routing Problem, Based on Spanning Tree and Shortest Path Relaxations". In: *Mathematical Programming* 20.1 (1981), pp. 255–282. ISSN: 00255610. DOI: 10.1007/BF01589353 (cit. on pp. 17, 33).

[77] Donald L Miller. “A Matching Based Exact Algorithm for Capacitated Vehicle Routing Problems”. In: *ORSA Journal on Computing* 7.1 (1995), pp. 1–9 (cit. on pp. 17, 27).

[78] Matteo Fischetti, Paolo Toth, and Daniele Vigo. “A Branch-and-Bound Algorithm for the Capacitated Vehicle Routing Problem on Directed Graphs”. In: *Operations Research* 42.5 (1994), pp. 846–859 (cit. on p. 17).

[79] Eleni Hadjiconstantinou, Nicos Christofides, and Aristide Mingozzi. “A New Exact Algorithm for the Vehicle Routing Problem Based Onq-Paths Andk-Shortest Paths Relaxations”. In: *Annals of Operations Research* 61.1 (1995), pp. 21–43 (cit. on p. 17).

[80] Philippe Augerat. “Approche Polyèdrale Du Problème de Tournées de Véhicules”. Institut National Polytechnique de Grenoble-INPG, 1995 (cit. on pp. 17, 29).

[81] Jens Lysgaard, Adam N. Letchford, and Richard W. Eglese. “A New Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem”. In: *Mathematical Programming* 100.2 (2004), pp. 423–445. ISSN: 00255610. DOI: 10.1007/s10107-003-0481-8 (cit. on pp. 17, 18).

[82] J. R. Araque G, G. Kudva, T. L. Morin, and J. F. Pekny. “A Branch-and-Cut Algorithm for Vehicle Routing Problems”. In: *Annals of Operations Research* 50.1 (Dec. 1994), pp. 37–59. ISSN: 0254-5330, 1572-9338. DOI: 10.1007/BF02085634. URL: http://link.springer.com/10.1007/BF02085634 (visited on 03/25/2022) (cit. on pp. 18, 30).

[83] NR Achuthan, L Caccetta, and SP Hill. “A New Subtour Elimination Constraint for the Vehicle Routing Problem”. In: *European Journal of Operational Research* 91.3 (1996), pp. 573–586 (cit. on p. 18).

[84] Ulrich Blasum and Winfried Hochstattler. “Application of the Branch and Cut Method to the Vehicle Routing Problem”. In: (2000), p. 22 (cit. on p. 18).

[85] N. R. Achuthan, L. Caccetta, and S. P. Hill. “An Improved Branch-and-Cut Algorithm for the Capacitated Vehicle Routing Problem”. In: *Transportation Science* 37.2 (May 2003), pp. 153–169. ISSN: 0041-1655, 1526-5447. DOI: 10.1287/trsc.37.2.153.15243. URL:

http://pubsonline.informs.org/doi/abs/10.1287/trsc.37.2.153.15243 (visited on 03/25/2022) (cit. on p. 18).

[86] Jacques Desrosiers, François Soumis, and Martin Desrochers. "Routing with Time Windows by Column Generation". In: *Networks. An International Journal* 14.4 (1984), pp. 545–565 (cit. on p. 18).

[87] Yogesh Agarwal, Kamlesh Mathur, and Harvey M Salkin. "A Set-Partitioning-Based Exact Algorithm for the Vehicle Routing Problem". In: *Networks. An International Journal* 19.7 (1989), pp. 731–749 (cit. on p. 18).

[88] Martin Desrochers, Jacques Desrosiers, and Marius Solomon. "A New Optimization Algorithm for the Vehicle Routing Problem with Time Windows". In: *Operations research* 40.2 (1992), pp. 342–354 (cit. on pp. 18, 33, 40, 43, 56, 107, 108).

[89] Niklas Kohl, Jacques Desrosiers, Oli BG Madsen, Marius M Solomon, and Francois Soumis. "2-Path Cuts for the Vehicle Routing Problem with Time Windows". In: *Transportation Science* 33.1 (1999), pp. 101–116 (cit. on p. 18).

[90] Guy Desaulniers, François Lessard, and Ahmed Hadjar. "Tabu Search, Partial Elementarity, and Generalized k-Path Inequalities for the Vehicle Routing Problem with Time Windows". In: *Transportation Science* 42.3 (2008), pp. 387–404 (cit. on pp. 18, 31, 35, 36).

[91] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa, and Renato F. Werneck. "Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem". In: *Mathematical Programming* 106.3 (May 2006), pp. 491–511. ISSN: 0025-5610, 1436-4646. DOI: 10.1007/s10107-005-0644-x. URL: http://link.springer.com/10.1007/s10107-005-0644-x (visited on 03/24/2022) (cit. on pp. 18, 19, 25, 26, 29, 33, 35, 36).

[92] Roberto Baldacci, Nicos Christofides, and Aristide Mingozzi. "An Exact Algorithm for the Vehicle Routing Problem Based on the Set Partitioning Formulation with Additional Cuts". In: *Mathematical Programming* 115.2 (2008), pp. 351–385. ISSN: 00255610. DOI: 10.1007/s10107-007-0178-5 (cit. on pp. 19, 26, 31, 120).

[93] Artur Pessoa, Marcus Poggi De Aragao, and Eduardo Uchoa. “Robust Branch-Cut-and-Price Algorithms for Vehicle Routing Problems”. In: *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008, pp. 297–325 (cit. on pp. 19, 26, 31).

[94] Claudio Contardo, Jean-François Cordeau, and Bernard Gendron. *A Branch-and-Cut-and-Price Algorithm for the Capacitated Location-Routing Problem*. CIRRELT, 2011 (cit. on pp. 19, 32, 33).

[95] Roberto Baldacci, Aristide Mingozzi, and Roberto Roberti. “New Route Relaxation and Pricing Strategies for the Vehicle Routing Problem”. In: *Operations Research* 59.5 (2011), pp. 1269–1283. ISSN: 0030364X. DOI: 10.1287/opre.1110.0975 (cit. on pp. 19, 26, 31, 32, 35, 40, 56, 84, 108).

[96] Stefan Røpke. “Branching Decisions in Branch-and-Cut-and-Price Algorithms for Vehicle Routing Problems”. In: *Presentation in Column Generation* 2012 (2012) (cit. on pp. 19, 25).

[97] Rafael Martinelli, Diego Pecin, and Marcus Poggi. “Efficient Elementary and Restricted Non-Elementary Route Pricing”. In: *European Journal of Operational Research* 239.1 (2014), pp. 102–111 (cit. on pp. 19, 108).

[98] Artur Pessoa, Ruslan Sadykov, and Eduardo Uchoa. “Enhanced Branch-Cut-and-Price Algorithm for Heterogeneous Fleet Vehicle Routing Problems”. In: *European Journal of Operational Research* 270.2 (Oct. 2018), pp. 530–543. ISSN: 03772217. DOI: 10.1016/j.ejor.2018.04.009. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377221718303126 (visited on 03/25/2022) (cit. on p. 20).

[99] Marco E Lübbecke and Jacques Desrosiers. “Selected Topics in Column Generation”. In: *Operations research* 53.6 (2005), pp. 1007–1023 (cit. on pp. 21, 23).

[100] Jacques Desrosiers and Marco E. Lübbecke. “Branch-Price-and-Cut Algorithms”. In: *Wiley Encyclopedia of Operations Research and Management Science* (2011), pp. 109–131. DOI: 10.1002/9780470400531.eorms0118 (cit. on p. 21).

[101] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh, and Pamela H Vance. “Branch-and-Price: Column Genera-

tion for Solving Huge Integer Programs". In: *Operations research* 46.3 (1998), pp. 316–329 (cit. on p. 21).

[102] Jacques Desrosiers and Marco E Lübbecke. "A Primer in Column Generation". In: *Column Generation*. Springer, 2005, pp. 1–32 (cit. on pp. 21, 24).

[103] Dominique Feillet. "A Tutorial on Column Generation and Branch-and-Price for Vehicle Routing Problems". In: *4or* 8.4 (2010), pp. 407–424 (cit. on pp. 21, 37).

[104] Ailsa H Land and Alison G Doig. "An Automatic Method for Solving Discrete Programming Problems". In: *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 105–132 (cit. on pp. 21, 112).

[105] Giovanni Righini and Matteo Salani. "New Dynamic Programming Algorithms for the Resource Constrained Elementary Shortest Path Problem". In: *Networks* 51.3 (2008), pp. 155–170. ISSN: 00283045. DOI: 10.1002/net.20212 (cit. on pp. 21, 34, 40, 108).

[106] P. C. Gilmore and R. E. Gomory. "A Linear Programming Approach to the Cutting-Stock Problem". In: *Operations Research* 9.6 (1961), pp. 849–859. JSTOR: 167051 (cit. on p. 21).

[107] Ruslan Sadykov. "Modern Branch-Cut-and-Price". PhD thesis. Université de Bordeaux, 2019 (cit. on pp. 21, 25, 85).

[108] George B Dantzig, Alex Orden, Philip Wolfe, et al. "The Generalized Simplex Method for Minimizing a Linear Form under Linear Inequality Restraints". In: *Pacific Journal of Mathematics* 5.2 (1955), pp. 183–195 (cit. on p. 23).

[109] Guy Desaulniers. *Branch-Price-and-Cut for Vehicle Routing*. 2018 (cit. on p. 24).

[110] Artur Pessoa, Eduardo Uchoa, and Marcus Poggi De Aragão. "A Robust Branch-Cut-and-Price Algorithm for the Heterogeneous Fleet Vehicle Routing Problem". In: *Networks* 54.4 (2009), pp. 167–177. ISSN: 00283045. DOI: 10.1002/net.20330 (cit. on pp. 26, 31).

[111] M Poggi de Aragao and Eduardo Uchoa. "Integer Program Reformulation for Robust Branch-and-Cut-and-Price Algorithms". In: *Mathemati-*

cal Program in Rio: A Conference in Honour of Nelson Maculan. Citeseer, 2003, pp. 56–61 (cit. on p. 26).

[112] Guy Desaulniers, Jacques Desrosiers, and Simon Spoorendonk. "Cutting Planes for Branch-and-Price Algorithms". In: *Networks. An International Journal* 58.4 (2011), pp. 301–310 (cit. on pp. 27, 29).

[113] François Vanderbeck and Laurence A Wolsey. "Reformulation and Decomposition of Integer Programs". In: *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 431–502 (cit. on p. 27).

[114] Daniel Villeneuve and Guy Desaulniers. "The Shortest Path Problem with Forbidden Paths". In: *European Journal of Operational Research* 165.1 (2005), pp. 97–107 (cit. on p. 27).

[115] David M Ryan and Brian A Foster. "An Integer Programming Approach to Scheduling". In: *Computer scheduling of public transport urban passenger vehicle and crew scheduling* (1981), pp. 269–280 (cit. on pp. 28, 85).

[116] Sylvie Gélinas, Martin Desrochers, Jacques Desrosiers, and Marius M Solomon. "A New Branching Strategy for Time Constrained Routing Problems with Application to Backhauling". In: *Annals of Operations Research* 61.1 (1995), pp. 91–109 (cit. on p. 28).

[117] Diego Pecin, Artur Pessoa, Marcus Poggi, Eduardo Uchoa, and Haroldo Santos. "Limited Memory Rank-1 Cuts for Vehicle Routing Problems". In: *Operations Research Letters* 45.3 (2017), pp. 206–209 (cit. on pp. 29, 32).

[118] Ahmed Hadjar, Odile Marcotte, and François Soumis. "A Branch-and-Cut Algorithm for the Multiple Depot Vehicle Scheduling Problem". In: *Operations Research* 54.1 (2006), pp. 130–149 (cit. on p. 29).

[119] Stefan Irnich, Guy Desaulniers, Jacques Desrosiers, and Ahmed Hadjar. "Path-Reduced Costs for Eliminating Arcs in Routing and Scheduling". In: *INFORMS Journal on Computing* 22.2 (2010), pp. 297–313. ISSN: 10919856. DOI: 10.1287/ijoc.1090.0341 (cit. on p. 29).

[120] Denis Naddef and Giovanni Rinaldi. "The Graphical Relaxation: A New Framework for the Symmetric Traveling Salesman Polytope". In: *Mathematical Programming* 58.1 (1993), pp. 53–88 (cit. on p. 29).

[121] Václav Chvátal. "Edmonds Polytopes and Weakly Hamiltonian Graphs". In: *Mathematical programming* 5.1 (1973), pp. 29–40 (cit. on pp. 29, 31, 111).

[122] Martin Grötschel and Manfred W Padberg. "On the Symmetric Travelling Salesman Problem I: Inequalities". In: *Mathematical Programming* 16.1 (1979), pp. 265–280 (cit. on p. 29).

[123] Jésus Rafael Araque, Leslie A Hall, and Thomas L Magnanti. "Capacitated Trees, Capacitated Routing, and Associated Polyhedra". In: (1990) (cit. on p. 30).

[124] Luis Gouveia. "A Result on Projection for the Vehicle Routing Problem". In: *European Journal of Operational Research* 85.3 (Sept. 1995), pp. 610–624. ISSN: 03772217. DOI: 10.1016/0377-2217(94)00025-8. URL: https://linkinghub.elsevier.com/retrieve/pii/0377221794000258 (visited on 02/01/2022) (cit. on pp. 30, 52).

[125] NR Achuthan, L Caccetta, and SP Hill. "Capacitated Vehicle Routing Problem: Some New Cutting Planes". In: *Asia-Pacific Journal of Operational Research* 15.1 (1998), p. 109 (cit. on p. 30).

[126] Adam N Letchford, Richard W Eglese, and Jens Lysgaard. "Multistars, Partial Multistars and the Capacitated Vehicle Routing Problem". In: *Mathematical Programming* 94.1 (2002), pp. 21–40 (cit. on pp. 30, 52).

[127] Adam N. Letchford and Juan José Salazar-González. "Projection Results for Vehicle Routing". In: *Mathematical Programming* 105.2-3 (2006), pp. 251–274. ISSN: 14364646. DOI: 10.1007/s10107-005-0652-x (cit. on pp. 30, 52, 53, 63).

[128] Mads Jepsen, Bjørn Petersen, Simon Spoorendonk, and David Pisinger. "Subset-Row Inequalities Applied to the Vehicle-Routing Problem with Time Windows". In: *Operations Research* 56.2 (2008), pp. 497–511 (cit. on p. 31).

[129] Pisinger Jepsen. "A Branch-and-Cut Algorithm for the Capacitated Profitable Tour Problem". In: *Branch-and-cut and Branch-and-Cut-and-Price Algorithms for Solving Vehicle Routing Problems* (2011), p. 99 (cit. on pp. 31, 40, 48).

[130] Bjørn Petersen, David Pisinger, and Simon Spoorendonk. "Chvátal-Gomory Rank-1 Cuts Used in a Dantzig-Wolfe Decomposition of the Vehicle Routing Problem with Time Windows". In: *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, 2008, pp. 397–419 (cit. on p. 31).

[131] Simon Spoorendonk and Guy Desaulniers. "Clique Inequalities Applied to the Vehicle Routing Problem with Time Windows". In: *INFOR: Information Systems and Operational Research* 48.1 (2010), pp. 53–67 (cit. on p. 32).

[132] Moshe Dror. "Note on the Complexity of the Shortest Path Models for Column Generation in VRPTW". In: *Operations Research* 42.5 (Oct. 1994), pp. 977–978. ISSN: 0030-364X, 1526-5463. DOI: 10.1287/opre.42.5.977. URL: http://pubsonline.informs.org/doi/abs/10.1287/opre.42.5.977 (visited on 03/24/2022) (cit. on pp. 33, 43).

[133] Martin Desrochers and François Soumis. "A Generalized Permanent Labelling Algorithm for the Shortest Path Problem with Time Windows". In: *INFOR: Information Systems and Operational Research* 26.3 (1988), pp. 191–212 (cit. on p. 33).

[134] Stefan Irnich and Guy Desaulniers. "Shortest Path Problems with Resource Constraints". In: *Column Generation*. Springer, 2005, pp. 33–65 (cit. on pp. 33, 37, 39).

[135] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. "An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints: Application to Some Vehicle Routing Problems". In: *Networks* 44.3 (2004), pp. 216–229. ISSN: 00283045. DOI: 10.1002/net.20033 (cit. on pp. 33–35, 40, 43, 56, 107).

[136] Richard Bellman. "On a Routing Problem". In: *Quarterly of applied mathematics* 16.1 (1958), pp. 87–90 (cit. on p. 34).

[137] Lester R Ford Jr. *Network Flow Theory*. Rand Corp Santa Monica Ca, 1956 (cit. on p. 34).

[138] M Jepsen, B Petersen, and S Spoorendonk. *A Branch-and-Cut Algorithm for the Elementary Shortest Path Problem with a Capacity Con-*

straint. 08. Citeseer, 2008. URL: http://www.diku.dk/forskning/Publikationer/tekniske%7B%5C_%7Drapporter/2008/08-01.pdf (cit. on pp. 34, 40, 41, 43, 47, 61, 110).

[139] Giovanni Righini and Matteo Salani. "Symmetry Helps: Bounded Bi-Directional Dynamic Programming for the Elementary Shortest Path Problem with Resource Constraints". In: *Discrete Optimization* 3.3 (2006), pp. 255–273. ISSN: 15725286. DOI: 10.1016/j.disopt.2006.05.007 (cit. on pp. 34, 40).

[140] Natashia Boland, John Dethridge, and Irina Dumitrescu. "Accelerated Label Setting Algorithms for the Elementary Resource Constrained Shortest Path Problem". In: *Operations Research Letters* 34.1 (2006), pp. 58–68 (cit. on pp. 34, 40, 108).

[141] Alain Chabrier. "Vehicle Routing Problem with Elementary Shortest Path Based Column Generation". In: *Computers and Operations Research* 33.10 (2006), pp. 2972–2990. ISSN: 03050548. DOI: 10.1016/j.cor.2005.02.029 (cit. on p. 35).

[142] Stefan Irnich and Daniel Villeneuve. "The Shortest-Path Problem with Resource Constraints and k-Cycle Elimination for $k \geq 3$". In: *INFORMS Journal on Computing* 18.3 (2006), pp. 391–406. ISSN: 08991499. DOI: 10.1287/ijoc.1040.0117 (cit. on p. 35).

[143] Roberto Roberti and Aristide Mingozzi. "Dynamic Ng-Path Relaxation for the Delivery Man Problem". In: *Transportation Science* 48.3 (2014), pp. 413–424 (cit. on p. 35).

[144] Leonardo Lozano and Andrés L Medaglia. "On an Exact Method for the Constrained Shortest Path Problem". In: *Computers & Operations Research* 40.1 (2013), pp. 378–384 (cit. on pp. 36, 40).

[145] Leonardo Lozano, Daniel Duque, and Andrés L Medaglia. "An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints". In: *Transportation Science* 50.1 (2016), pp. 348–357 (cit. on pp. 36, 40).

[146] Guy Desaulniers, Diego Pecin, and Claudio Contardo. "Selective Pricing in Branch-Price-and-Cut Algorithms for Vehicle Routing". In: *EURO*

Journal on Transportation and Logistics 8.2 (2019), pp. 147–168 (cit. on p. 36).

[147] Olivier Du Merle, Daniel Villeneuve, Jacques Desrosiers, and Pierre Hansen. "Stabilized Column Generation". In: *Discrete Mathematics* 194.1-3 (1999), pp. 229–237 (cit. on p. 37).

[148] Louis-Martin Rousseau, Michel Gendreau, and Dominique Feillet. "Interior Point Stabilization for Column Generation". In: *Operations Research Letters* 35.5 (2007), pp. 660–668 (cit. on p. 37).

[149] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and Francois Vanderbeck. "In-out Separation and Column Generation Stabilization by Dual Price Smoothing". In: *International Symposium on Experimental Algorithms*. Springer. 2013, pp. 354–365 (cit. on p. 37).

[150] Artur Pessoa, Ruslan Sadykov, Eduardo Uchoa, and François Vanderbeck. "Automation and Combination of Linear-Programming Based Stabilization Techniques in Column Generation". In: *INFORMS Journal on Computing* 30.2 (2018), pp. 339–360 (cit. on pp. 37, 82).

[151] Luigi Di Puglia Pugliese and Francesca Guerriero. "Solution Approaches for the Elementary Shortest Path Problem". In: (2010) (cit. on pp. 37, 40).

[152] Luigi Di Puglia Pugliese and Francesca Guerriero. "A Survey of Resource Constrained Shortest Path Problems: Exact Solution Approaches". In: *Networks* 62.3 (2013), pp. 183–200 (cit. on p. 37).

[153] Stefan Irnich. "Resource Extension Functions: Properties, Inversion, and Generalization to Segments". In: *OR Spectrum* 30.1 (Nov. 12, 2007), pp. 113–148. ISSN: 0171-6468, 1436-6304. DOI: 10.1007/s00291-007-0083-6. URL: http://link.springer.com/10.1007/s00291-007-0083-6 (visited on 05/28/2022) (cit. on p. 39).

[154] Ruslan Sadykov, Eduardo Uchoa, and Artur Pessoa. "A Bucket Graph–Based Labeling Algorithm with Application to Vehicle Routing". In: *Transportation Science* 55.1 (2021), pp. 4–28. ISSN: 15265447. DOI: 10.1287/TRSC.2020.0985 (cit. on pp. 40, 79, 83, 118, 120).

[155] Giovanni Righini and Matteo Salani. "Dynamic Programming Algorithms for the Elementary Shortest Path Problem with Resource

Constraints". In: *Electronic Notes in Discrete Mathematics* 17 (2004), pp. 247–249. ISSN: 15710653. DOI: 10.1016/j.endm.2004.03.047 (cit. on pp. 40, 43).

[156] Leonardo Taccari. "Integer Programming Formulations for the Elementary Shortest Path Problem". In: *European Journal of Operational Research* 252.1 (2016), pp. 122–130. ISSN: 03772217. DOI: 10.1016/j.ejor.2016.01.003 (cit. on pp. 40, 109).

[157] Michael Drexl and Stefan Irnich. "Solving Elementary Shortest-Path Problems as Mixed-Integer Programs". In: *OR Spectrum. Quantitative Approaches in Management* 36.2 (2014), pp. 281–296 (cit. on p. 40).

[158] Markó Horváth and Tamás Kis. "Solving Resource Constrained Shortest Path Problems with LP-based Methods". In: *Computers & Operations Research* 73 (2016), pp. 150–164 (cit. on p. 41).

[159] J. E. Beasley and N. Christofides. "An Algorithm for the Resource Constrained Shortest Path Problem". In: *Networks* 19.4 (1989), pp. 379–394. ISSN: 10970037. DOI: 10.1002/net.3230190402 (cit. on pp. 43, 44, 47).

[160] Moshe Sniedovich. "Dijkstra's Algorithm Revisited: The Dynamic Programming Connexion". In: *Control and cybernetics* 35.3 (2006), pp. 599–620 (cit. on p. 43).

[161] Irina Dumitrescu and Natashia Boland. "Improved Preprocessing, Labeling and Scaling Algorithms for the Weight-Constrained Shortest Path Problem". In: *Networks: An International Journal* 42.3 (2003), pp. 135–153 (cit. on p. 43).

[162] W. Matthew Carlyle, Johannes O. Royset, and R. Kevin Wood. "Lagrangian Relaxation and Enumeration for Solving Constrained Shortest-Path Problems". In: *Networks* 52.4 (2008), pp. 256–270. ISSN: 00283045. DOI: 10.1002/net.20247 (cit. on p. 43).

[163] Ranga Muhandiramge and Natashia Boland. "Simultaneous Solution of Lagrangean Dual Problems Interleaved with Preprocessing for the Weight Constrained Shortest Path Problem". In: *Networks* 53.4 (2009), pp. 358–381. ISSN: 00283045. DOI: 10.1002/net.20292 (cit. on p. 43).

[164] Dominique Feillet, Pierre Dejax, and Michel Gendreau. "Traveling Salesman Problems with Profits". In: *Transportation science* 39.2 (2005), pp. 188–205 (cit. on p. 48).

[165] Bruce L Golden, Larry Levy, and Rakesh Vohra. "The Orienteering Problem". In: *Naval Research Logistics (NRL)* 34.3 (1987), pp. 307–318 (cit. on p. 48).

[166] Gilbert Laporte and Silvano Martello. "The Selective Travelling Salesman Problem". In: *Discrete Applied Mathematics* 26.2-3 (1990), pp. 193–207. ISSN: 0166218X. DOI: 10.1016/0166-218X(90)90100-Q (cit. on p. 48).

[167] Mauro Dell'Amico, Francesco Maffioli, and Peter Värbrand. "On Prize-Collecting Tours and the Asymmetric Travelling Salesman Problem". In: *International Transactions in Operational Research* 2.3 (1995), pp. 297–308 (cit. on p. 48).

[168] Egon Balas. "The Prize Collecting Traveling Salesman Problem". In: *Networks. An International Journal* 19.6 (1989), pp. 621–636 (cit. on p. 48).

[169] Egon Balas. "The Prize Collecting Traveling Salesman Problem: II. Polyhedral Results". In: *Networks. An International Journal* 25.4 (1995), pp. 199–216 (cit. on p. 48).

[170] Roberto Baldacci, Mauro Dell'Amico, and J Salazar González. "The Capacitated M-Ring-Star Problem". In: *Operations research* 55.6 (2007), pp. 1147–1162 (cit. on pp. 48, 52).

[171] Adam N Letchford, Saeideh D Nasiri, and Dirk Oliver Theis. "Compact Formulations of the Steiner Traveling Salesman Problem and Related Problems". In: *European Journal of Operational Research* 228.1 (2013), pp. 83–92 (cit. on p. 48).

[172] Matteo Fischetti, Juan Jose Salazar Gonzalez, and Paolo Toth. "Solving the Orienteering Problem through Branch-and-Cut". In: *INFORMS Journal on Computing* 10.2 (1998), pp. 133–148 (cit. on p. 48).

[173] Michel Gendreau, Gilbert Laporte, and Frédéric Semet. "A Branch-and-cut Algorithm for the Undirected Selective Traveling Salesman Problem". In: *Networks* 32.4 (1998), pp. 263–273. ISSN: 00283045. DOI:

10.1002/(sici)1097-0037(199812)32:4<263::aid-net3>3.3.co;2-h (cit. on p. 48).

[174] Laurence A Wolsey. *Integer Programming*. John Wiley & Sons, 2020 (cit. on p. 50).

[175] Ruslan Sadykov and François Vanderbeck. "BaPCod-a Generic Branch-and-Price Code". PhD thesis. Inria Bordeaux Sud-Ouest, 2021 (cit. on pp. 57, 81, 85, 88, 115).

[176] Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis II. "An Analysis of Several Heuristics for the Traveling Salesman Problem". In: *SIAM journal on computing* 6.3 (1977), pp. 563–581 (cit. on p. 58).

[177] David S Johnson and Lyle A McGeoch. "Experimental Analysis of Heuristics for the STSP". In: *The Traveling Salesman Problem and Its Variations*. Springer, 2007, pp. 369–443 (cit. on p. 58).

[178] Barun Chandra, Howard Karloff, and Craig Tovey. "New Results on the Old K-Opt Algorithm for the Traveling Salesman Problem". In: *SIAM Journal on Computing* 28.6 (1999), pp. 1998–2029 (cit. on p. 60).

[179] Andrew V Goldberg. "An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm". In: *Journal of algorithms* 22.1 (1997), pp. 1–29 (cit. on p. 67).

[180] Ralph E Gomory and Tien Chung Hu. "Multi-Terminal Network Flows". In: *Journal of the Society for Industrial and Applied Mathematics* 9.4 (1961), pp. 551–570 (cit. on p. 67).

[181] Gerhard Reinelt. "Tsplib95". In: *Interdisziplinäres Zentrum für Wissenschaftliches Rechnen (IWR), Heidelberg* 338 (1995), pp. 1–16 (cit. on p. 80).

[182] Gerhard Reinelt. "TSPLIB—A Traveling Salesman Problem Library". In: *ORSA journal on computing* 3.4 (1991), pp. 376–384 (cit. on p. 80).

[183] François Vanderbeck. "Branching in Branch-and-Price: A Generic Scheme". In: *Mathematical Programming* 130.2 (2011), pp. 249–294 (cit. on p. 82).

[184] E. D. Dolan and J.J. Moré. "Benchmarking Optimization Software with Performance Profiles". In: *Mathematical Programming* 91.2 (2002), pp. 201–213 (cit. on p. 90).

[185] Andrea Lodi and Andrea Tramontani. “Performance Variability in Mixed-Integer Programming”. In: *Theory Driven by Influential Applications*. INFORMS, 2013, pp. 1–12 (cit. on pp. 91, 112).

[186] Martina Fischetti and Matteo Fischetti. “Matheuristics”. In: *Handbook of Heuristics*. Springer, 2018, pp. 121–153 (cit. on p. 108).

[187] Matteo Fischetti and Andrea Lodi. “Local Branching”. In: *Mathematical programming* 98.1-3 (2003), pp. 23–47 (cit. on pp. 108, 111, 112).

[188] Bezalel Gavish and Stephen C Graves. “The Travelling Salesman Problem and Related Problems”. In: (1978) (cit. on p. 109).

[189] Richard T Wong. “Integer Programming Formulations of the Traveling Salesman Problem”. In: *Proceedings of the IEEE International Conference of Circuits and Computers*. IEEE Press Piscataway NJ. 1980, pp. 149–152 (cit. on p. 109).

[190] A Claus. “A New Formulation for the Travelling Salesman Problem”. In: *SIAM Journal on Algebraic Discrete Methods* 5.1 (1984), pp. 21–25 (cit. on p. 109).

[191] Brian W Kernighan and Shen Lin. “An Efficient Heuristic Procedure for Partitioning Graphs”. In: *The Bell system technical journal* 49.2 (1970), pp. 291–307 (cit. on p. 109).

[192] Matteo Fischetti, Fred Glover, and Andrea Lodi. “The Feasibility Pump”. In: *Mathematical Programming* 104.1 (2005), pp. 91–104 (cit. on p. 111).

[193] Emilie Danna, Edward Rothberg, and Claude Le Pape. “Exploring Relaxation Induced Neighborhoods to Improve MIP Solutions”. In: *Mathematical Programming* 102.1 (2005), pp. 71–90 (cit. on p. 111).

[194] Edward Rothberg. “An Evolutionary Algorithm for Polishing Mixed Integer Programming Solutions”. In: *INFORMS Journal on Computing* 19.4 (2007), pp. 534–541 (cit. on p. 111).

[195] Adam N Letchford and Georgia Souli. “Lifting the Knapsack Cover Inequalities for the Knapsack Polytope”. In: *Operations Research Letters* 48.5 (2020), pp. 607–611 (cit. on p. 112).

[196] Laurence A Wolsey. "Valid Inequalities for 0–1 Knapsacks and Mips with Generalised Upper Bound Constraints". In: *Discrete Applied Mathematics* 29.2-3 (1990), pp. 251–261 (cit. on p. 112).

[197] Manfred W Padberg, Tony J Van Roy, and Laurence A Wolsey. "Valid Linear Inequalities for Fixed Charge Problems". In: *Operations Research* 33.4 (1985), pp. 842–861 (cit. on p. 112).

[198] Robert C Brigham and Ronald D Dutton. "On Clique Covers and Independence Numbers of Graphs". In: *Discrete Mathematics* 44.2 (1983), pp. 139–144 (cit. on p. 112).

[199] Alberto Caprara and Matteo Fischetti. "${$0, 1/2$}$-Chvátal-Gomory Cuts". In: *Mathematical Programming* 74.3 (1996), pp. 221–235 (cit. on p. 112).

[200] Ricardo Lima and EWO Seminar. "IBM ILOG CPLEX-What Is inside of the Box?" In: *Proc. 2010 EWO Seminar*. 2010, pp. 1–72 (cit. on p. 112).

[201] Andrea Lodi. "The Heuristic (Dark) Side of MIP Solvers". In: *Hybrid Metaheuristics*. Springer, 2013, pp. 273–284 (cit. on p. 112).

[202] Robert Bixby and Edward Rothberg. "Progress in Computational Mixed Integer Programming–a Look Back from the Other Side of the Tipping Point". In: *Annals of Operations Research* 149.1 (2007), p. 37 (cit. on p. 112).

[203] Laurence A Wolsey and George L Nemhauser. *Integer and Combinatorial Optimization*. Vol. 55. John Wiley & Sons, 1999 (cit. on p. 112).

[204] Manfred Padberg and Giovanni Rinaldi. "A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems". In: *SIAM review* 33.1 (1991), pp. 60–100 (cit. on p. 112).

[205] Leonid Genrikhovich Khachiyan. "A Polynomial Algorithm in Linear Programming". In: *Doklady Akademii Nauk*. Vol. 244. 5. Russian Academy of Sciences. 1979, pp. 1093–1096 (cit. on p. 112).

[206] E Robert Bixby, Mary Fenelon, Zonghao Gu, Ed Rothberg, and Roland Wunderling. "MIP: Theory and Practice—Closing the Gap". In: *IFIP Conference on System Modeling and Optimization*. Springer, 1999, pp. 19–49 (cit. on p. 113).