Titre: Title:	Improvements on Column-Generation-Based Algorithms for Vehicle Routing and Other Combinatorial Problems
Auteur: Author:	Luciano Carlos Azevedo da Costa
Date:	2020
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Azevedo da Costa, L. C. (2020). Improvements on Column-Generation-Based Algorithms for Vehicle Routing and Other Combinatorial Problems [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/4226/

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Improvements on Column-Generation-Based Algorithms for Vehicle Routing and Other Combinatorial Problems

LUCIANO CARLOS AZEVEDO DA COSTA
Département de mathématiques et de génie industriel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Mathématiques

Mars 2020

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Improvements on Column-Generation-Based Algorithms for Vehicle Routing and Other Combinatorial Problems

présentée par **Luciano Carlos AZEVEDO DA COSTA**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

Issmaïl EL HALLAOUI, président
Guy DESAULNIERS, membre et directeur de recherche
Claudio CONTARDO, membre et codirecteur de recherche
Jean-François CORDEAU, membre
Jean-François CÔTÉ, membre externe

DEDICATION

To my parents, Carlos and Luciane,
to my sister, Lays, and to the
love of my life, Renally...

ACKNOWLEDGEMENTS

The completion of this thesis was only possible due to the support of numerous people. They have directly or indirectly assisted me during some moment of this long journey, that was this Ph.D. To all of them, I extend my deepest gratitude. In particular, I would like to sincerely thank:

- My advisors professors Guy Desaulniers and Claudio Contardo. Thanks a lot for having given me the opportunity of pursuing my Ph.D. at Polytechnique Montreal and for having believed in my work. I do not have words to express my gratitude for the countless hours that you have spent teaching, guiding, and mentoring me, not to mention all the time correcting my long texts. You have always been very kind and supportive during all these years. I am very fortunate to have worked under the supervision of such inspiring and brilliant professors.

- My family back in Brazil, especially, my parents Carlos and Luciane, and my sister, Lays, for their unconditional love and support. Thanks for having always been my safe haven in all the moments of anguish, anxiety, and suffering. Thanks for having taught me to be the person I have become, which always strives to make dreams come true. I love you.

- My girlfriend, soon to be fiancé, Renally. For the past ten years, you have been my best friend and you have taken part in most of my accomplishments. Thanks for your support, love, and friendship. Thanks for understanding the reason of my absence. Thanks for having changed the plans of your life and having come to spend some time in Canada so that we could be together. I love you.

- The collaborators in the works composing this thesis: Diego Pecin and Julian Yarkony. Thanks for your work, patience, availability, and for always having found the time to answer my long e-mails and several questions.

- The professors composing the jury: Daniel Aloise, Issmail El Hallaoui, Jean-François Cordeau, and Jean-François Côté. Thanks for having accepted to be in my jury and for all the comments, suggestions, and detailed corrections that you gave me.

- My flatmates, along all these years, who have become friends: Bruno, Emanuel, Gabriel, Larissa, Leandro, Raphael, and Roosevelt. Thank you for putting up with me and all

my stress. Thanks for the moments of de-stress and board-game nights. You have made these years in Montreal more bearable.

- My Brazilian and Colombian friends: Aldair, Alfredo, Breno, Camila, Diego Fiorotto, Diego Pecin, Gislaine, Karim, Larissa, Luiza, Pedro, Renata, and Vinicius. Thanks for having always helped me to feel closer to home.

- The wonderful people that I had the opportunity to meet in Montreal, especially: Carlos, Claudio Sole, Clement, David, Eglantine, Filippo, Gianluca, Giulia, Greta, Jaime, Julian, Kenjy, Lucie, Maria, Mathieu Tanneau, Matthieu Gruson, Michael, Paul, Rodrigo, Rosemarie, Serena, and Vilmar. All of you played an important role in all of this. Your friendship and great affection toward me were essential for me to endure all the hardships during the Ph.D. Thanks for all the great moments that we have spent together.

- My officemates: Mahsa, Sebastian, Utsav, Nadia, and Rachid.

- The GERAD staff: Marie, Marilyne, Karine, Carole, Edoh, and Pierre. Thanks for always being so kind and assistive. Thanks for all your availability and for always having rendered me daily assistance, especially during my period as a GERAD student representative.

- If today I am about to finish my Ph.D., I own all my gratitude to Professor Anand Subramanian that introduced me to the field of Operations Research. Thanks for all the guidance and for having always believed in me. Also, despite the long-distance, thank you for all your encouragement and support during the time that I spent in Montreal.

- I could never forget my friends: Raphael, Walton, and Teobaldo. Thanks for the numerous consultations and the countless hours on Skype/Hangout/WhatsApp calls for discussing some research related topics or just for talking non-sense. During all my time abroad, but especially in the last months, your support was fundamental.

- Last, but not least, the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Fonds de recherche du Québec – Nature et technologies (FQRNT) for their financial support, which made this thesis possible.

RÉSUMÉ

Plusieurs applications dans le contexte de la logistique et de la planification de la production peuvent être modélisées comme des problèmes d'optimisation combinatoire (POC). En particulier, l'un des problèmes les plus étudiés dans ce domaine est le problème de tournées de véhicules (PTV). Le PTV consiste à trouver des tournées de véhicules qui minimisent le coût total de transport pour visiter un ensemble de clients, de telle sorte que leur demande soit complètement satisfaite en une seule visite, et que la capacité des véhicules ne soit jamais dépassée. Présentement, la principale méthode de résolution exacte pour les PTVs est la génération de colonnes. Dans cette thèse, nous nous intéressons à l'étude des algorithmes de génération de colonnes et proposons de nouvelles idées pour améliorer leur efficacité.

Dans le Chapitre 4, nous présentons une revue de littérature très exhaustive dans laquelle nous mettons en évidence les principales contributions algorithmiques et de modélisation apportées au cours des dernières années dans la cadre du développent des algorithmes de génération de colonnes et de plans coupants intégrés à des méthodes d'énumération implicite pour le PTV. Notre étude est divisée en deux parties principales. Dans la première partie, nous présentons des aspects qui peuvent s'appliquer à la plupart des variantes de PTV, à savoir : des algorithmes de résolution du sous-problème de la génération de colonnes, la séparation de plans coupants, les stratégies de branchement et la stabilisation des variables duales dans le problème-maître. La deuxième partie est dédiée à la résolution de problèmes spécifiques. Dans cette partie, nous discutons comment les spécificités de chaque problème peuvent êtres traitées lors du développement des algorithmes d'énumération implicite combinant génération de colonnes et plans coupants. On étude les attributs suivants : l'existence d'une flotte hétérogène et des dépôts multiples, la considération de fenêtres de temps souples chez les clients, la possibilité d'effectuer des livraisons fractionnées, les coûts dépendant du temps, la réalisation de cueillettes et livraisons, la présence d'incertitude dans les données et des aspects environnementaux.

Dans le Chapitre 5, nous proposons un algorithme sélectif pour résoudre des sous-problèmes de la génération de colonnes afin de générer des routes relaxées de type *arc-ng*. Notre méthode considère une généralisation de la dominance par ensemble proposée par Bulhões et al. [1]. Les résultats numériques obtenus sur des instances du PTV avec fenêtres de temps montrent que le nouveau mécanisme aide à réduire le nombre d'étiquettes non-dominées dans l'algorithme d'étiquetage utilisé pour résoudre le sous-problème et, par conséquent, le temps de calcul.

Enfin, dans le Chapitre 6, nous présentons une nouvelle méthode de stabilisation pour

des POCs avec des structures qui favorisent l'parution de dégénérescence. Le nouvel algorithme de stabilisation, appelé `dyn`-SAR, est basé sur la séparation dynamique de contraintes agrégées, qui sont obtenues en additionnant des contraintes du problème maître de génération de colonnes. L'effet de stabilisation induit par `dyn`-SAR provient des fortes interactions qui surviennent entre les variables duales, ce qui n'est pas observé lors de la résolution explicite d'une formulation de partition d'ensemble (recouvrement / empaquetage). L'intérêt principal pour l'utilisation du `dyn`-SAR est dû à sa simplicité et généralité. Ce dernier aspect est confirmé dans nos expériences, où nous considérons des problèmes dont la fonction objectif et le sous-problème de génération de colonnes sont considérablement différents. Les résultats numériques montrent un avantage important du `dyn`-SAR par rapport à une méthode de génération de colonnes standard en termes de nombre d'itérations et de temps de calcul.

ABSTRACT

Several applications arising in the context of logistics and production planning can be modeled as combinatorial optimization problems (COPs). In particular, one of the most studied problems in this field is the vehicle routing problem (VRP). The VRP is the problem of finding least-cost routes to visit a set of customers in such a way that their demand is completely satisfied in a single visit, and the capacity of vehicles is not exceeded. Nowadays, the leading exact method to cope with different classes of VRPs is column generation (CG). In this thesis, we are interested in studying CG algorithms and propose new ideas to enhance their efficiency.

In Chapter 4, we present a methodological survey in which we highlight and discuss the main algorithmic and modeling contributions made over the years in the context of branch-price-and-cut methods for VRPs. Our study is divided into two main parts. In the first part, we discuss topics that may apply to most VRPs variants, namely: pricing algorithms, cut separation, branching strategies, and dual variable stabilization. The second part is more problem-oriented and describes how aspects such as heterogeneous fleet, multi-depots, soft time windows, split deliveries, time dependency, pickups and deliveries, uncertainty, and environmental aspects can be handled in devising branch-price-and-cut algorithms.

In Chapter 5, we propose a selective pricing algorithm to solve pricing subproblems defined in terms of *arc-ng*-route relaxations. Our method extends the set-based dominance rule proposed by Bulhões et al. [1], making it more general and stronger. Computational experiments performed over instances of the VRP with time windows show that the proposed mechanism helps in reducing the number of non-dominated labels kept by the labeling algorithm and, as a consequence, the CPU time.

Finally, in Chapter 6, we develop a new stabilization framework to tackle COPs with degenerate structures. The new stabilization method, called `dyn`-SAR, relies on the dynamic separation of aggregated constraints, which are obtained by summing up constraints from the CG master problem. The stabilization effect induced by `dyn`-SAR is due to strong interactions that arise from dual variables, which is not observed when solving explicitly a set-partitioning (covering/packing) formulation. The main interests in using the `dyn`-SAR method are its simplicity and generality. The latter aspect is confirmed in our experiments, where we solve instances from problems differing considerably in their objective function and pricing subproblem. Numerical results show a clear advantage of `dyn`-SAR over a standard CG method in terms of both the number of iterations and running time.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

2-cyc-SPPRC	2-cycle-elimination Shortest Path Problem with Resource Constraints
2PC	2-Path Cut
arc-ng-SPPRC	*arc-ng*-Shortest Path Problem with Resource Constraints
BB	Branch-and-Bound
BC	Branch-and-Cut
BP	Branch-and-Price
BPC	Branch-Price-and-Cut
BPP	Bin Packing Problem
BPPC	Bin-Packing Problem with Conflicts
C-SDVRP	Commodity-Constrained Split Delivery Vehicle Routing Problem
CCP	Chance-Constrained Program
CCVRPSD	Chance-Constrained VRP with Stochastic Demands
CG	Column Generation
CI	Capacity-Indexed
CLRP	Capacitated Location-Routing Problem
COP	Combinatorial Optimization Problem
CPTP	Capacitated Profitable Tour Problem
CS	Cutting-Stock
CTOP	Capacitated Team Orienting Problem
CumVRP	Cumulative Vehicle Routing Problem
CVRP	Capacitated Vehicle Routing Problem
DARP	Dial-a-Ride Problem
DARPSRP	Dial-a-Ride Problem with Split Requests and Profits
DDOI	Deep Dual-Optimal Inequality
DI	Dual Inequality
DOI	Dual-Optimal Inequality
DP	Dynamic Programming
DSDVRPTW	Discrete Split Delivery Vehicle Routing Problem with Time Windows
DSSR	Decremental State-Space Relaxation
DTI	Delivery Triangle Inequality
DW	Dantzig-Wolfe
Dyn-SAR	Dynamic Separation of Aggregated Constraints
EFC	Expected Failure Cost

EMP	Extended Master Problem
EMVRP	Energy Minimization Vehicle Routing Problem
ERMP	Extended Restricted Master Problem
ESPPRC	Elementary Shortest Path Problem with Resource Constraints
EVRP	Electric Vehicle Routing Problem
EVRPTW	Electric Vehicle Routing Problem with Time Windows
FIFO	First-in-First-out
GVRP	Green Vehicle Routing Problem
HFVRP	Heterogeneous Fleet Vehicle Routing Problem
IP	Integer Programming
IPEC	Infeasible Path Elimination Constraints
IPS	Improved Primal Simplex
ISUD	Integral Simplex Using Decomposition
k-CEC	k-cycle Elimination Cut
k-cyc-SPPRC	k-cycle-elimination Shortest Path Problem with Resource Constraints
KPC	Knapsack Problem with Conflicts
kPCs	k-path Cuts
lam-SRC	Limited-arc-memory Subset-Row Cut
LDS	Limited Discrepancy Search
lm-SRC	Limited-memory Subset-Row Cut
LP	Linear Programming
LP-BKP	Linear Relaxation of Bounded Knapsack Problem
lvm-SRC	Limited-vertex-memory Subset-Row Cut
MDVRP	Multi-Depot Vehicle Routing Problem
MDVRPI	Multi-Depot Vehicle Routing Problem with Inter-depot Routes
MEC	Mutual Exclusion Constraint
MIP	Mixed-Integer Programming
MP	Master Problem
MPPE	Multi-Person Pose Estimation
MTVRP	Multi-Trip Vehicle Routing Problem
MTVRPTW-LD	Multi-Trip VRPwith Time Windows and Limited Trip Duration
MWSP	Minimum Weight Set Packing
ng-SPPRC	*ng*-Shortest Path Problem with Resource Constraints
PDP	Pickup and Delivery Problem
PDP-SL	Pickup and Delivery Problem with Scheduled Lines
PDPS	Pickup and Delivery Problem with Shuttles

PDPT	Pickup and Delivery Problem with Transfers
PDPTW	Pickup and Delivery Problem with Time Windows
PDPTWL	PDP with Time Windows and LIFO constraints
PDPTWMS	Pickup and Delivery Problem with Time Windows and Multiple Stacks
PRP	Pollution Routing Problem
PS	Pricing Subproblem
R-DARP	Rich-DARP
RCC	Rounded Capacity Cut
REF	Resource Extension Function
RMP	Restricted Master Problem
RVRP	Robust Vehicle Routing Problem
SCC	Strengthened Capacity Cut
SDC	Strong Degree Cut
SDCTOP	Split Delivery Capacitated Team Orienting Problem
SDVRP	Split Delivery Vector Packing Problem
SDVRPTW	Split Delivery Vehicle Routing Problem with Time Windows
SkPC	Strong k-Path Cuts
SMVC	Strong Minimum Number of Vehicles Inequality
SPDP	Synchronized Pickup and Delivery Problem
SPPRC	Shortest Path Problem with Resource Constraints
SPR	Stochastic Program with Recourse
SRC	Subset-Row Cut
SVRP	Stochastic Vehicle Routing Problem
SVRPTW	Selective Vehicle Routing Problem with Time Windows
TDVRPTW	Time Dependent Vehicle Routing Problem with Time Windows
TOP	Team Orienting Problem
TSP	Travelling Salesman Problem
TSPTW	Travelling Salesman Problem with Time Windows
VPP	Vector Packing Problem
VRP-SL	Vehicle Routing Problem with Service Level constraints
VRPDSTW	VRP with Deliveries, Selective Pickups and Time Windows
VRPP	Vehicle Routing Problem with Profits
VRPSD	Vehicle Routing Problem with Stochastic Demands
VRPSPD	Vehicle Routing Problem with Simultaneous Pickup and Deliveries
VRPTW	Vehicle Routing Problem with Time Windows
VRPTW-ST	VRP with Time Windows and Stochastic Service Times

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Problem setting and motivation

Several problems arising in the context of logistics and production planning can be modeled as combinatorial optimization problems (COPs). A COP can be formally defined as follows. Let $\mathcal{N} = \{1, \ldots, n\}$ be a finite set of items. A feasible solution can be associated with a set $\mathcal{S} = \{S_1, \ldots, S_k\}$, where each S_i is a subset of $\mathcal{N}$. With each solution $\mathcal{S}$ is associated a cost $c_{\mathcal{S}}$, and the set of all feasible solutions $\mathcal{S}$ is denoted $\mathcal{F}$. The structure of set $\mathcal{F}$ varies according to the problem at hand, as different feasibility rules may arise. In the case of the vehicle routing problem (VRP) [3], $\mathcal{N}$ corresponds to the set of customers that must be visited. In turn, a solution $\mathcal{S}$ is associated with a set of routes $\{S_1, \ldots, S_k\}$. A COP consists of finding a solution $\mathcal{S} \in \mathcal{F}$ in such a way that the total cost is minimized or maximized. More formally:

$$\min_{\mathcal{S} \in \mathcal{F}} c_{\mathcal{S}} \tag{1.1}$$

Notice that, performing an exhaustive enumeration of all subsets $\mathcal{S} \in \mathcal{F}$ may be computationally prohibitive, due to the cardinality of $\mathcal{F}$, that is typically exponential on the size of $\mathcal{N}$. For example, if we consider a VRP where 100 customers must be visited, the number of possible routes is 100! ($\approx 9.33 \times 10^{157}$), which is much larger than the number of stars in the Universe! [4]. Thus, developing efficient methods to deal with COPs is crucial.

Over the years, a variety of strategies has been proposed to cope with COPs, among which we can find exact and heuristic/metaheuristic methods, or even hybrid methods, that combine both [5]. Heuristic algorithms are suitable for tackling real-life applications. In general, these methods can be easily implemented and are capable of providing good solutions in reasonable amounts of time. Nevertheless, they do not give any guarantee about the quality of the solutions obtained. Heuristic methods are out of the scope of this thesis, and for this reason, are not discussed in this document. A very recent and detailed discussion about metaheuristics can be found in [6].

In contrast, exact algorithms are capable of providing proven optimal solutions, but at a higher computational effort. In this thesis, we focus on the exact methods that are based on column generation (CG) [7]. These methods are known to be the state-of-the-art strategy to solve several COPs, such as vehicle routing and scheduling [8, 9], packing [10], machine scheduling [11], computer vision problems [12, 13], among others.

When applying CG, COPs are expressed through extended formulations, i.e., formulations containing a large number of variables. In this context, a COP is typically expressed as a set-partitioning (covering/packing) problem, where each variable is associated with a subset $\mathcal{S}$ induced by the problem structure. For example, when modeling a vehicle routing/scheduling problem as a set-partitioning problem, each variable corresponds to a feasible route. For bin-packing problems or cutting stock problems [10], each variable corresponds to a packing/cutting pattern. The same characteristic may be found in several other COPs.

As will be discussed with more details in Chapter 2, CG is an iterative algorithm to deal with linear problems containing a large number of variables (see [7, 14, 15]). It follows the principle of the simplex method, i.e., at each iteration, a basic solution is found, and then variables with negative reduced costs are searched to enter the basis. However, given the massive number of variables that may be associated with the problems being solved, looking explicitly for all the columns to enter the basis may be computationally prohibitive. Thus, CG algorithms consider at each time only a subset of variables, i.e., a restricted master problem (RMP). Then, at each iteration of the method, variables are generated by solving a pricing subproblem (PS). Solving these subproblems typically corresponds to finding combinatorial objects like paths, sets, or permutations [15]. Hence, state-of-the-art algorithms can be used to solve the subproblems. The PS considers dual information from the RMP and returns, if possible, a variable with a negative reduced cost. This process continues iteratively until no more variables with a negative reduced cost can be found, which means that an optimal solution to the problem has been found.

In this thesis, special attention is dedicated to the VRP, this problem being the central theme of Chapters 4 and 5. The VRP is one of the most studied problems in the field of operations research (OR). It was proposed by Dantzig and Ramser [16] when solving a practical problem arising in the context of gasoline delivery from the bulk terminal to service stations. In the classic VRP, the so-called Capacitated VRP (CVRP), a homogeneous fleet of capacitated vehicles is available at a central depot to visit a set of customers. Every customer must be visited exactly once, and their demand must be completely satisfied. The CVRP consists of designing a set of routes, starting and ending at the same and unique depot, in such a way that the capacity of the vehicles is never exceeded. The most common objective related to this problem is the minimization of the traveling costs, which are usually proportional to the traveling distances. In the literature, a variety of VRP variants has been proposed to model different aspects of real-life problems [3, 17]. Other COPs, such as the bin-packing with conflicts [18, 19] and the multi-person pose estimation [13], are also considered in this thesis. Their definitions are given in Chapter 6.

1.2 Objectives

In recent years, CG has been the leading technique to cope with many classes of VRPs. Since the seminal work by Desrosiers et al. [20], several methods and strategies have been proposed and considered in the design of CG methods for VRPs. Specifically, CG has been successfully employed to develop branch-price-and-cut (BPC) algorithms. Yet, performing a proper implementation of a CG (branch-and-price) algorithm or even being able to understand it, is often a hard task [21]. Additionally, despite the success achieved by CG algorithms when tackling COPs, they may face some issues related to its bad convergence behavior and to the computational burden of solving the CG pricing subproblem. The latter is particularly critical when solving VRPs.

Given the above, the objective of this thesis is to discuss and propose techniques that can be used to enhance the performance of CG based algorithms. More specifically:

- Highlight and discuss the main methodological and modeling contributions made over the years in the context of BPC for VRPs;
- Design less restrictive dominance rules to allow pricing subproblems arising from VRPs to be solved more efficiently;
- Propose a stabilization framework to reduce the impact of dual instability and degenerate structures when applying CG to COPs.

1.3 Thesis outline

The remainder of this document is organized as follows: in Chapter 2, we give some background and notions on how a CG algorithm works and how a BPC algorithm can be implemented. Moreover, we present a concise literature review concerning the main topics to be discussed in this thesis. Chapter 3 describes the structure of this document. Chapter 4 consists of a methodological survey, where a detailed study about the different aspects involving the design of BPC algorithms for VRPs is performed. Chapter 5 describes a new pricing algorithm to be applied to VRPs. Chapter 6 presents a new stabilization framework that is based on the dynamic separation of aggregated constraints of the RMP. Finally, in Chapter 7, we provide a general discussion regarding the three objectives of this thesis, and in Chapter 8, we draw some conclusions.

CHAPTER 2 CRITICAL LITERATURE REVIEW

Although this thesis is composed of self-contained chapters, in this introductory chapter, we explain some important definitions and concepts related to CG based algorithms. Moreover, we provide a concise literature review regarding the main strategies designed to address some of the issues faced by CG algorithms.

2.1 Column generation for combinatorial optimization problems

In this section, we give a general description of a CG algorithm. Initially, we discuss the underlying structure of combinatorial optimization problems (COPs) favoring the application of CG algorithms. Later, we present the main components of a CG algorithm.

2.1.1 Dantzig-Wolfe decomposition

Because integer programming (IP) problems are also defined over a discrete domain, they are closely linked to COPs. As a consequence, practically all COPs can be expressed as an IP problem in its compact form [22]:

$$\begin{aligned} \min \quad & c^\intercal \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \geq b \\ & D\mathbf{x} \geq d \\ & \mathbf{x} \in \mathbb{Z}_+^n, \end{aligned} \tag{2.1}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{Z}_+^{m \times n}$, and $D \in \mathbb{Z}_+^{l \times n}$, with n being the number of decision variables, and m and l denoting the number of constraints to which matrices of coefficients A and D are respectively associated with. From now on, the terms COPs and IP problems will be used interchangeably in this text. The previous formulation is said to be compact because the number of variables and constraints is polynomial in the size of the problem.

In practical situations, solving a COP such as (2.1) may not be an easy task. Even if the problem (2.1) is a linear programming (LP) problem, due to the domain over which the variables of the problem are defined, all the theory and algorithms that have been developed in the literature to solve LP problems may not be applied directly to solve COPs.

Nevertheless, some COPs have a particular structure that makes them suitable for the application of the Dantzig-Wolfe (DW) decomposition. DW decomposition is a technique proposed

by Dantzig and Wolfe [23] that can be applied to reformulate problems containing sparse and well-structured constraint matrices. This structure allows them to be decomposed into independent blocks (sub-systems/subproblems), each one corresponding to a subset of variables and constraints from the original problem. These blocks can be treated separately and then combined afterward by a coordinating problem (also called *master problem*), to obtain the solution for the original problem. In this reformulation, the solution of each subproblem is expressed in the coordinating problem using new variables. The idea behind this approach is to reduce the complexity of large-scale problems through the solution of smaller problems.

From problem (2.1), it is possible to define the discrete set $X = \{\mathbf{x} \in \mathbb{Z}_+^n \mid D\mathbf{x} \geq d\}$. Because X is assumed to be finite, problem (2.1) can be equivalently written as:

$$\begin{aligned} \min \quad & c^\intercal \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \geq b \\ & \mathbf{x} \in conv(X), \end{aligned} \tag{2.2}$$

where $conv(X)$ is the convex hull of X.

For the sake of simplicity, we assume $conv(X)$ to be a bounded set. Let q be the number of its extreme points. Hence, we may express the feasible set of problem (2.1) as being $X = \{x_1, \ldots, x_q\}$, where each element of X corresponds to a feasible solution of the problem. As a consequence, one may write any point in X as the convex combination of all extreme points $\{x_1, \ldots, x_q\}$. More formally,

$$\begin{aligned} & \mathbf{x} = \sum_{j=1}^{q} x_j \lambda_j \\ & \sum_{j=1}^{q} \lambda_j = 1 \\ & \lambda_i \geq 0 \quad i = 1, \ldots, q \end{aligned} \tag{2.3}$$

It is important to remark that, in a more general case, set X might be unbounded. Hence, $\mathbf{x}$ in (2.3) would need to be written as the linear combination of extreme points and extreme rays. However, because all problems discussed in this thesis are associated with a bounded $conv(X)$, we limit our discussion to the simplest case.

By replacing (2.3) into (2.2), and by denoting $c_j = c^\intercal x_j$, and $a_j = Ax_j$, problem (2.1) can be reformulated as:

$$\min \sum_{j=1}^{q} c_j \lambda_j \tag{2.4}$$

$$\text{s.t.} \sum_{j=1}^{q} a_j \lambda_j \geq b \tag{2.5}$$

$$\sum_{j=1}^{q} \lambda_j = 1 \tag{2.6}$$

$$\lambda_j \geq 0 \quad j = 1, \ldots, q \tag{2.7}$$

$$\mathbf{x} = \sum_{j=1}^{q} x_j \lambda_j \tag{2.8}$$

$$\mathbf{x} \in \mathbb{Z}_+^n. \tag{2.9}$$

This new problem is the so-called *master problem* of the DW decomposition.

DW decomposition allows the development of efficient algorithms to solve COPs. Provided that the structure of the sub-systems $D\mathbf{x} \geq d$ does not possess the *integrality property*, i.e., the solution of the linear relaxation of the problem is integer, bounds generated when solving the linear relaxation of (2.4)–(2.9) are stronger than those obtained from the linear relaxation of (2.1) [24]. This is due to the fact that by employing a DW reformulation, one works with a partial representation of the convex hull of the original problem, leading to a reduction in the *degree of infeasibility* of the problem. Moreover, extended formulations like (2.4)–(2.9) help breaking symmetries in the formulation [25], and allow *concealing* some potential complex aspects arising from a given problem definition (e.g., its non-linearity [8]) by encoding it within the vector a_j. Nevertheless, a disadvantage of reformulating a problem employing DW decomposition is that the coordinating problem may contain an exponentially large number of variables. Thus, handling all the variables at once may be computationally prohibitive. Therefore, the use of a technique such as CG, which is discussed in the next section, is suitable to address these problems.

The effectiveness of DW decomposition in enabling the design of efficient algorithms is even higher if the matrix D has a block diagonal structure, i.e., the matrix D may be rearranged in such a way to be represented as:

$$D = \begin{pmatrix} D_1 & \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & D_2 & \ldots & \mathbf{0} \\ \vdots & & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \ldots & D_\ell \end{pmatrix}. \tag{2.10}$$

As a consequence, problem (2.1) can be rewritten as:

$$\begin{array}{ll} \min & c^{\intercal}\mathbf{x} \\ \text{s.t.} & A\mathbf{x} \geq b \\ & D_k\mathbf{x}^k \geq d_k, \quad k = 1, \ldots, \ell \\ & \mathbf{x}^k \in \mathbb{Z}_+^{n_k}, \quad k = 1, \ldots, \ell \end{array} \tag{2.11}$$

where $\mathbf{x}^k$, $k = 1, \ldots, \ell$, are disjoint vectors such that $\mathbf{x} = (x_1, x_2, \ldots, x_k)$.

The block diagonal structure allows the set X to be partitioned into disjoint sets $X^k = \{x^k \in \mathbb{Z}_+^{n_k} \mid D_k x^k \geq d_k\}$, $k = 1, \ldots, \ell$. In this case, relation (2.3) becomes:

$$\begin{aligned} & \mathbf{x}^k = \sum_{j=1}^{q_k} x_j^k \lambda_j^k && k = 1, \ldots, \ell \\ & \sum_{j=1}^{q_k} \lambda_j^k = 1 && k = 1, \ldots, \ell \\ & \lambda_i^k \geq 0 && k = 1, \ldots, \ell,\ i = 1, \ldots, q_k, \end{aligned} \tag{2.12}$$

where q_k corresponds to the number of extreme points in X^k.

In turn, the COP (2.11) can be presented as follows:

$$\min \sum_{k=1}^{\ell} \sum_{j=1}^{q^k} c_j^k \lambda_j^k \tag{2.13}$$

$$\text{s.t.} \sum_{k=1}^{\ell} \sum_{j=1}^{q^k} a_j^k \lambda_j^k \geq b \tag{2.14}$$

$$\sum_{j=1}^{q^k} \lambda_j = 1 \qquad k = 1, \ldots, \ell \tag{2.15}$$

$$\lambda_j^k \geq 0 \qquad k = 1, \ldots, \ell,\ i = 1, \ldots, q_k \tag{2.16}$$

$$\mathbf{x}^k = \sum_{j=1}^{q_k} x_j^k \lambda_j \qquad k = 1, \ldots, \ell \tag{2.17}$$

$$\mathbf{x}^k \in \mathbb{Z}_+^{n_k} \qquad k = 1, \ldots, \ell. \tag{2.18}$$

2.1.2 Column generation

CG is an iterative approach to deal with (linear) problems containing a vast number of variables (see [7, 14, 15]), like the ones obtained via DW decomposition. The basis for this technique has been conceived in [23, 26, 27].

To explain the principle of CG, let us consider the reformulation (2.4)–(2.9) of problem (2.1).

Its linear relaxation can be obtained by dropping the integrality of $\mathbf{x}$ variables. Additionally, let Ω be the set of indices $\{1, \ldots, q\}$. CG follows the principle of the simplex method, i.e., at each iteration, a basic solution is found, and then variables with a negative reduced costs are searched to enter the basis. However, because the size of set Ω is typically very large, looking explicitly for all the variables to enter the basis may not be possible. For this reason, CG algorithms consider at each time only a reasonably small subset $\Omega' \subseteq \Omega$, defining the so-called *restricted master problem* (RMP). By working with a row-wise representation of constraints (2.5), with set M containing their indices, the RMP is expressed as follows:

$$\min \sum_{j \in \Omega'} c_j \lambda_j \tag{2.19}$$

$$\text{s.t.} \sum_{j \in \Omega'} a_j^i \lambda_j \geq b_i \qquad i \in M \tag{2.20}$$

$$\sum_{j \in \Omega'} \lambda_j = 1 \tag{2.21}$$

$$\lambda_j \geq 0 \qquad j \in \Omega'. \tag{2.22}$$

In a CG algorithm, at each iteration, new variables are generated by solving a pricing sub-problem (PS). The PS considers dual information from the RMP and returns, if possible, a variable with a negative reduced cost. This process continues iteratively until no more variables with a negative reduced cost can be found, meaning than an optimal solution for the problem has been found. From problem (2.1), the PS would be formulated as follows:

$$\begin{aligned} \min \quad & (c^\intercal - \pi^\intercal A)^\intercal \mathbf{x} - \gamma \\ \text{s.t.} \quad & D\mathbf{x} \geq d \\ & \mathbf{x} \in \mathbb{Z}_+^n, \end{aligned} \tag{2.23}$$

where π and γ are the dual variables associated, respectively, with constraints (2.20) and (2.21). The term $(c - \pi^\intercal A)$ corresponds to the reduced cost associated with the vector of variables $\mathbf{x}$. If we consider the variables individually, and adopt the notation from (2.19)–(2.22), the PS could be expressed as:

$$j^* \in \arg\min_{j \in \Omega} \{c^* := c_j - \pi^\intercal a_j - \gamma\}, \tag{2.24}$$

which returns a variable in Ω with the least reduced cost (most negative). When c^* is such that $c^* > 0$, the search for new variables ends.

For several COPs, the PS (2.23) arising from the DW decomposition may correspond to problems with some rich structure. Thus, it may be possible to benefit from families of

algorithms in the literature to solve these problems efficiently. For instance, the VRP and the bin packing problem have their PSs respectively formulated as the elementary shortest path problem with resource constraints (ESPPRC) and the 0-1 knapsack problem. Despite being $\mathcal{NP}$-hard problems, these two problems can be efficiently solved or approximated using pseudo-polynomial algorithms. This topic is further discussed in Section 2.2. For other problems, however, it may be necessary to solve mixed-integer programming (MIP) problems as their PS.

Note that the CG algorithm described in this section is not employed to tackle COPs directly, but rather their linear relaxation. In this case, it is necessary to apply additional procedures to restore the integrality of the obtained solutions. This subject is discussed in the next section. Note that, more recently, some effort has been made toward the development of CG algorithms that are capable of solving the problem without the need for relaxing integrality requirements [28, 29]. This subject is briefly discussed in Section 2.3.

2.1.3 Branch-price-and-cut algorithms

Branch-and-bound (BB) algorithms are one of the most popular methods for solving COPs. BB consists of solving (linear) relaxations of the original problem to explore the solution space. The method enumerates feasible solutions by employing a tree data structure. Each node (*branch*) of the tree represents a region of the solution space. However, not all feasible solutions are enumerated. By using lower and upper bounds (*bound*), which are updated while exploring the tree, some of the nodes (regions) are discarded when they are deemed not to lead to solutions better than the ones currently available. A good tutorial about BB algorithms is given in Chapter 7 of Wolsey [30].

The success of BB algorithms relies on their capacity of computing efficiently tight lower and upper bounds for the problem at hand. The smaller the gap between these bounds, the smaller the number of nodes in the BB tree. For this reason, over the years, enhanced algorithms have been designed by embedding more sophisticated bounding procedures into the BB methods. The better bounds provided by these new methods allow larger instances to be solved. One technique widely used in the late 1990s is the addition of cutting planes (valid inequalities) to the relaxations being solved at each node of the search tree. In this approach, each time a problem relaxation is solved, if the solution is not feasible for the original problem, valid inequalities are generated to cut this solution off. This process is repeated until either a feasible solution is found or no violated cuts are identified. BB algorithms solving linear relaxations strengthened by cutting planes constitute the so-called *branch-and-cut* (BC) algorithms. More details about these procedures can be found in [31].

Despite the huge success accomplished by BC when solving COPs, decomposition methods are currently the leading techniques to cope with some complex COPs. As discussed in the previous section, formulations obtained via DW decomposition are known to provide bounds tighter than those obtained with compact formulations, provided that PSs do not have the integrality property. Therefore, researchers have started to take advantage of this aspect by using extended formulations to produce bounds for BB algorithms. Methods combining CG and BB frameworks are called *branch-and-price* (BP) algorithms. When they were first proposed at the beginning of the 1960s, CG methods were mostly employed to solve LP problems with specific structures. However, after the work by Desrosiers et al. [20], who proposed the first non-trivial BP algorithm, CG started to be seen as a powerful ally to tackle hard COPs. Later, starting with Nemhauser and Park [32], researchers started combining both BC and BP, leading to *branch-price-and-cut* (BPC) algorithms.

2.1.4 Cuts separation

In this section, we briefly discuss the use of cutting planes to reinforce formulations obtained using DW decomposition. There are two main strategies to devise cuts to be incorporated into the CG RMP: 1) by employing variables $\mathbf{x}$ from the compact formulation, or 2) by considering directly variables λ from the extended formulation. According to Poggi de Aragão and Uchoa [33], depending on the choice for 1) or 2), cuts can be classified, respectively, as *robust cuts* and *non-robust cuts*. The distinction between these two types of cuts arises from the impact that they will have on the structure of the PS being solved.

Let us first consider the generic COP (2.1) expressed in its compact form reinforced with valid inequalities $G\mathbf{x} \geq f$,

$$\begin{aligned} \min \quad & c^\intercal \mathbf{x} \\ \text{s.t.} \quad & A\mathbf{x} \geq b \\ & D\mathbf{x} \geq d \\ & G\mathbf{x} \geq f \\ & \mathbf{x} \in \mathbb{Z}^n_+. \end{aligned} \tag{2.25}$$

By performing variable change (2.3) over $G\mathbf{x} \geq f$, and by applying the same decomposition employed in the previous section, we obtain the following RMP, where $g_j = Gx_j$:

$$\min \sum_{j \in \Omega'} c_j \lambda_j \tag{2.19 revisited}$$

$$\text{s.t.} \sum_{j \in \Omega'} a^i_j \lambda_j \geq b_i \qquad i \in M \tag{2.20 revisited}$$

$$\sum_{j\in\Omega'} g_j^i \lambda_j \geq f_i \qquad i \in H \tag{2.26}$$

$$\sum_{j\in\Omega'} \lambda_j = 1 \tag{2.21 revisited}$$

$$\lambda_j \geq 0 \qquad j \in \Omega', \tag{2.22 revisited}$$

where H corresponds to the set of indices of robust constraints (2.26).

Consequently, similarly to what is done in (2.23), the PS can be written as follows:

$$\begin{aligned} \min \quad & (c^\intercal - \pi^\intercal A - \alpha^\intercal G)^\intercal \mathbf{x} - \gamma \\ \text{s.t.} \quad & D\mathbf{x} \geq d \\ & \mathbf{x} \in \mathbb{Z}_+^n, \end{aligned} \tag{2.27}$$

where α is the vector of dual variables associated with constraints (2.26). Notice that the dual variables associated with the additional constraints (2.26) can be directly transferred to the matrix of costs considered when solving the PS. Hence, the structure of the PS is not affected. For this reason, the same algorithm employed to solve (2.23) can also be used to solve (2.27). Another way to devise robust cuts is by considering formulations in which variables are indexed according to some attributes considered in the problem such as load [34] or time [35] consumption. Even if these formulations may have a pseudo-polynomially large number of variables, they can be efficiently handled in the PS.

Now, let us consider the RMP reinforced with inequality (2.28) written directly in terms of the λ variables:

$$\min \sum_{j\in\Omega'} c_j \lambda_j \tag{2.19 revisited}$$

$$\text{s.t.} \sum_{j\in\Omega'} a_j^i \lambda_j \geq b_i \qquad i \in M \tag{2.20 revisited}$$

$$\sum_{j\in\Omega'} t_j^i \lambda_j \geq v_i \qquad i \in W \tag{2.28}$$

$$\sum_{j\in\Omega'} \lambda_j = 1 \tag{2.21 revisited}$$

$$\lambda_j \geq 0, \qquad j \in \Omega' \tag{2.22 revisited}$$

where W corresponds to the set of indices of non-robust constraints (2.28).

Because constraints (2.28) are not necessarily defined as linear functions of the coefficients of $\mathbf{x}$, it is not possible to incorporate their dual variable σ to the cost structure of the PS, as

done in (2.27). Thus, the PS is formulated as follows:

$$\begin{aligned} \min \quad & (c^\intercal - \pi^\intercal A)^\intercal \mathbf{x} - \gamma - T(A\mathbf{x})\ \sigma \\ \text{s.t.} \quad & D\mathbf{x} \geq d \\ & \mathbf{x} \in \mathbb{Z}_+^n, \end{aligned} \tag{2.29}$$

where T is the matrix associated with coefficients $\mathbf{t}$ in (2.28), which are function of the columns (extreme points) generated so far. Contrarily to what happens with γ, the impact of σ on the value of (2.29) is not constant, i.e., it varies in terms of the solution at hand. Due to the non-linear relation induced by the use of non-robust cuts, the complexity of solving the PS may increase considerably.

In Section 4.2.3 and 4.3.2 we present the main robust and non-robust cuts designed in the context of BPC algorithms to solve VRPs. Desaulniers et al. [36], in turn, provide further explanations about the use of cutting planes in the context of generic BPC algorithms.

2.2 Complexity of solving the pricing subproblem

As mentioned in subsection 2.1.1, when CG PSs do not possess the integrality property, DW reformulation yields bounds better than those obtained by solving the corresponding compact formulations. Nevertheless, subproblems with such structure are typically $\mathcal{NP}$-hard, which makes solving them already a challenging task. Since PSs must be solved numerous times during a CG algorithm, devising efficient algorithms to solve them has been one crucial concern for researchers in the field.

Solving these PSs typically corresponds to finding combinatorial objects like paths, sets, or permutations [15]. Therefore, one may exploit the structure of these objects and employ efficient algorithms that have been proposed in the literature to solve them. One technique that has been successfully applied in the literature to solve CG pricing subproblems is dynamic programming (DP). Depending on the problem at hand, DP enables the design of efficient algorithms to tackle hard problems. For example, when solving bin packing problems and VRPs using CG, their PSs are formulated as a knapsack problem and an ESPPRC, respectively. Both the knapsack problem and the ESPPRC are classical $\mathcal{NP}$-hard problems [37,38] that may be solved or approximated using DP with pseudo-polynomial worst-case complexity algorithms [39, 40].

Yet, using DP alone does not guarantee that solving the CG PS subproblem will become an easy task. As a matter of fact, in the last years, a huge effort has been made toward the development of strategies to alleviate the difficulty of solving CG PSs. In the case of the

VRP, for instance, depending on the variant being considered, the ESPPRC arising may be very complex. The elementarity requirements, the use of non-robust cuts, the consideration of alternative objectives (e.g., minimization of greenhouse gas emission), to name a few attributes, has motivated the conception of several strategies to enhance CG algorithms to solve VRPs. Some of the strategies to cope with the complicating attributes listed previously are: consideration of state-space relaxation [41,42] and route-relaxations [1,2]; use of heuristics to accelerate the solution of the PS [43, 44]; development of memory mechanisms to reduce the impact of non-robust cuts in the PS [45–47], among others. In Chapter 4, we present a detailed discussion concerning all the major strategies proposed in the literature to improve the implementation of CG-based algorithms to solve VRPs. Finally, it is important to mention that having a complex PS is not a particularity of VRPs. Other complex problems such as crew scheduling [48], machine scheduling [11], and computer vision problems [13] also require enhancements in their PS solver to be addressed efficiently.

2.3 Convergence issues

Another major concern associated with CG algorithms is their poor convergence. This aspect is mainly due to the presence of degeneracy in the RMP and to the instability of the dual variables. From a primal point of view, degeneracy typically appears in applications where the columns in the matrix of coefficients of the RMP are dense, i.e., when the number of non-zeros per column is relatively high (e.g., more than 10 on average [35, 49]). This structure leads to primal bases composed of several zero-valued variables. Degenerate RMPs entail multiple optimal dual solutions, which may cause the algorithm to converge slowly (*tailing-off effect*), and the value of the RMP to remain constant for several iterations throughout the process (*plateau effect*). In turn, when starting a CG algorithm, only a few variables are present in the RMP, which yields a poor representation of the feasible dual space [50] and, hence, dual solutions of bad quality. As a consequence, dual variables tend to oscillate intensively before they converge to their optimal values (*bang-bang effect*). In fact, from one iteration to another, dual variables might move from a good dual point to a worse one, thus affecting the convergence of the algorithm.

A first family of methods devised to face degenerate problems relies on reduced forms of the RMP containing fewer constraints. The idea is to reduce the size of the primal basis, hence alleviating degeneracy. In this context, Elhallaoui et al. [49] develop the dynamic constraint aggregation (DCA) method that reduces the number of constraints in the RMP, by dynamically aggregating them. This aggregation is updated throughout the process to ensure the exactness of the algorithm. The DCA relies on the observation that, in applications

such as vehicle scheduling/routing, some of the tasks/customers often appear together in the generated columns. Later, Elhallaoui et al. [51] extend the DCA by proposing reductions at the master and pricing problem levels. From the DCA, a family of methods was derived, namely: the improved primal simplex (IPS) [28]; the row-reduced CG [52]; and the integral simplex using decomposition algorithm (ISUD) [29,53]. The IPS considers a reduced version of the problem where degeneracy in the RMP is exploited by removing constraints deemed redundant. In the row-reduced CG, the number of constraints in the RMP is limited to the number of strictly positive basic variables in the current RMP. Gauthier et al. [54] provide further analysis and insights on the DCA and the IPS, as well as on the positive edge rule, a technique that allows identifying non-degenerate pivots in the IPS. Finally, ISUD seeks to find a sequence of adjacent basic integer solutions of nonincreasing costs by performing a series of degenerate pivots before moving to a better adjacent extreme point. A limitation of all these methods is that they are limited to the solution of problems with set partitioning constraints. The recent work by Tahir et al. [53] apply ISUD to devise an integral CG heuristic capable of generating optimal or near-optimal solutions in reasonable computational time.

Another strategy described in the literature to keep the size of the RMP reasonable is to generate the constraints of the problem dynamically [55, 56]. It is noteworthy that these techniques do not constitute cutting plane algorithms since the generated constraints are not used to reinforce the formulation but rather to ensure the feasibility of the problem. The idea is to avoid redundant constraints in the formulation.

Degeneracy in the primal may also be addressed from a dual perspective. While being mostly concerned in controlling the oscillation of dual variables, stabilization techniques also help in reducing the degeneracy in CG algorithms. As discussed in the next paragraphs, most of the strategies employed to handle dual instability impact on the structure of primal problems.

An intuitive strategy to control the oscillation of dual variables is to restrain their movements to a specific region in the dual space. In the literature, this has been achieved by either 1) imposing bounds on the values of dual variables to force them to remain in the neighborhood of the incumbent dual solution [57], or 2) considering a piecewise-linear function to penalize the dual objective for moving far away from the stability center [58–60]. Algorithmically, this method consists in modifying the RMP to add artificial variables, whose costs in the objective function model the shape of the piecewise-function chosen. The changes incurred to the RMP help overcoming degeneracy as they correspond to perturbations applied to the RMP constraints [59]. Similarly, the existence of some good estimator of the optimal dual solution may be considered to alleviate the impact of bad dual values in an attempt to guide the CG process. Smoothing techniques employ dual information from previous iterations

to *correct* the dual variables obtained from the RMP at each iteration. The weighted DW decomposition proposed by Wentges [61] consists in solving the PS by considering a modified dual vector $\tilde{\pi} = \alpha\hat{\pi} + (1 - \alpha)\pi$, where $\hat{\pi}$ corresponds to the best dual vector obtained so far, and $\alpha \in [0, 1)$ is a parameter indicating the smoothing level. Neame [62] suggests to use the weighted sum over dual vectors obtained from previous iterations as the smoothing component and Pessoa et al. [63] propose self-adjusting schemes for the parameter α.

Alternatively, Rousseau et al. [64] and Gondzio et al. [65] suggest favoring the use of points located in the interior of the dual convex hull as an attempt to alleviate the harm caused by dual instability to the convergence of CG algorithms. Extreme dual values may yield columns very unlikely to be selected in an optimal solution, which may bring some noise to the algorithm convergence. Rousseau et al. [64] consider dual interior points defined as convex combinations of dual extreme points obtained by solving several randomly modified RMPs. In turn, Gondzio et al. [65] apply a primal-dual CG method in which non-optimal and well-centered solutions of the RMP are considered throughout the CG process.

In addition to modifying the implementation of a standard CG algorithm to incorporate stabilization mechanisms, one may consider some prior knowledge about the domain of the dual variables to devise inequalities to restrain the feasible dual space. A straightforward way of doing this is to replace equality constraints by inequality constraints in the RMP [21,27,64], provided that the bound of the linear relaxation is preserved. This practice reduces by half the dual space since dual variables will only assume non-negative (non-positive) values.

The concept of dual inequality (DI) has been explicitly introduced in the literature by Valério de Carvalho [66] and was later extended by Ben Amor et al. [67]. Ben Amor et al. classify DIs as *dual-optimal inequalities* (DOIs) and *deep dual-optimal inequalities* (DDOIs), depending on whether they discard any dual-optimal solution, or not. In his work, Valério de Carvalho [66] exploits the structure of bin packing (BP) and cutting stock (CS) problems to derive dual pair and subset inequalities (PIs, SIs). Gschwind and Irnich [68] come up with the notion of dynamic (D)DOIs. Instead of separating (D)DOIs statically, and incorporating them into the RMP before starting the CG process, the authors propose to separate violated (D)DOIs on the fly as a by-product of the pricing algorithm. Additionally, it is suggested to employ valid DIs that might not constitute DOIs, nor DDOIs for a given problem at hand. Although this *overstabilization* may cause (all) dual optimal solutions to be discarded, in practice, it helps in the convergence of the algorithm. Hessler et al. [69] and Gschwind et al. [70] have employed the concepts of overstabilization and dynamic/static separation of DOIs and (D)DOIs to devise tailored CG algorithms to solve vector packing problems (VPPs) and the commodity-constrained split delivery vehicle routing problem (C-SDVRP), respectively.

More recently, some authors have developed ideas in the spirit of making the use of DOIs less of a static approach. Yarkony et al. [12] propose *variant* DOIs in the context of computer vision problems. These DOIs differ from invariant (static) ones because the bounds for dual variables are computed at each iteration of CG before solving the RMP. Finally, in seeking to overcome one of the main drawbacks associated with the use of DOIs, i.e., its lack of generality to be applied to different families of problem, Lokhande et al. [71] introduce the notion of *flexible dual inequalities* when solving set-packing problems. These new DOIs do not require a full knowledge about the problem structure, but rather on the columns that are currently available at the RMP. Haghani et al. [72] extend the work by Lokhande et al. to address facility location problems modeled using set-covering formulations.

CHAPTER 3 ORGANIZATION OF THE THESIS

The objective of this thesis is to discuss and propose techniques that can be used to enhance the performance of CG-based algorithms. In this chapter, we present how this document is organized and explain how each chapter contributes to achieving the objectives described in Chapter 1.

Chapters 4, 5, and 6 form the main contributions of this thesis. Chapters 4 and 5 correspond to the two papers that have been produced during the Ph.D. Chapter 6, in turn, consists of an investigation on the use of a new stabilization technique to improve the convergence of CG algorithms.

Chapter 4 consists of a methodological survey on branch-price-and-cut (BPC) algorithms to solve different VRP variants. The motivation for this work comes from the fact that CG has been intensively applied in the last years to solve a variety of VRP variants. Yet, sometimes, it can be quite challenging for a CG practitioner to have a holistic understanding of all the tools and techniques available in the literature. In our study, we do not limit ourselves to make of a list of papers as would be more common in a literature review paper. Instead, we synthesize and highlight some of the main modeling and methodological strategies developed over the years in the context of BPCs for VRPs. The survey contains two main parts. The first part is dedicated to *generic tools*, i.e., ideas and techniques that apply to different variants of VRPs. Concepts like pricing, cutting, branching, the use of upper bounds, and stabilization techniques are discussed in this part. The second part is more problem-oriented. We focus on the *specific contributions* that have been proposed in the literature to deal with the different attributes associated with distinct VRP variants. We discuss how a general BPC algorithm can be adapted to handle characteristics such as heterogeneous fleet and multiple depots, profits, soft time windows, multiple trips, split services, time dependency, cumulative costs, environmental aspects, uncertainty, and pickups and deliveries. The content of this chapter has been recently published in *Transportation Science* [9].

In the subsequent parts of this thesis, two important issues associated with general CG algorithms are addressed: the computational burden of solving the CG pricing subproblems, and the bad convergence behavior due to the aggressive oscillation of RMP dual variables. In Chapter 5, we develop a selective pricing algorithm relying on the *arc-ng*-route relaxation proposed by Bulhões et al. [1]. This technique employs a memory mechanism to remember nodes that have been visited recently within a route. Compared to the method described in [1], the new selective algorithm allows the consideration of less restrictive dominance

rules. These new rules yield a considerable reduction in the number of non-dominated labels kept by the labeling algorithm applied to solve the pricing problem arising from the vehicle routing problem with time windows (VRPTW). The findings of this work have been recently submitted to *International Transactions in Operational Research*.

In Chapter 6, we present a new stabilization technique that helps to reduce the dual oscillation occurring in some highly degenerate COPs. The new stabilization framework employs a dynamic separation of aggregated constraints of the original RMP. The performance of the technique is tested on three different problems: vehicle routing problem with time windows, bin packing with conflicts, and multi-person pose estimation. Despite its simplicity and the structural difference between the problems, the new strategy yields significant speedups when compared with standard CG algorithms.

In Chapter 7, a general discussion is provided, where we place our results regarding other works in the literature. Finally, in Chapter 8, we summarize our work and discuss some of the limitations associated with the proposed methods. Also, we indicate some future research avenues that are compatible with the ideas developed during the execution of the Ph.D. project.

CHAPTER 4 ARTICLE 1: EXACT BRANCH-PRICE-AND-CUT ALGORITHMS FOR VEHICLE ROUTING

Authors: Luciano Costa, Claudio Contardo, and Guy Desaulniers
Published in Transportation Science, 2019 [1].

Abstract. Vehicle routing problems (VRPs) are among the most studied problems in operations research. Nowadays, the leading exact algorithms for solving many classes of VRPs are branch-price-and-cut algorithms. In this survey paper, we highlight the main methodological and modeling contributions made over the years on branch-and-price(-and-cut) algorithms for VRPs, whether they are generic or specific to a VRP variant. We focus on problems related to the classical VRP, i.e., problems in which customers must be served by several capacitated trucks, and which are not combinations of a VRP and another optimization problem.

Keywords. *Branch-price-and-cut; vehicle routing; survey; generic tools; variant-specific contributions.*

4.1 Introduction

The vehicle routing problem (VRP) is one of the most studied problems in operations research. It was introduced by Dantzig and Ramser [16] to solve a practical problem of delivering gasoline from a bulk terminal to service stations. Its basic version, the capacitated VRP (CVRP), consists of finding feasible routes to visit a set of customers in such a way that each customer is visited exactly once to completely satisfy its demand. A route is said to be feasible if it starts and ends at a given depot and if the sum of the demands of the visited customers does not exceed vehicle capacity. The most common objective for this problem is the minimization of the traveling costs, which is often assumed to be proportional to the total traveled distance.

Over the years, several extensions to the CVRP have been proposed. Most of them are inspired by real-life applications and consider different attributes, namely: start of service time windows at the customers, pickup and delivery requests, split deliveries, multiple depots, heterogeneous fleet, etc. For a detailed classification of the VRP variants, see Eksioglu et al. [73]; Irnich et al. [74]; and Braekers et al. [75].

VRPs have been tackled by (meta)heuristics and exact algorithms. Despite the important

[1] Available at [9].

role played by heuristics when dealing with real-life problems, we do not review them in this survey. Most of the exact algorithms in the VRP literature rely on branch-and-bound (BB) to explore implicitly the solution space. Because the performance of the BB algorithms depends on the quality of the bounds obtained throughout the tree, it is common practice to employ some techniques to improve the quality of these bounds. BB algorithms can be combined with the generation of cutting planes, forming the so-called branch-and-cut (BC) algorithms, or with column generation (CG), resulting in branch-and-price (BP) algorithms. When both techniques are exploited simultaneously, this leads to branch-price-and-cut (BPC) algorithms.

For many years, BC algorithms were deemed the best algorithms to address VRPs [76, 77]. However, since the seminal work of Desrosiers el al. [20], a lot of effort has been put into the development of efficient BP and BPC algorithms, which made them, nowadays, the leading algorithms for solving many classes of VRPs. BP algorithms are BB algorithms in which the linear relaxations are solved by means of CG. CG is an iterative procedure that can tackle linear programs containing a huge number of variables (see [7, 14, 15]). In the context of VRPs, it relies on a subproblem (called the pricing problem) to generate routes dynamically and on a master problem (MP) to select the best ones.

BPC can be seen as a generic framework to solve VRPs. In this respect, several works have proposed generic algorithmic enhancements that are applicable to most VRPs. On the other hand, tailored BPC algorithms, including specific tools/procedures to handle the particularities of a VRP variant, have also been devised. In this context, Feillet [21] illustrates the importance of methodological studies on CG-based algorithms by pointing out some reasons that can make these algorithms hard to understand and reproduce, namely: 1) the inherent complexity associated with the methodology and the large literature in this field; 2) the variety of perspectives from which these algorithms can be explored; and 3) the lack of comprehensive descriptions of these algorithms.

To the best of our knowledge, there is no work in the literature discussing specificities of the BP/BPC algorithms designed for different classes of VRPs. The only paper providing a related analysis is that of Baldacci et al. [78]. Nevertheless, they limit their analysis to the CVRP and the VRP with time windows (VRPTW). The present paper is broader in the sense that it provides a survey of the methodological developments proposed for BP/BPC algorithms applied to a wide variety of VRPs. Even if several transportation problems can be modeled/addressed as routing problems, we limit our discussion to problems related to the CVRP, i.e., problems in which customers/requests must be served by several capacitated trucks, and that are not combinations of a VRP and another optimization problem. There-

fore, problems such as vehicle scheduling, inventory routing, ship routing, location-routing, etc., are out of scope.

Our goal for this survey is to highlight the main methodological and modeling contributions made over the years on BP/BPC algorithms for VRPs, whether they are generic or specific to a VRP variant. Consequently, we do not provide an exhaustive review of all papers presenting a BP/BPC algorithm for a VRP. Given the large number of VRP variants covered and papers cited, we have also chosen to present and discuss no computational results.

This paper is structured as follows. The remainder of this section defines the VRPTW that will serve as our main VRP example and provides a mathematical formulation for it. Section 4.2 presents the main components of a basic BPC algorithm. Generic tools for improving the performance of BPC algorithms are discussed in Section 4.3. Contributions specific to VRP variants are reviewed in Section 4.4. Finally, in Section 4.5, we draw some conclusions.

4.1.1 Problem description

For the sake of simplicity, when presenting the main concepts and ingredients of BPC algorithms, the VRPTW is used as an example. The choice of this problem comes from the fact that, while it is a relatively simple problem, solving it by BPC is still quite challenging. In fact, many state-of-the-art techniques incorporated into BPC algorithms have been proposed for the VRPTW.

The VRPTW can be formally defined as follows. Let $G = (V, A)$ be a complete and directed graph, where $V = V' \cup \{0, n+1\}$ is the set of vertices and A the set of arcs. V' is the set of n customers. Vertices 0 and $n+1$ represent the depot at the start and the end of a route, respectively. Each customer $i \in V'$ is associated with a demand $q_i > 0$, a service time $s_i > 0$ and a time window $[e_i, l_i]$, with $0 \leq e_i \leq l_i$, which specifies the earliest and the latest time at which service can start at customer i. We consider $q_0 = s_0 = e_0 = q_{n+1} = s_{n+1} = e_{n+1} = 0$ and $l_0 = l_{n+1} = T$, where T represents the horizon length. The traveling cost and the traveling time associated with each arc $(i, j) \in A$ are denoted by c_{ij} and t_{ij}, respectively. Besides, an unlimited fleet of homogeneous vehicles with capacity Q is available at the depot. The VRPTW consists of designing feasible routes such that each customer $i \in V'$ is visited exactly once by a route and the sum of the costs of the routes is minimized. A feasible route corresponds to an elementary path $r = (i_0 = 0, i_1, \ldots, i_{k-1}, i_k = n+1)$ in G such that the vehicle capacity is not exceeded, i.e., $\sum_{j=0}^{k} q_{i_j} \leq Q$, and the time windows at the visited vertices are met. The latter conditions can be verified by computing recursively the start of

service time t_{i_j} at every visited vertex i_j, $j \in \{0, 1, \ldots, k\}$, as follows

$$\begin{aligned} t_{i_0} &= e_{i_0} \\ t_{i_{j+1}} &= \max\{e_{i_{j+1}}, t_{i_j} + s_j + t_{i_j,i_{j+1}}\}, \quad \forall j \in \{0, 1, \ldots, k-1\}, \end{aligned}$$

where the maximum function is used to model the possibility that a vehicle arrives before the opening of a time window and waits until its opening to start service. The time windows are met if $t_{i_j} \in [e_{i_j}, l_{i_j}]$, $\forall j \in \{0, 1, \ldots, k\}$. Given these feasibility conditions, some arcs in A can be discarded to yield arc set $A = \{(i, j) \in V \times V \mid q_i + q_j \leq Q, e_i + s_i + t_{ij} \leq l_j\} \setminus \{(0, n+1), (n+1, 0)\}$. Finally, the cost c_r of route r is given by $c_r = \sum_{j=0}^{k-1} c_{i_j,i_{j+1}}$.

The VRPTW can be formulated in terms of binary arc-flow variables x_{ij} to indicate whether a vehicle travels or not along a given arc $(i, j) \in A$ (for more details, see Desaulniers et al. [79]). Arc-flow formulations have the advantage of containing a polynomial number of variables, but they provide, in general, weak linear relaxations. An alternative way of modeling the VRPTW is by means of a set partitioning formulation [80], which we present in the next section. Arc-flow and set partitioning formulations are often referred to as *compact* and *extended* formulations, respectively.

As discussed previously, the VRPTW is defined over the directed graph G. Nevertheless, other VRP variants such as the CVRP can be formulated using an undirected graph. For the sake of clarity, we will assume a directed graph G throughout this paper unless otherwise specified.

4.1.2 Set partitioning formulation

Let Ω be the set of all feasible routes, i.e., elementary routes that satisfy vehicle capacity and time windows. A binary parameter a_i^r specifies whether or not customer $i \in V'$ is visited on a route $r \in \Omega$. For each route $r \in \Omega$, define a binary variable λ_r which is equal to 1 if r is selected in the solution and 0 otherwise.

The VRPTW can be formulated as the following set partitioning model:

$$\min \quad \sum_{r \in \Omega} c_r \lambda_r \tag{4.1}$$

$$s.t. \quad \sum_{r \in \Omega} a_i^r \lambda_r = 1, \qquad \forall i \in V' \tag{4.2}$$

$$\lambda_r \in \{0, 1\}, \qquad \forall r \in \Omega. \tag{4.3}$$

The objective function (4.1) aims at minimizing the total routing cost. Constraints (4.2)

ensure that each customer is visited exactly once. Finally, constraints (4.3) define the domain of the variables.

A big advantage of formulating the VRPTW as (4.1)–(4.3) is that some of the constraints (e.g., the time windows) do not need to be explicitly expressed in the formulation: they are rather implicit from the definition of set Ω. Consequently, set partitioning formulations typically provide better lower bounds than those obtained with compact formulations. Nevertheless, they admit a huge number of variables, which makes it impossible to handle them all at once. Fortunately, BPC is a suitable technique to overcome this drawback.

4.2 Components of a basic BPC algorithm

In this section, we present the components of a basic BPC algorithm. For further details on BPC algorithms, the reader is referred to Barnhart et al. [14], Desaulniers et al. [7], Lübbecke and Desrosiers [15], and Feillet [21], where important insights about the development of BPC algorithms are highlighted.

As stated in the introduction, a BPC algorithm is a BB algorithm where the lower bounds are computed by CG and the cutting planes are added to strengthen the linear relaxations encountered in the search tree. In this context, such a linear relaxation is called a MP which is solved by CG. CG is an iterative algorithm that solves at each iteration a restricted MP (RMP) and a pricing problem. The RMP is a linear program defined as the MP restricted to a subset of its route variables. The pricing problem can be an arbitrary optimization problem which is solved either to find new variables to add to the current RMP or to prove that the current RMP solution can be extended (by setting all non-generated variables to zero) to yield an optimal solution to the MP.

Below, we discuss the MP, the pricing problem, cutting planes, and branching decisions when BPC is used to solve model (4.1)–(4.3) of the VRPTW.

4.2.1 The master problem

At the root node of the BB search tree, the MP is given by:

$$\min \quad \sum_{r\in\Omega} c_r\lambda_r \tag{4.4}$$

$$s.t. \quad \sum_{r\in\Omega} a_i^r\lambda_r = 1, \qquad \forall i \in V' \tag{4.5}$$

$$\lambda_r \geq 0, \qquad \forall r \in \Omega'. \tag{4.6}$$

In model (4.4)-(4.6), the variables λ_r, $r \in \Omega$, are implicitly upper bounded by one through constraints (4.5). Note that cuts can be added to this MP (see Section 4.2.3) and that, at other nodes of the search tree, branching decisions can modify it (see Section 4.2.4).

An optimal solution to (4.4)-(4.6) provides a lower bound at the root node of the search tree. When applying CG to solve the MP, a RMP, obtained by replacing Ω with a relatively small subset $\Omega' \subseteq \Omega$ in (4.4)–(4.6), is solved at each iteration. The solution process yields an optimal primal solution for this RMP and a complementary dual solution $(\pi_i)_{i \in V'}$, where π_i is the dual variable associated with constraint (4.5) indexed by $i \in V'$. Extending this primal solution to the MP (i.e., by setting $\lambda_r = 0$ for all $r \in \Omega \setminus \Omega'$) results in an optimal MP solution if the reduced cost $\bar{c}_r = c_r - \sum_{i \in V'} a_i^r \pi_i$ of λ_r is nonnegative for every route $r \in \Omega \setminus \Omega'$. The role of the pricing problem (see Section 4.2.2) is to find routes with a negative reduced cost or to prove that none exist. When negative reduced cost routes are identified, they are added to subset Ω' before starting a new iteration. Otherwise, the CG process stops with an optimal solution to the MP.

In their BPC algorithms, many authors have employed branching rules and cutting planes defined from arc flows $(x_{ij})_{(i,j) \in A}$. These arc flows can be easily computed from a solution to the MP using the following expression

$$x_{ij} = \sum_{r \in \Omega} b_{ij}^r \lambda_r, \tag{4.7}$$

where b_{ij}^r is a binary parameter indicating whether or not arc $(i, j) \in A$ is traversed by route $r \in \Omega$.

4.2.2 The pricing problem

The pricing problem consists of finding a feasible route $r \in \Omega$ with a negative reduced cost $\bar{c}_r$. This problem can be modeled as an elementary shortest path problem with resource constraints (ESPPRC) on graph G. Resources are quantities (e.g. time, load, etc.) that are used to assess the feasibility of a route or to compute the cost of a route. Their values vary along a path according to so-called *resource extension functions* (REFs), which are defined for each resource and each arc in G. In addition, at each vertex of G, *resource windows* restrict the values that can be taken by the resources. Efficient dynamic programming algorithms for solving the pricing problem might be designed when the REFs present the following two properties: 1) they only depend on the resource consumptions and on the values associated with the arc and its tail vertex, allowing the computation of resource values at each vertex; and 2) they are non-decreasing with respect to the resource values, ensuring the possible

application of a dominance rule to reduce route enumeration [8, 81, 82]. Irnich [82] shows how REFs can be used to model complex route costs and constraints arising in VRPs and how they allow some algorithmic procedures to be employed, namely: path representations, efficient cost computations, and constant time feasibility checking performed while building/concatenating paths. More general REFs are presented in Section 4.4, where problems with many attributes are discussed.

When solving the ESPPRC, a modified cost $\bar{c}_{ij} = c_{ij} - \pi_i$ is associated with each arc (i, j) in G, with $\pi_0 = 0$. This ensures that the cost of a path in G corresponds to the reduced cost of the corresponding route $r \in \Omega$:

$$\bar{c}_r = c_r - \sum_{i \in V'} a_i^r \pi_i = \sum_{(i,j) \in A} c_{ij} b_{ij}^r - \sum_{i \in V'} a_i^r \pi_i = \sum_{(i,j) \in A} \bar{c}_{ij} b_{ij}^r. \tag{4.8}$$

Unlike the classical shortest path problem [83], which is polynomially solvable, solving the ESPPRC is not an easy task. Because the arc costs $\bar{c}_{ij}$ may be negative, there might exist negative cost cycles. Dror [38] showed that the ESPPRC is $\mathcal{NP}$-hard in the strong sense. For this reason, when developing BPC algorithms, some authors have replaced the ESPPRC by the shortest path problem with resource constraints (SPPRC) as the pricing problem (see [81]). The SPPRC is a relaxation of the ESPPRC that allows the generation of paths with cycles (i.e., routes with multiple visits to the same customer). In this case, the set Ω of feasible routes is enlarged to include these routes and the meaning of parameter a_i^r (resp. b_{ij}^r) changes to represent the number of times customer $i \in V'$ is visited (resp. arc $(i, j) \in A$ is traversed) by a route $r \in \Omega$. Note that, when the coefficients a_i^r can take positive integer values, formulation (4.1)–(4.3) remains valid because constraints (4.2) exclude integer solutions containing non-elementary routes. On the one hand, the SPPRC is easier to solve than the ESPPRC as it can be solved by a pseudo-polynomial algorithm as shown by Desrochers et al. [84] who devised the first BP algorithm for a VRP. On the other hand, solving a SPPRC as the pricing problem may significantly reduce the quality of the lower bounds provided by the MP. We discuss in Section 4.3 other relaxations that have been proposed in the literature, seeking a better compromise between bound quality and total computational time.

4.2.3 Cutting planes

Despite the fact that extended formulations can provide better lower bounds than those obtained with compact formulations, these bounds might still be too weak to yield an efficient algorithm. Thus, when designing BP algorithms for tackling complex VRPs, it is common

practice to reinforce the MP with valid inequalities.

As discussed in Section 4.2.1, any feasible solution to an extended model can be converted into a solution to a corresponding compact model. Thus, by applying (4.7) to a cutting plane of the form $\sum_{(i,j)\in A} \beta_{ij} x_{ij} \leq \beta_0$, one can rewrite this inequality in terms of the route variables as follows:

$$\sum_{r\in\Omega} \sum_{(i,j)\in A} \beta_{ij} b_{ij}^r \lambda_r \leq \beta_0, \tag{4.9}$$

and use it to strengthen the MP.

According to the classification of Poggi de Aragão and Uchoa [33], cuts of the form (4.9) are called *robust cuts* because they do not increase the complexity of the pricing problem. Indeed, because they can be expressed in terms of the arc-flow variables, their dual values can be directly incorporated into the modified cost $\bar{c}_{ij}$ of each arc $(i,j) \in A$ as follows. Let ρ be the dual variable associated with (4.9). The reduced cost $\bar{c}_r$ of a variable λ_r, $r \in \Omega$, becomes:

$$\bar{c}_r = c_r - \sum_{i\in V'} a_i^r \pi_i - \rho \sum_{(i,j)\in A} \beta_{ij} b_{ij}^r = \sum_{(i,j)\in A} (c_{ij} - \pi_j - \rho\beta_{ij}) b_{ij}^r = \sum_{(i,j)\in A} \bar{c}_{ij} b_{ij}^r, \tag{4.10}$$

by setting $\bar{c}_{ij} = c_{ij} - \pi_j - \rho\beta_{ij}$ for all arcs $(i,j) \in A$. Note that several robust cuts, each with a dual value, can be handled simultaneously. Note also that not all cuts defined in terms of the arc-flow variables are useful in a BPC algorithm because many families of cuts are implied by the definition of the routes.

Valid inequalities can also be defined directly in terms of the route variables. Let $\sum_{r\in\Omega} \beta_r \lambda_r \leq \beta_0$ be such a generic cut and $\sigma < 0$ the dual variable associated with it. Then, the reduced cost of a route $r \in \Omega$ rewrites as follows:

$$\bar{c}_r = \sum_{(i,j)\in A} \bar{c}_{ij} b_{ij}^r - \beta_r \sigma. \tag{4.11}$$

Since the variable coefficients β_r, $r \in \Omega$, are not necessarily defined as linear functions of the arc flows, the dual value σ may not be directly transferred to the modified arc costs. Such cuts are, thus, said to be *non-robust* and handling their dual values increases the complexity of the pricing problem as additional unrestricted resources are required in the ESPPRC (see [36]). It is important to note that, regardless of this increased complexity, non-robust cuts have a great potential for reducing the integrality gaps.

In Sections 4.3 and 4.4, we discuss various families of cuts that were incorporated in BPC

algorithms for solving VRPs. For more details about cutting planes in BPC algorithms, see the general framework introduced by Desaulniers et al. [36].

4.2.4 Branching decisions

To derive integer solutions, branching is the ultimate operation to perform in a BPC algorithm. In this section, we give an overview of branching decisions in BPC algorithms for VRPs. Details on the most common ones are discussed in Section 4.3.3.

If we consider model (4.1)–(4.3), it seems natural to branch on the route variables, that is, to choose a variable $\lambda_r \in \Omega$ such that $\lambda_r \in (0, 1)$ in the current MP solution and to impose $\lambda_r = 0$ on one branch and $\lambda_r = 1$ on the other. Imposing the latter decision is straightforward. On the other hand, imposing $\lambda_r = 0$ is not easy and often ineffective. Indeed, one must also prevent route r to be re-generated by the pricing problem which becomes a more complex ESPPRC, namely, a ESPPRC with forbidden paths [85]. Furthermore, fixing a unique route variable to zero does not restrict much the solution space and typically yields an unbalanced search tree.

To alleviate this drawback, more aggressive branching decisions can be devised by considering the arc-flow variables of the compact formulation, instead of the route variables of the extended formulation. These branching decisions are defined through relation (4.7), and have the advantage of involving many route variables simultaneously beside being robust. Examples of branching decisions performed with respect to the arc-flow variables are: branching directly on the flow on an arc, branching on the number of vehicles leaving the depot, and branching on the flow into/out of a set of vertices (see Section 4.3.3).

4.3 Generic tools

In this section, we present ideas applicable to different variants of VRPs. The topics discussed are classified as follows: pricing, cutting, branching, using upper bounds, and stabilizing dual values.

4.3.1 Pricing

For most VRPs, the pricing problem is an ESPPRC or a relaxation of it that is typically solved by a labeling algorithm. We start by presenting basic labeling algorithms before discussing various ESPPRC relaxations and speed up tools.

Basic labeling algorithms.

For clarity reasons, let us start by describing a labeling algorithm for the SPPRC pricing problem arising for the VRPTW. In this algorithm, labels represent partial paths in G which all begin at the origin depot, i.e., vertex 0. Starting from vertex 0, labels are extended through the network, passing by some of the customers, until the destination depot (vertex $n+1$) is reached. In its basic version, a label L representing a path p is a tuple $L = (v, \bar{c}, q, t)$, where $v \in V$ is the last vertex in p; $\bar{c}$ the reduced cost of p; q the cumulated load along p; and t the earliest time at which service can start at vertex v if p is used to reach v. A label L also stores its predecessor label to allow to build complete paths once the algorithm is finished.

Let $L_0 = (0, 0, 0, 0)$ be the initial label at vertex 0 and denote by $v(L)$, $\bar{c}(L)$, $q(L)$, $t(L)$ the components of a label L associated with a path $p(L)$ ending at vertex $v(L) = i$. If one performs a label extension along an arc (i, j), a new label L' is obtained by applying the following relations:

$$v(L') = j \tag{4.12}$$

$$\bar{c}(L') = \bar{c}(L) + \bar{c}_{ij} \tag{4.13}$$

$$q(L') = q(L) + q_j \tag{4.14}$$

$$t(L') = \max\{t(L) + t_{ij}, a_j\}. \tag{4.15}$$

This new label represents path $p(L') = p(L) \oplus (i, j)$, where we use $\oplus$ as a concatenation symbol. After performing this extension, the feasibility of path $p(L')$ is verified by checking if the cumulated resources are within the resource windows at vertex j. Path $p(L')$ is feasible if $q(L') \in [0, Q]$ and $t(L') \in [a_j, b_j]$. Otherwise, it is infeasible and label L' is discarded.

The efficiency of a labeling algorithm depends on its ability to eliminate non-useful paths. For this reason, labels are compared through a *dominance rule* in order to identify and remove unnecessary labels. Let L be a label and $\mathscr{E}(L)$ be the set of feasible path extensions of path $p(L)$, i.e., a path w in G starting at vertex $v(L)$ belongs to $\mathscr{E}(L)$ if $p(L) \oplus w$ is a feasible path. A label L' can be discarded if there exists a set of labels $\mathcal{L} = \{L_1, L_2, \ldots, L_{|\mathcal{L}|}\}$ such that, for all feasible extensions $\omega \in \mathscr{E}(L')$, there exists a label $L \in \mathcal{L}$ for which: i) $p(L) \oplus \omega$ is feasible and ii) the reduced cost of $p(L) \oplus \omega$ is less than or equal to that of $p(L') \oplus \omega$. These two conditions specify that L' is not required to describe the set of Pareto-optimal paths ending at any vertex $v \neq v(L)$ and can, thus, be discarded. Note that conditions i) and ii) are too difficult to be assessed because it would require an enumeration of all paths and extensions. Therefore, these two conditions are replaced by the following sufficient conditions. Let L_1

and L_2 be two labels such that $v(L_1) = v(L_2)$. L_1 is said to dominate L_2 if:

$$\bar{c}(L_1) \leq \bar{c}(L_2) \tag{4.16}$$

$$q(L_1) \leq q(L_2) \tag{4.17}$$

$$t(L_1) \leq t(L_2). \tag{4.18}$$

This dominance rule is valid because the REFs defined by the right-hand side of (4.13)-(4.15) are non-decreasing with respect to $\bar{c}(L)$, $q(L)$ and $t(L)$, respectively (see [8]). As discussed in the next sections, this dominance rule needs to be modified to accommodate specificities related to different pricing problems.

Depending on the data structure used to store the labels and the order in which the labels are extended, the above labeling algorithm can have a pseudo-polynomial time complexity (see, e.g., [40, 81]). This is an important advantage over the algorithms that can solve the ESPPRC which is $\mathcal{NP}$-hard in the strong sense [38].

Despite the success achieved by BPC algorithms using a SPPRC pricing problem, elementarity constraints may have a huge impact on the quality of the bounds generated by the MP [86]. For this reason, authors have turned their attention to the development of efficient algorithms that can either solve the ESPPRC or provide the so-called elementary bounds. One of the first attempts in this direction was made by Beasley and Christofides [87], who propose a dynamic programming algorithm that manages elementarity through a *customer resource vector* ($\mathcal{E}$) which stores the customers visited along a path. With this new label definition, the dominance rule must also include:

$$\mathcal{E}(L_1) \subseteq \mathcal{E}(L_2) \tag{4.19}$$

in addition to relations (4.16)–(4.18). Condition (4.19) stipulates that label L_1 cannot dominate label L_2 if $p(L_1)$ contains a vertex j that has not been visited in $p(L_2)$, i.e., $j \in \mathcal{E}(L_1) \setminus \mathcal{E}(L_2)$. In this case, there might exist feasible extensions in $\mathscr{E}(L_2)$ which include vertex j and decrease the reduced cost. Given that these extensions are not in $\mathscr{E}(L_1)$, L_2 cannot be discarded. Improvements on this dominance rule are proposed by Chabrier [88].

Feillet et al. [86] present another exact algorithm to solve the ESPPRC. They extend the idea of Beasley and Christofides [87] by introducing a new label definition that replaces set $\mathcal{E}$ with a set of unreachable vertices $\mathcal{U}$. A vertex j is added to $\mathcal{U}(L)$ if it is visited along path $p(L)$ or if it becomes unreachable due to resource limitations, i.e., $q(L) + q_j > Q$ or $t(L) + t_{ij} > b_j$.

Condition (4.19) then becomes:

$$\mathcal{U}(L_1) \subseteq \mathcal{U}(L_2). \tag{4.20}$$

Through computational tests, Feillet et al. [86] show that using elementary routes can yield much better lower bounds at the root node of the search tree.

ESPPRC relaxations.

Given the complexity of the ESPPRC, many authors have designed BPC algorithms that rely on relaxations of the ESPPRC. In most cases, the elementarity requirements are totally or partially relaxed. Totally relaxing them gives the SPPRC which allows any cycle to be part of a generated path. We discuss below several stronger relaxations that differ by the families of cycles they forbid. Note that, although elementarity can also be enforced via cuts (e.g., strong degree and k-cycle-elimination cuts), we focus here on strategies applied to labeling algorithms. Cutting planes are discussed in Section 4.3.2. The reader is referred to Contardo et al. [89] for a complete analysis on different ways to reach the lower bound achieved with only elementary routes. Christofides et al. [90] introduce a technique, called k-cycle elimination, that has been largely employed to avoid cycles. It consists of forbidding cycles of length k or less to be formed while solving the SPPRC. This relaxation is known as the k-cyc-SPPRC. Solving it not only increases the lower bound quality but can also accelerate the algorithm running time. For the sake of clarity, we present separately the case $k = 2$. In fact, the 2-cyc-SPPRC has been largely applied in the literature, as well as combined with state-of-the-art techniques.

2-cycle elimination: In the 2-cyc-SPPRC, cycles of the form $i - j - i$ are forbidden. The use of 2-cycle elimination is particularly interesting because it yields stronger bounds without changing the complexity of the labeling algorithm (see [91]).

In fact, Christofides et al. [90] show that to prevent cycles of length two, it is sufficient to keep, for each possible resource state, at most two least-cost labels L_1 and L_2 whose predecessor labels are associated with different vertices. Two examples on the use of 2-cycle elimination can be found in: Desrochers et al. [84], who introduce the first exact algorithm capable of solving 100-customer VRPTW instances, and Fukasawa et al. [43], who propose a BPC algorithm to solve the CVRP and generated q-routes with the pricing problem. A q-route is a path that respects vehicle capacity but may contain cycles.

k-cycle elimination: To the best of our knowledge, k-cycle elimination with $k \geq 3$ has only been tested by Irnich and Villeneuve [92] and Fukasawa et al. [43]. Irnich and Villeneuve propose a new representation of partial paths which is based on the notion of *set form*. The idea is to use a vector to represent all the paths in which a given customer $i \in V'$ is placed at the jth position in the path. In the case of the k-cyc-SPPRC, one uses a vector of length k to store the k last vertices of a path. Then, a set $\mathcal{H}$ formed by the union of all required set forms is defined to represent all k-cycle elimination constraints. For example, for the 4-cyc-SPPRC, consider a label L such that the four last visited vertices are (a, b, c, v). Then, $\mathcal{H}(L)$ contains the set forms: $(v, \cdot, \cdot, \cdot)$, $(\cdot, v, \cdot, \cdot)$, $(\cdot, \cdot, v, \cdot)$, $(\cdot, \cdot, \cdot, v)$, $(c, \cdot, \cdot, \cdot)$, $(\cdot, c, \cdot, \cdot)$, $(\cdot, \cdot, c, \cdot)$, $(b, \cdot, \cdot, \cdot)$, $(\cdot, b, \cdot, \cdot)$, $(a, \cdot, \cdot, \cdot)$. Any extension of $p(L)$ whose beginning matches one of these forms is forbidden as it would create a cycle of length four or less. Given that the cardinality of $\mathcal{H}(L)$ is quadratic in k, this approach may rapidly become prohibitive as k increases. Fukasawa et al. [43] use this path representation to assess the impact of eliminating cycles with length four or less in their BPC algorithm for the CVRP.

Partial elementarity: Desaulniers et al. [44] introduce another ESPPRC relaxation called the partially ESPPRC. It can be seen as an ESPPRC that requires elementarity only for a subset $\mathcal{E}_{max}$ of customers, whose maximal cardinality is determined *a priori*. Set $\mathcal{E}_{max}$ is built dynamically from scratch by adding customers that are visited more than once in a route of a MP optimal solution. Each time that a customer i is added to $\mathcal{E}_{max}$, the labeling algorithm is adjusted to forbid multiple visits to customer i and all columns visiting multiple times this customer are removed from the current RMP. The MP is then re-optimized and other customers might be added to $\mathcal{E}_{max}$. Note that, if the maximal cardinality of $\mathcal{E}_{max}$ is less than n, there is no guarantee that elementary routes will be obtained at the end of the algorithm.

ng-path relaxation: Currently, state-of-the-art BPC algorithms use the *ng*-path relaxation in their pricing procedures. The *ng*-path concept proposed by Baldacci et al. [2] relies on the definition of a neighborhood (also called *ng*-set) $\mathcal{N}_i$ for each customer $i \in V'$. For a given integer parameter $\Delta \geq 0$, a neighborhood $\mathcal{N}_i \subseteq V'$ contains the Δ closest customers to i and vertex i itself. Different criteria (e.g., distance, time) can be used to define the proximity of the customers. An *ng*-path is not necessarily elementary: it can contain a cycle starting and ending at a vertex j if and only if there exists a vertex i in this cycle such that $j \notin \mathcal{N}_i$. Hence, such a cycle is disallowed if and only if $j \in \mathcal{N}_i$ for every vertex i it contains. In a labeling algorithm, the neighborhoods of the customers visited along a path $p(L) = (0, i_1, \ldots, i_k)$ represented by a label L are used to form a set $\Pi(L) \subseteq V'$ of customers to which L cannot be extended without violating the *ng*-path cycling restrictions. Let $V(L) = \{i_1, \ldots, i_k\}$ be

the customers visited in $p(L)$. Then, $\Pi(L)$ is given by:

$$\Pi(L) = \{i_u \in V(L) : i_u \in \bigcap_{s=u}^{k} \mathcal{N}_{i_s}\} \tag{4.21}$$

and is called the *memory* of $p(L)$. Contrarily to sets $\mathcal{E}(L)$ and $\mathcal{U}(L)$, once a vertex j enters a set $\Pi(L)$, it may leave the memory when $p(L)$ is extended, i.e., j can be *forgotten* and revisited.

The labeling algorithm for the ESPPRC described in Subsection 4.3.1 can be adapted to solve this new relaxation denoted *ng*-SPPRC. In the label definition, set $\mathcal{E}$ is replaced by set Π. A label L is only extended over an arc (i, j) if $j \notin \Pi(L)$. When a new label L' is created, set $\Pi(L')$ is computed as follows:

$$\Pi(L') = (\Pi(L) \cap \mathcal{N}_j) \cup \{j\}. \tag{4.22}$$

In the dominance rule, condition (4.19) is replaced by:

$$\Pi(L_1) \subseteq \Pi(L_2). \tag{4.23}$$

Note that set Π can be enriched by adding unreachable vertices with regard to vehicle capacity, time windows, or other side constraints. Thus, if $\mathcal{F}$ is the set of all customers where an extension is not feasible, relation (4.23) can also be replaced by:

$$\mathcal{F}(L_1) \subseteq \mathcal{F}(L_2). \tag{4.24}$$

Because the cardinality of Π is less than or equal to Δ, condition (4.24) is less restrictive than (4.20) and, consequently, more labels might be discarded than in the labeling algorithm for the ESPPRC. When used as a pricing problem, the *ng*-SPPRC produces lower bounds that are between those derived with the SPPRC and the ESPPRC: if $\Delta = 0$, then $\mathcal{N}_i = \{i\}$, $\forall i \in V'$, and the *ng*-SPPRC is the SPPRC; if $\Delta = |V'| - 1$, then $\mathcal{N}_i = V'$, $\forall i \in V'$, and the *ng*-SPPRC is the ESPPRC. Note that, although unlikely, cycles of length two can be formed when solving the *ng*-SPPRC except when $\mathcal{N}_i = V'$, $\forall i \in V'$. According to Contardo and Martinelli [93], the advantage of solving the *ng*-SPPRC as an alternative to the k-cyc-SPPRC can be attributed to the fact that in the former, cycles are measured in terms of distance, while in the latter, they are measured according to the number of visited customers. Given that long cycles in terms of distance have not much chance to appear in a MP optimal solution, the *ng*-SPPRC represents a powerful relaxation. In fact, Poggi and Uchoa [94] show empirically on some difficult CVRP instances that using the *ng*-SPPRC with $\Delta = 8$ as the

pricing problem yields lower bounds similar to those achieved with the 5-cyc-SPPRC, but requires computational times that are comparable to those obtained with the 4-cyc-SPPRC.

An important parameter in the *ng*-SPPRC is Δ, the size of the neighborhoods, which is normally defined *a priori* and interferes on the labeling algorithm complexity. On the one hand, the larger the value of Δ, the closer to the ESPPRC the *ng*-SPPRC becomes. On the other hand, the algorithm complexity increases exponentially with the value of Δ. Baldacci et al. [2] show that $\Delta = 8$ represents a good trade-off between lower bound quality and computing time for the VRPTW, whereas Pecin et al. [46] use $\Delta \in \{10, 20\}$ for Solomon's and Gehring and Homberger's VRPTW datasets. Finally, note that better-tailored neighborhoods can be defined in terms of arcs as suggested by Bulhões et al. [1].

Speed up tools.

Given that the pricing problem is solved many times in a BPC algorithm, accelerating the solution of the pricing problem is a critical aspect for improving the performance of the algorithm [95]. For this reason, the following techniques are proposed to accelerate its solution.

Decremental state-space relaxation: Christofides et al. [96] introduce the concept of state-space relaxation which consists of relaxing some constraints of a problem and devising an efficient algorithm for solving the resulting relaxation to produce lower bounds. When applied to the pricing problem, this approach tends to generate infeasible paths that may yield poor lower bounds. Seeking to improve the bounds provided in this way, some authors observe that by dynamically increasing the set of customers subject to elementarity restrictions, elementary routes might be obtained in reduced computational times. In this context, Boland et al. [41] and Righini and Salani [42] propose independently a technique called *decremental state-space relaxation* (DSSR) which includes a mechanism for re-inserting dynamically constraints that were previously relaxed.

For the ESPPRC, a DSSR algorithm provides at each iteration a lower bound on its optimal value and works as follows. Initially, a SPPRC is solved and, if the least-cost path is cycle-free, it is optimal for the ESPPRC. Otherwise, the customer(s) visited more than once in this path are added to a set $\hat{\mathcal{E}}$. In the next iteration, one solves a more restricted SPPRC where elementary conditions are only imposed to the vertices in $\hat{\mathcal{E}}$. This procedure continues until an elementary route is found or the optimal path has a non negative (reduced) cost. Moreover, in the context of a CG algorithm, the procedure can be stopped if the labeling algorithm produces several paths and at least one of them is elementary and has a negative cost. To improve the performance of DSSR, Desaulniers et al. [44] initialize the set $\hat{\mathcal{E}}$ at a

CG iteration to the final set of the previous iteration as they observed during their tests a large intersection between the sets $\hat{\mathcal{E}}$ generated in consecutive CG iterations.

Martinelli et al. [97] combine the concept of *ng*-paths and DSSR. In their algorithm, neighborhoods $\mathcal{N}_i$, $i \in V'$, are computed *a priori*, but are not directly used. Instead, subsets $\tilde{\mathcal{N}}_i \subseteq \mathcal{N}_i$, $i \in V'$, are considered in the labeling algorithm. These subsets are initially empty and augmented when the labeling algorithm returns an optimal path that contains a forbidden cycle with respect to the neighborhoods $\mathcal{N}_i$ $i \in V'$. Contardo and Martinelli [93] reinforce the sets $\tilde{\mathcal{N}}_i$ by enlarging them with the addition of vertices forming cycles of length two, even if they were not in the original sets $\mathcal{N}_i$. This approach ensures that the bounds obtained are at least as strong as those of the *ng*-SPPRC with 2-cycle elimination. Contardo et al. [89] devise a similar approach where there are no initial sets $\mathcal{N}_i$, $i \in V'$, the sets $\tilde{\mathcal{N}}_i$, $i \in V'$, are initialized with a small number of closest customers and they are updated (up to a maximum size per neighborhood) only once the MP is solved, using customers that are visited more than once in a route that is part of the MP optimal solution. Finally, dynamic neighborhoods whose size can increase or decrease are proposed in Bulhões et al. [1].

Bidirectional labeling.

Proposed by Righini and Salani [98], *bidirectional labeling* consists of propagating labels in both directions (from vertex 0 to its successors and from vertex $n+1$ to its predecessors) and then joining forward and backward labels to produce complete feasible routes. Basically, all the theory described thus far for a (forward) labeling algorithm can be applied to a backward algorithm thanks to the results in Irnich [82] showing that REFs can be inverted. To avoid producing twice as many labels as in a monodirectional labeling algorithm, the extension of a path in one direction is stopped when it is guaranteed that its remaining part can be generated in the other direction. This can be done in two ways: by arc bounding or by resource bounding. Arc bounding consists of computing an upper bound (in polynomial time) on the number of vertices that can still be visited by the path without exceeding resource constraints. If this number is less than the number of vertices already visited by the path, the extension is stopped. Resource bounding considers a *critical resource*, whose consumption is monotone along a path, to stop the path extension. A path is only extended if the accumulated amount of the critical resource does not exceed the half-way of its availability. For example, if the load is chosen to be the critical resource, only labels whose load consumption is less than $Q/2$ can be extended. Since the number of generated labels can grow exponentially with the number of arcs in the paths, keeping the length of the paths short in both directions reduce the total number of labels and speed up the labeling process, especially when the feasible

paths can contain a large number of vertices.

Note that, with bidirectional labeling, the same path can sometimes be obtained multiple times by concatenating more than one pair of forward and backward labels. To reduce as much as possible the concatenation process and avoid generating the same path multiple times, several rules may be considered. One example is given by Baldacci et al. [99] who only allow the concatenation of a pair of forward and backward labels if their critical resource consumptions differ by the smallest amount possible along the resulting path.

A final aspect that is worth discussing is the potentially unbalanced numbers of labels generated by the forward and backward algorithms, i.e., one direction generates much more labels than the other, even if the half-way point is carefully chosen. This can be caused by asymmetric data coming from the VRP instances or by the dual information obtained over the CG iterations. Trying to balance the workload between the forward and backward algorithms, Tilk et al. [100] propose a dynamic half-way point strategy that is applied at each iteration. Let H_F and H_B be resource limits associated with the forward and the backward algorithm, respectively. In a standard algorithm, $H_F = H_B = Q/2$ if load is the critical resource, and all forward extensions are performed before the backward extensions. In a dynamic half-way point algorithm, the values of H_F and H_B are updated dynamically during the algorithm execution, always ensuring that $H_F \geq H_B$ to guarantee optimality. Moreover, forward and backward labels are extended in an alternate way. To determine which label to extend next, the algorithm chooses first the direction with the least number of generated labels, number of processed labels, or number of unprocessed labels. It then chooses the label in the selected direction according to a standard rule (e.g., least load) and extends it before updating H_F or H_B based on the critical resource values in the newly created labels. A similar strategy is conceived by Pecin et al. [45].

Completion bounds.

Depending on the dominance rule used, some unpromising labels may be unnecessarily kept in a labeling algorithm. It is, however, possible to reinforce a dominance rule with the use of *completion bounds*. Let L be a label associated with path $p(L)$. A completion bound for $p(L)$ can be obtained by computing a lower bound $lb(L)$ on the reduced cost of all feasible extensions in $\mathscr{E}(L)$ that reach vertex $n+1$. If $\bar{c}(L)+lb(L) \geq 0$, then label L can be discarded without losing any negative reduced cost route. Completion bounds are used in several papers, including [2, 45, 46, 93, 97, 99, 101].

Computing the best completion bound for every generated label would make the labeling algorithm too time consuming. Consequently, good approximations, such as the following

ones, are rather used. Martinelli et al. [97] compute completion bounds inside a DSSR procedure. They use the labels computed at the previous DSSR iteration to estimate completion bounds. When they solve symmetric CVRP instances, completion bounds are extracted directly from a best reduced cost matrix (see [2]). On the other hand, when solving asymmetric problems, they change the direction (forward or backward) of the labeling algorithm from one iteration to the other in the DSSR algorithm. When non-robust cuts are employed to strengthen the MP, completion bounds can be computed like in Contardo et al. [102] and Contardo and Martinelli [93]. They only consider at most 20% of the largest dual values associated with these cuts and to under- or overestimate the impact of the remaining ones, ensuring the bound validity. Finally, Pecin et al. [45, 46] extend the approach of Contardo and Martinelli [93] to apply completion bounds in bidirectional algorithms.

Heuristic pricing.

Despite the fact that many speed up techniques have been proposed to accelerate exact dynamic programming algorithms, they can still remain very time consuming. Because there is no need to solve the pricing problem exactly, except to prove the optimality of the current solution in the last CG iteration, fast and effective heuristics have been developed to find negative reduced cost variables. Their purpose is to reduce the number of calls to the exact labeling algorithm, often yielding a substantial reduction of the total computational time.

In most cases, pricing heuristics are adaptations of exact labeling algorithms. They can be obtained by either relaxing certain dominance rules or by heuristically reducing the size of the network. Some authors only keep a few labels for each pair of time and load values at each node [43, 45, 46, 97]. This implies that several non-dominated labels may be discarded, including those leading to optimal solutions. To generate q-routes, Fukasawa et al. [43] scale down customer demands and vehicle capacity by some factor $\rho > 1$, i.e., $q'_i = \lceil q_i/\rho \rceil$, $\forall i \in V'$, and $Q' = \lceil Q/\rho \rceil$.

Another way of eliminating labels is by employing aggressive dominance rules. For instance, Desaulniers et al. [44] only consider a (possibly empty) restricted subset of customer resources when applying dominance test (4.19). When performing graph reduction, several criteria may be used to remove arcs from the network. If one decides to eliminate unpromising arcs based on their current reduced cost, it can be done by only keeping the best incoming and outcoming arcs at each customer with respect to their reduced costs [43–45] or by reducing graph density using the same ranking approach [93]. Alternatively, Chabrier [88] removes from the network the arcs presenting a high resource consumption. Although heuristic labeling algorithms are largely employed, some BPC algorithms benefit from well-known heuristics.

As an example, Desaulniers et al. [44] and Archetti et al. [103] use tabu search to generate routes with a negative reduced cost. Finally, note that, as suggested by Desaulniers et al. [44], different pricing heuristics can be invoked at each CG iteration and throughout the whole CG process depending on the current phase of the process (beginning, middle or end). To ensure optimality, an exact algorithm must always be executed at least once, in the last CG iteration.

Miscellaneous approaches.

We start this subsection by discussing a trick that can be used for the VRPTW. For some instances, vehicle capacity may not be really restrictive. For this reason, it can be neglected in the pricing problem while rounded capacity inequalities (see Subsection 4.3.2) may be added to the MP whenever necessary [46]. This practice facilitates the solution of the pricing problem and can make the BPC algorithm more efficient.

Instead of using a labeling algorithm for solving the pricing problem of the VRPTW, Rousseau et al. [104] devise a constraint programming algorithm which allows to generate elementary routes. This algorithm includes arc elimination constraints, a dynamic programming search strategy, and a lower bounding procedure based on an assignment problem.

Feillet et al. [105] incorporate concepts from constraint programming such as *limited discrepancy search* (LDS), *label loading* and *meta extensions* to dynamic programming algorithms as an attempt to quickly generate paths with negative reduced cost. LDS is a tree search method that employs a heuristic criterion to define, for each vertex, which outgoing arcs are the most promising. Each other arc yields a discrepancy and an upper bound on the number of discrepancies that a path can contain is imposed to limit the search. This upper bound is initialized to a small value and increased as needed. Label loading and meta extensions, in turn, use information from the current RMP solution to accelerate the route generation process. A metavertex is created for each vertex of each route in this solution and loaded with a label representing the end of the corresponding route. During the labeling process, these new labels can be added to partial paths to generate complete routes or can be used to compute completion bounds.

Contrarily to other authors who try to accelerate labeling algorithms, Lozano et al. [95] develop a pulse algorithm to solve the ESPPRC by extending the work of Lozano and Medaglia [106]. In a pulse algorithm, paths connecting two vertices in a graph are found by propagating pulses through the network. It can be seen as a graph exploration procedure that follows a depth-first search strategy, where pruning is used to discard unpromising paths. Three pruning strategies are developed: infeasibility pruning, that checks for a vi-

olation of the structural constraints (vehicle capacity, time windows, elementarity); bound pruning, that uses primal solutions and lower bounds to discard sub-optimal solutions; and rollback pruning, that compares two pulses differing only by the last vertex visited. The pulse algorithm differs from a labeling algorithm in many aspects, namely: a dominance rule is not required because it does not need to handle long lists of labels; many pruning strategies in addition to pruning by infeasibility can be devised; and bidirectional search cannot be applied due its recursive nature. Finally, because recursive pulses explore one vertex at a time until it reaches the destination vertex, the algorithm can perform searches over different vertices in parallel.

Very recently, Desaulniers et al. [107] introduce a new paradigm for the pricing problem, called *selective pricing*. Selective pricing stems from the observation that, when an ESPPRC relaxation is used as a pricing problem, the CG algorithm can be stopped when there is a proof that no elementary routes with a negative reduced cost exist, even if there are still some non-elementary routes with a negative reduced cost. Consequently, the labeling algorithm can be selective and discards cautiously non-elementary routes even if they are not dominated. As an example, the authors implement selective pricing by modifying a labeling algorithm used to solve the *ng*-SPPRC. Note that this strategy has the potential to yield better lower bounds.

Finally, Boschetti et al. [108] observe that, because labeling algorithms work in a stage-wise way, ESPPRC relaxations can be solved more efficiently in a GPU environment. Therefore, they investigate the impact of using parallelization procedures to achieve time reduction for route relaxations such as q-routes and *ng*-routes. In particular, they develop parallel label extension procedures and a strategy to efficiently manage the sets $\Pi(L)$ of vertices that cannot be reached by extending the corresponding label L without violating the *ng*-restrictions.

4.3.2 Cutting

This section describes cutting strategies used in BPC algorithms for VRPs. Here, we limit our discussion to the CVRP and the VRPTW. Valid inequalities for other VRPs are discussed in Section 4.4. For the sake of clarity, we present robust and non-robust cuts separately.

Robust cuts.

Robust cuts, as defined by Fukasawa et al. [43], have the advantage of not changing the structure of the pricing problem. For VRPs, these cuts are often defined directly in terms of the arc-flow variables. They are effective at closing the integrality gap for some easy instances,

but are not sufficient for harder ones. Note that many cuts used in BC frameworks are already implied by the structure of the routes considered in the MP. For example, Letchford and Salazar-González [109] prove for the CVRP that all generalized large multistar cuts are implied by model (4.1)–(4.3), even if the set of routes contains q-routes. This aspect is illustrated by comparing the families of cuts used in the successful algorithms of Fukasawa et al. [43] and Pecin et al. [45] for the CVRP. After solving the root node relaxation, Fukasawa et al.'s algorithm chooses between a BC and a BPC algorithm to solve the problem. The BC algorithm considers many valid inequalities used by Lysgaard et al. [77], namely: Rounded capacity, framed capacity, strengthened comb, multistar, and extended hypotour cuts. In the BPC algorithm, only rounded capacity cuts contribute effectively to improving the lower bound obtained at the root node. On its side, the BPC algorithm of Pecin et al. [45] only applies rounded capacity and strengthened comb cuts.

Robust cuts can also be defined via capacity-indexed (CI) variables (see their definition below). Because these variables can be expressed by means of q-routes, the dual variables associated with cuts defined in terms of CI variables can be directly incorporated into the reduced cost of q-routes when solving a pricing problem allowing these routes.

***k*-path cuts, subtour elimination constraints, and rounded capacity cuts:** Cutting planes belonging to the family of k-path cuts (kPCs) are applied in many algorithms to solve VRPs. The kPCs assume that, given a subset $S \subseteq V'$ of customers, one can estimate a lower bound $k(S)$ on the number of vehicles required to serve all customers in S. In this case, at least $k(S)$ paths must enter set S in any feasible solution. The following inequality is valid for the CVRP and the VRPTW:

$$X(S) = \sum_{(i,j)\in\delta^-(S)} x_{ij} \geq k(S), \tag{4.25}$$

where $\delta^-(S) = \{(i,j) \in A \,|\, i \in V \setminus S, j \in S\} \subset A$ is the subset of arcs entering S and $X(S)$ is the total flow entering S. Note that, because a route may enter S more than once, the left-hand side of (4.25) expressed in terms of the routing variables provides an upper bound on the number of vehicles used to serve the demand of set S. A strengthened version of these cuts is introduced by Baldacci et al. [99] and discussed in Subsection 4.3.2.

How $k(S)$ is computed varies depending on the problem at hand. For the CVRP, $k(S)$ corresponds to the solution of a bin packing problem (BPP), where the bins have capacity Q and the weights of the items are given by the demands of the customers in S. Because the BPP is $\mathcal{NP}$-hard, the right-hand side of (4.25) can be replaced by $k(S) = \lceil d(S)/Q \rceil$ with

$d(S) = \sum_{i \in S} q_i$, yielding the subtour elimination constraints [76, 110] if $k(S) = 1$ and the rounded capacity cuts (RCCs) [76, 111] if $k(S) \geq 2$.

For the VRPTW, computing $k(S)$ is not an easy task because it requires solving a VRPTW restricted to subset S. Kohl et al. [112], however, show that determining only if $k(S) > 1$ can be manageable as this corresponds to solving a traveling salesman problem (TSP) with time windows (TSPTW), which can be done in pseudo-polynomial time using dynamic programming. If the TSPTW is infeasible for a subset S such that $X(S) < 2$, one obtains a so-called 2-path cut (2PC). In [112], the 2PCs are separated by enumeration. Desaulniers et al. [44] propose a generalization of the kPCs that may assign different coefficients to the arc-flow variables in (4.25) depending on the customers that can be visited by a path entering subset S through the corresponding arc.

Cuts defined with capacity-indexed variables: Another way of deriving robust cuts for VRPs is by using CI variables. These variables are introduced by Pessoa et al. [113] and successfully employed by Pessoa et al. [114] to derive robust cuts for their BPC algorithms. CI variables can be defined over a multigraph $G_Q = (V, A_Q)$, where A_Q is a set containing arcs $(i,j)^\kappa$, $\forall (i,j) \in A$, $\kappa = 0, \ldots, Q - d_i$ and $(0,i)^Q$, $\forall i \in V'$. With each arc $(i,j)^\kappa$ is associated a binary variable y_{ij}^κ that indicates if a vehicle traverses (i,j) with a residual capacity κ or equivalently with a load $Q - \kappa$. The CI variables can be easily written in terms of route variables when they are generated by a pricing problem involving a load resource (e.g., q-route variables). Let $b_{ij}^{r\kappa}$ be a binary coefficient that indicates if arc (i,j) is traversed by route r carrying a load $Q - \kappa$. Then, one can write:

$$y_{ij}^\kappa = \sum_{r \in \Omega} b_{ij}^{r\kappa} \lambda_r, \qquad \forall (i,j)^\kappa \in A_Q. \tag{4.26}$$

In a formulation defined in terms of the CI variables, a generic constraint indexed by s has the form $\sum_{(i,j)^\kappa \in A_Q} \beta_{ij}^{\kappa s} y_{ij}^\kappa \geq \beta_s$. By applying (4.26), we obtain the following constraint for the set partitioning model (4.1)–(4.3):

$$\sum_{r \in \Omega} \left(\sum_{(i,j)^\kappa \in A_Q} \beta_{ij}^{\kappa s} b_{ij}^{r\kappa} \right) \lambda_r \geq \beta_s. \tag{4.27}$$

The dual variable associated with this constraint can be easily handled in a labeling algorithm if condition (4.17) is replaced by $q(L_1) = q(L_2)$. Note that this new condition does not change the theoretical complexity of the labeling algorithm (see, e.g., [45] for details).

Let us present a family of cuts defined over CI variables introduced by Pessoa et al. [113].

Given a set $S \subseteq V'$ of customers, denote by $\delta_Q^-(S)$ the subset of arcs in A_Q, with any capacity index, entering S. The associated RCC can be written:

$$\sum_{(i,j)^\kappa \in \delta_Q^-(S)} y_{ij}^\kappa \geq \lceil d(S)/Q \rceil = k(S). \tag{4.28}$$

Let $\kappa^* = d(S) - Q(k(S) - 1) - 1$ be the largest load that a vehicle can deliver to the customers in S such that the remaining demand $d(S) - q^*$ still requires at least $k(S)$ vehicles to be delivered. If less than $k(S)$ vehicles with a residual capacity $\kappa > \kappa^*$ enters set S, then at least one other vehicle must enter S. This statement yields the following strengthened RCC defined in terms of the CI variables:

$$\tfrac{k(S)+1}{k(S)} \sum_{(i,j)^\kappa \in \delta_Q^-(S) \,:\, \kappa > \kappa^*} y_{ij}^\kappa + \sum_{(i,j)^\kappa \in \delta_Q^-(S) \,:\, \kappa \leq \kappa^*} y_{ij}^\kappa \geq k(S) + 1. \tag{4.29}$$

This inequality dominates (4.28) and can be further strengthened by including arcs exiting S (see [114]).

Other families of cuts defined in terms of CI variables are the homogeneous extended capacity cuts and the triangle clique cuts. See [113, 114] for details.

Non-robust cuts.

As mentioned in Section 4.2.3, handling the dual value of a non-robust cut in the pricing problem increases its complexity. In this section, we discuss some families of non-robust cuts developed in the context of VRPs and how their dual values can be handled in the labeling algorithms.

Subset row cuts: The subset-row cuts (SRCs) proposed by Jepsen et al. [115] are Chvátal-Gomory cuts of rank 1 derived from a subset of constraints (4.2). Given an index subset $C \subseteq V'$ of these rows, a SRC is defined as:

$$\sum_{r \in \Omega} \left\lfloor \gamma \sum_{i \in C} a_i^r \right\rfloor \lambda_r \leq \lfloor \gamma |C| \rfloor, \tag{4.30}$$

where $\gamma = 1/k$, with $k \in \{1, \ldots, |C|\}$. A given route variable λ_r has a non-zero coefficient in (4.30) if $\sum_{i \in C} a_i^r \geq k$, i.e., if route r visits more than k customers in C. Some recent research works [45, 46, 116, 117] investigate the impact of using other values of $k \in]0, |C|[$. Moreover, Petersen et al. [118] and Pecin et al. [117] consider more general rank-1 Chvátal-Gomory cuts

by allowing a specific multiplier γ_i for each customer $i \in C$. In the following, we focus on the case with identical multipliers and integer k values unless otherwise specified.

By varying the cardinality of C and the value of γ, different families of SRCs can be derived. However, most BPC algorithms consider SRCs obtained with $|C| \leq 5$. Larger sets C would be harder to separate and thus be only marginally useful. According to Pecin et al. [45], the interesting SRCs are:

- 3-SRCs: $|C| = 3$ and $\gamma = 1/2$. These are the most popular SRCs which are employed in [115], [44], [2], [93] and [45];
- 4-SRCs: $|C| = 4$ and $\gamma = 2/3$. Used by Pecin et al. [45];
- 5,2-SRCs: $|C| = 5$ and $\gamma = 1/2$. Used by Pecin et al. [45];
- 5,1-SRCs: $|C| = 5$ and $\gamma = 1/3$. Used by Pecin et al. [45];
- 1-SRCs: $|C| = 1$ and $\gamma = 1/2$. These inequalities, also called strong degree cuts, are applied by Contardo et al. [93] and Pecin et al. [45]. We discuss them later in this section.

From (4.11) and (4.30), the reduced cost of a route r in which one SRC s has been added to the MP is $\bar{c}_r = \sum_{(i,j)\in A} \bar{c}_{ij} b^r_{ij} - \sigma_s \nu^r_s$, where $\nu^r_s = \lfloor \gamma \sum_{i \in C_s} a^r_i \rfloor$, C_s is the set of vertices (customers) defining SRC s, and $\sigma_s \leq 0$ is the associated dual variable. The last term of this reduced cost can be seen as paying a *penalty* σ_s for every k visits to the customers in set C_s. To handle such a penalty, the following modifications to the basic labeling algorithms described in Section 4.3.1 are proposed by Jepsen et al. [115].

Let Θ be the set of all SRCs s in the MP such that $\sigma_s < 0$. For every $s \in \Theta$, a new resource $\nu_s(L)$ is added to the definition of a label L to count the number of times (mod k) that a customer in C_s has been visited in the associated path. Consequently, when extending a label L along an arc $(i, j) \in A$ to create a new label L', $\nu_s(L')$ is set equal to $\nu_s(L)$ if $j \notin C_s$ and to $\nu_s(L) + 1$ (mod k) otherwise. In the latter case, if $\nu_s(L') = 0$, then σ_s is subtracted from $\bar{c}(L')$.

The dominance rule also needs to be modified. As mentioned in [36], one way to do so is to add conditions

$$\nu_s(L_1) \leq \nu_s(L_2), \qquad \forall s \in \Theta, \tag{4.31}$$

when checking if a label L_1 dominates a label L_2. Indeed, if $\nu_s(L_1) > \nu_s(L_2)$ for a given SRC s, then it might be possible that, for a feasible extension of L_1 and L_2, the dual value σ_s is

subtracted from the reduced cost when extending L_1 but not when extending L_2. Jepsen et al. [115] develop a stronger dominance rule which replaces conditions (4.16) and (4.31) by:

$$\bar{c}(L_1) - \sum_{s \in \Theta_{1,2}} \sigma_s \leq \bar{c}(L_2), \tag{4.32}$$

where $\Theta_{1,2} \subseteq \Theta$ is the subset of SRCs for which $\nu_s(L_1) > \nu_s(L_2)$.

Because Jepsen et al. [115] showed that separating the SRCs is $\mathcal{NP}$-hard, SRCs are separated by full or restricted enumeration depending on the maximum cardinality of the sets C.

Finally, note that Baldacci et al. [2] implement a weak version of the 3-SRCs (weak-SRCs). For a given set C, a weak-SRC only contains variables for which the associated route traverses at least one arc $(i, j) \in A$ such that $i, j \in C$. The weak-SRCs have the advantage of being robust.

Limited-memory SRCs: One way to reduce the impact of the SRCs on the labeling algorithms would be to re-define the cuts such that, for a larger number of labels L and SRCs s, $\nu_s(L) = 0$. Pecin et al. [45] exploit this observation to introduce a weaker version of the SRCs called the limited-memory SRCs (lm-SRCs). A lm-SRC is defined by a set C, a multiplier γ and a memory set M ($C \subseteq M \subseteq V'$). It writes as:

$$\sum_{r \in \Omega} \alpha(C, M, \gamma, r) \lambda_r \leq \lfloor \gamma |C| \rfloor, \tag{4.33}$$

where each coefficient α is computed as a function of (C, M, γ, r) and satisfies $\alpha(C, M, \gamma, r) \leq \lfloor \gamma \sum_{i \in C} a_i^r \rfloor$. On the one hand, if $M = V'$, the lm-SRC is identical to the corresponding SRC. On the other hand, if $M \subset V'$, the lm-SRC s defined by (C_s, M_s, γ) may be weaker but its dual value σ_s can be handled more easily in the pricing problem. In this case, the value of $\alpha(C_S, M_s, \gamma, r)$ is determined by applying along route r a modified version of the labeling algorithm that handles the SRCs' dual values described above. Every time that a vertex $j \in C_s$ is visited and $\nu_s(L') = 0$, the value of α increases by one and σ_s is subtracted from the reduced cost. However, every time that a vertex $j \notin M_s$ is visited, the resource value $\nu_s(L')$ is reset to 0, i.e., the previous visits to customers in C_s are forgotten. In this case, the value of α has less chances to increase than in the full-memory case and, thus, $\alpha(C_s, M_s, \gamma, r) \leq \lfloor \gamma \sum_{i \in C_s} a_i^r \rfloor$.

Observe that the smaller the sets M are, the faster the labeling algorithm and the weaker the cuts will be. Given a regular SRC associated with a set C and a multiplier γ that is violated by the current solution of the MP, Pecin et al. [45] convert this SRC into a lm-SRC by finding

a small-sized set M that yields the same violation. This set M contains all vertices in C and some of the subsets of vertices that are visited between two customers in C in a route r for which $\lfloor \gamma \sum_{i \in C} a_i^r \rfloor > 0$ and $\lambda_r > 0$ in the MP solution. See [45] for details.

Despite the success achieved by lm-SRCs in the solution of the CVRP, they still present scalability issues for solving large-scale instances with loose structural constraints. In such instances, long feasible routes are generated, leading to large memory sets and intractable pricing problems. To address this issue, Pecin et al. [46] introduce new lm-SRCs where their memory is defined in terms of arcs instead of vertices. This gives rise to two families of lm-SRCs, namely, limited-vertex-memory SRCs (lvm-SRCs) and limited-arc-memory SRCs (lam-SRCs). A lam-SRC is expressed as in (4.33) except that the vertex memory M is replaced by an arc memory AM. A coefficient $\alpha(C, AM, \gamma, r)$ in a lam-SRC is computed similarly to a coefficient $\alpha(C, M, \gamma, r)$ in a lvm-SRC: previous visits to customers in C are forgotten when an arc $(i, j) \notin AM$ is traversed. Consequently, when comparing a lvm-SRC defined for a triplet (C, M, γ) and a lam-SRC defined for a triplet (C, AM, γ) with $AM \subset \{(i, j) \in A \,|\, i, j \in M\}$, the latter should have less impact on the labeling algorithm but be weaker. Note that a lvm-SRC defined for a customer set C, a vertex memory M and a multiplier γ can be seen as a special case of the lam-SRC with the same C and γ, and the arc memory $AM = \{(i, j) \in A \,|\, i, j \in M\}$.

Like for the lvm-SRCs, Pecin et al. [46] define the arc memory AM of a lam-SRC such that it is of small size size and preserves the level of violation of the corresponding SRC. Set AM is composed of some of the subsets of arcs traversed between two visits to customers in C in a route r for which $\lfloor \gamma \sum_{i \in C} a_i^r \rfloor > 0$ and $\lambda_r > 0$ in the MP solution (see [46]). During the separation of the lam-SRC cuts, it may happen that a violated cut is found for the same set C and multiplier γ of a previously generated cut but with different arc memories AM and AM'. These two memories can be merged for defining a unique cut with arc memory $AM \cup AM'$. A similar strategy can be applied for the lvm-SRCs.

Elementary cuts: Balas [119] introduces the elementary cuts as logical implications of the set partitioning constraints (4.2). These cuts ensure that, if a route $r \in \Omega$ is chosen in a solution and does not visit a customer $i \in V'$, then no route visiting i as well as at least one customer visited by r can be choosen. Pecin et al. [46] propose a new family of valid inequalities that dominate those of Balas [119]. When the set of routes contains only elementary ones, this dominance is strict. Given a customer subset $C \subset V'$, a customer $i \in V' \setminus C$, and multipliers $\gamma_i^C = (|C| - 1)/|C|$ and $\gamma_j^C = 1/|C|$, $\forall j \in C$, these new rank-1

Chvátal-Gomory inequalities, also called elementary cuts in Pecin et al. [46], are given by:

$$\sum_{r\in\Omega}\left\lfloor \gamma_i^C a_i^r + \sum_{j\in C}\gamma_j^C a_j^r \right\rfloor \lambda_r \leq 1. \tag{4.34}$$

Another advantage of these rank-1 Chvátal-Gomory cuts over those proposed by Balas [119] is that all the theory developed for the other families of rank-1 Chvátal-Gomory inequalities can be re-applied, namely, the label definition, the dominance rule, and the limited memory mechanisms. Furthermore, the elementary cuts with $|C| \in \{2,3,4\}$ correspond to general rank-1 Chvátal-Gomory cuts [118] with $|C| \in \{3,4,5\}$ and respective vector γ of multipliers $(1/2,1/2,1/2)$, $(2/3,1/3,1/3,1/3)$ and $(3/4,1/4,1/4,1/4,1/4)$ (and their permutations). The latter are among the optimal multipliers identified by Pecin et al. [117]. Contrarily to the other rank-1 Chvátal-Gomory cuts presented so far, a heuristic procedure based on local search is developed by Pecin et al. [46] for separating the elementary cuts.

Strengthened capacity cuts: Baldacci et al. [99] introduce the strengthened capacity cuts (SCCs) as a strengthened version of the RCCs. If one applies relation (4.7) to inequality (4.25), a robust inequality of the form $\sum_{r\in\Omega}\rho_S^r\lambda_r \geq k(S)$ is obtained, where $\rho_S^r = \sum_{(i,j)\in\delta^-(S)} b_{ij}^r$ is the number of times that route $r \in \Omega$ enters set S. Note that, even if Ω contains only elementary routes, ρ_S^r may be greater than 1 because a route may enter and leave S more than once. Baldacci et al. [120] redefine ρ_S^r as a binary parameter indicating whether route r visits at least one customer in S. With this new definition, a SCC can be expressed as:

$$\sum_{r\in\Omega}\rho_S^r\lambda_r \geq k(S). \tag{4.35}$$

When compared to RCCs (4.25), SCCs are stronger because they are not negatively affected by routes entering set S more than once. They are, however, non-robust cuts which require one additional binary resource in the label definition for each cut to indicate whether a route has already entered into the corresponding set S. When a route visits a customer in S for the first time, the dual value of the associated cut is subtracted from the reduced cost. The dominance rule must take these additional resources into account in a similar fashion as the resources for the SRCs.

To generate violated SCCs, Baldacci et al. [99] call the heuristic CVRPSEP separation routines [121] to identify violated RCCs that are converted into SCCs. Poggi and Uchoa [94] observe that, if there exists a set S' such that $S' \subset S$ and $k(S') = k(S)$, the SCC defined over S' dominates the one defined for S and, thus, only the cut associated with the former

set needs to be added to the MP. This aspect is important in an attempt to limit the number of non-robust cuts added to the MP.

Clique cuts: The clique inequalities are well-known valid inequalities for the set partitioning problem [122] which induce facets. They are defined over an undirected conflict graph $G' = (\Omega, E')$, in which an edge between two vertices $r_1, r_2 \in \Omega$ exists if r_1 and r_2 visit both a customer $i \in V'$, i.e., $a_i^{r_1} \geq 1$ and $a_i^{r_2} \geq 1$. A clique $\hat{\Omega}$ in G' is a set of vertices that are conflicting pairwise and, thus, the corresponding routes are not allowed to appear simultaneously in a feasible solution to the problem. Therefore, a clique $\hat{\Omega}$ yields the following valid inequality:

$$\sum_{r \in \hat{\Omega}} \lambda_r \leq 1. \tag{4.36}$$

Spoorendonk and Desaulniers [123] explore the application of clique cuts in a BPC algorithm for the VRPTW. They devise a modified labeling algorithm that handles the dual values of the clique cuts. This new labeling algorithm relies on an approximate clique representation that employs a minimal set $\chi_{min}(\hat{\Omega})$ of conflicting rows. It requires adding for each clique cut defined for a set $\hat{\Omega} \subseteq \Omega$, $|\chi_{min}(\hat{\Omega})| + 1$ binary resources to the definition of a label to detect when a route contributes to this cut. The dominance rule also needs to be modified. A greedy heuristic is used to separate the cuts.

***k*-cycle elimination cuts and strong degree cuts:** If set Ω contains non-elementary routes, some valid inequalities may be used to impose elementarity. Indeed, Contardo and Martinelli [93] present the k-cycle elimination cuts (k-CECs) that aim at preventing routes containing cycles of length $k \geq 2$ to be part of a MP solution. Given a route $r \in \Omega$ and a vertex $j \in V'$, let α_j^{kr} be a parameter indicating the number of times that route r visits vertex j either for the first time or after at least k vertices since the last visit to j. A k-CEC associated with vertex j is expressed as:

$$\sum_{r \in \Omega} \alpha_j^{kr} \lambda_r \geq 1. \tag{4.37}$$

The k-CECs ensure that no route $r \in \Omega$ visiting a vertex $j \in V'$ more than once and such that $\alpha_j^{kr} < a_j^r$ will be in the solution of the MP. To deal with the dual values of the generated k-CECs in the labeling algorithm, a new resource is added to the label definition for each generated k-CEC. For a cut associated with vertex j, this resource takes value k until reaching

j for the first time. It is reset to 0 at every visit to j. Then, it counts the number of vertices different from j that are visited consecutively. The dual value associated with this cut is subtracted from the reduced cost every time that vertex j is visited and the value of this resource is greater than or equal to k. As explained in Contardo and Martinelli [93], the dominance rule must be modified. These cuts are separated by inspection.

If one considers $k = \infty$, we obtain the strongest k-CECs, called strong degree cuts (SDCs), which are introduced by Contardo et al. [102] for the capacitated location-routing problem (CLRP). Let α_j^r be a binary parameter equal to 1 if route $r \in \Omega$ visits vertex $j \in V'$ at least once and 0 otherwise. The SDC for a vertex j is:

$$\sum_{r \in \Omega} \alpha_j^r \lambda_r \geq 1. \tag{4.38}$$

For this case, the additional resource for a SDC associated with a customer j simply indicates whether or not vertex j has already been visited. The associated dual value is subtracted from the reduced cost only at the first visit to customer j.

Contardo et al. [102] show that, even when non-elementary routes can be generated by the pricing problem, adding SDCs yields a lower bound equal to the lower bound obtained when considering only elementary routes. Contardo et al. [89] further show empirically that combining SDCs and *ng*-routes is a very effective strategy to reach this bound.

4.3.3 Branching

Here, we present the most commonly used branching strategies in BPC algorithms for VRPs.

Branching on arcs/edges.

Desrosiers et al. [20] introduce branching on arc-flow variables x_{ij}, $(i, j) \in A$, for the VRPTW. It is probably the most popular strategy due to its simplicity, ease of implementation, and robustness. In BPC algorithms, setting $x_{ij} = 0$ can be imposed by removing all variables λ_r, $r \in \Omega$, with $b_{ij}^r > 0$ that are present in the current RMP, and arc (i, j) from arc set A to avoid generating such route variables. Any other decision made on an arc-flow variable x_{ij} can be re-written using relation (4.7) in terms of the route variables λ_r, $r \in \Omega$, and added to the MP. Then, the associated dual variable can be incorporated in the modified cost of the arc (i, j) when solving the pricing problem.

When arc (i, j) is associated with a set partitioning constraint (4.2), i.e., $i \in V'$ or $j \in V'$, the addition of a new constraint to the MP can be avoided. Indeed, in this case, x_{ij} is binary and

the associated branching decisions are $x_{ij} = 0$ and $x_{ij} = 1$. The former decision is treated as mentioned above. The latter decision means that arc (i, j) must be in the solution. To enforce this, we remove from the current RMP, all route variables associated with a route traversing an arc (i, k), with $k \neq j$, if $i \in V'$, and an arc (k, j), with $k \neq i$, if $j \in V'$. All these arcs must be removed from arc set A to ensure that vertex j will always be visited immediately after vertex i in any route generated by the pricing problem.

When the problem is defined over an undirected graph like the CVRP, branching on edge-flow variables can be applied. Indeed, most of the theory described above can be employed. However, given that, for such a problem, the pricing problem is often better defined as a ESPPRC on a directed graph, the flow x'_{ij} on an edge $\{i, j\}$ is computed as $x'_{ij} = x_{ij} + x_{ji}$, where x_{ij} and x_{ji} are the flows on the arcs (i, j) and (j, i) in this directed graph. In this case, setting $x'_{ij} = 0$ can be imposed as above. However, imposing $x'_{ij} = 1$ requires adding a constraint in the MP.

Branching on the number of vehicles.

When the number of vehicles used in a solution to the MP is fractional, one can branch on this number which can be expressed as $\sum_{j \in V'} x_{0j}$, i.e., the total flow on the arcs leaving the origin depot. Note that this quantity is also equal to the sum of all the route variables when there is a single depot. Such a branching decision is implemented through the addition of a constraint in the MP. In the pricing problem, its dual value needs to be considered in the modified cost of all the arcs leaving the depot.

This branching strategy has often a significant impact on the size of the search tree. Given that there can exist fractional solutions in which the number of vehicles used is integer, it is not sufficient to explore to whole search tree and must be combined with another branching strategy. In fact, Desrochers et al. [84] branch in priority on the number of vehicles used and to branch on the arc-flow variables when this number is integer.

Branching on cutsets.

Even if branching on arcs/edges has been largely used in BPC algorithms for VRPs, it has the disadvantage of inducing only local changes to the current fractional solution. For this reason, when implementing a BC algorithm to solve the CVRP, Augerat et al. [124] branch on cutsets as an attempt to obtain larger perturbations. Let us define this branching strategy in terms of the edge-flow variables x_e, $e \in E$, where E is the edge set. Let $S \subseteq V'$ be a subset of customers, called a *cutset*. The total flow entering S is given by $\sum_{e \in \delta(S)} x_e$, where $\delta(S)$

denotes the set of edges containing exactly one extremity in S. Assuming that at least one route enters S, we can write $\sum_{e\in\delta(S)} x_e = 2\kappa(S) + \phi(S)$, where $\kappa(S)$ is a non-negative integer and $0 \leq \phi(S) < 2$. In an integer solution, $\phi(S) = 0$. Consequently, if $\phi(S) > 0$, then one can branch on cutset S by imposing $\sum_{e\in\delta(S)} x_e \leq 2\kappa(S)$ on one branch and $\sum_{e\in\delta(S)} x_e \geq 2\kappa(S)+2$ on the other. These decisions can be imposed as constraints in the MP. Note that most works focus on sets S with $\kappa(S) = 1$. Note also that, in their BPC algorithm, Pecin et al. [45] apply equivalent branching decisions by considering only the arcs entering into a cutset in a directed graph.

The challenge with this branching strategy is the exponential number of cutsets that can be used to define the branching decisions. In this regard, several simple heuristics are designed by Augerat et al. [124], Naddef and Rinaldi [111], Lysgaard et al. [77], Fukasawa et al. [43] and Pecin et al. [45] to select the cutsets to branch on. In general, these heuristics seek to yield balanced search trees and a higher impact on the generated lower bounds.

Branching on resource windows.

Assuming that travel and service times are integer, Gelinas et al. [125] branch on time windows but a similar branching scheme can be devised for other resources such as load. In a fractional solution to the MP, there might exist several routes visiting the same customer $i \in V'$ but with different start of service times. Under certain conditions, splitting the time window $[e_i, l_i]$ of customer i in two disjoint time windows $[e_i, t]$ and $[t+1, l_i]$, with $t \in [e_i, l_i-1]$, can make some of these routes infeasible for window $[e_i, t]$ and others infeasible for $[t + 1, l_i]$. Branching on time windows consists of finding a customer for which splitting its time window in two sub-windows would eliminate the current fractional solution in both branches, and of determining how it should be split. These two intertwined choices must be carefully made to devise an efficient branching strategy. Once customer i and time t are identified, the branching decisions are applied as follows. In the branch associated with $[e_i, t]$, we replace time window $[e_i, l_i]$ by $[e_i, t]$ in the pricing problem and eliminate from the current RMP all routes in which customer i is visited outside $[e_i, t]$. In the other branch, we proceed similarly but with the new time window $[t + 1, l_i]$.

Dell'Amico et al. [126] devise a similar branching technique on resources that are used to handle the vehicle capacity in a VRP with simultaneous pickups and deliveries (see Section 4.4.10). Finally, Christiansen and Lysgaard [127] branch on the expected cumulative demand in the context of the VRP with stochastic demands (Section 4.4.9).

Strong branching.

Some recent successful BPC algorithms rely on strong branching [43,45,46]. The idea behind strong branching is to quickly evaluate the impact of a set of branching candidates (arcs, time windows, subset of customers, etc.) on the lower bounds that would be obtained in each child node and to select the best one according to some criterion. For example, when applying arc branching at a given node, a subset of arcs $A^C \subset A$ that are candidates for branching is determined and, for each arc a in A^C, the corresponding branching decisions are successively applied to compute the lower bounds lb_a^- and lb_a^+ that would be achieved in both child nodes if this arc was selected for branching. Then, after computing all these lower bounds, one can choose the best candidate as an arc $a^* \in \arg\max_{a \in A^C} \min\{lb_a^-, lb_a^+\}$ and add to the search tree only the child nodes associated with the decisions for arc a^*.

Strong branching can reduce significantly the number of nodes to explore in the search tree but often at the expense of increasing the time to select the branching candidate at each node. Consequently, it is common practice to limit the size of the set of candidates to evaluate and to compute approximate lower bounds in the strong branching selection process. Similar to the strong branching scheme of Lysgaard et al. [77] involving branching decisions on cutsets, Fukasawa et al. [43] select, at each node of the search tree requiring branching, between 5 and 10 candidate cutsets S according to a criterion based on the flow entering S. Then, the lower bounds for each candidate is evaluated heuristically by performing a few CG iterations.

In Irnich [128], the size of the candidate set is defined dynamically during the exploration of the search tree and is proportional to the relative optimality gap. Therefore, it is more selective at the beginning of the algorithm when no good solutions have been found and at the top of the search tree when the optimality gap is still high, and evaluates less candidates when the gap gets smaller. The lower bounds for each candidate are, however, computed by an exact CG algorithm.

For the CVRP, Pecin et al. [45] assess the impact of branching on cutsets by proposing an aggressive hierarchical hybrid strong/pseudocost branching, that consists of three phases: i) candidate set construction, ii) refinement, and iii) strong branching. In the first phase, a set of candidate cutsets is found. Half of them are chosen based on their pseudocosts (average lower bound increases when the cutset was previously selected for evaluation) and the other half are new cutsets. As in Irnich [128], the size of the candidate set depends on the relative optimality gap. During the refinement phase, lower bound values lb^- and lb^+ are heuristically estimated for each candidate by only considering the variables in the current RMP. These quick evaluations allow to select a smaller set of candidates which are then better evaluated in the last phase using a heuristic CG algorithm.

4.3.4 Using upper bounds

In this section, we discuss two techniques that can be exploited in BPC algorithms for VRPs when good lower and upper bounds are available. The first technique consists of fixing to zero the value of certain variables in order to reduce the size of the model. The second is based on the enumeration of elementary routes to help close the integrality gap at a node of the search tree. This latter technique can be seen as an alternative to branching.

Variable fixing by reduced cost.

Variable fixing can be performed in different ways. Here, we concentrate on variable fixing by reduced cost which requires a lower and an upper bound on the optimal value of the problem. The general idea behind variable fixing is as follows (see, e.g., [31]). Let $P := \{\min\ c^\top x : Ex = b, x \in \mathbb{Z}^n_+\}$ be an integer linear program, and $\bar{z}$ an upper bound on its optimal value, derived from a feasible solution. Also, let $D := \{\max\ \pi^\top b : \pi E \leq c\}$ be the dual problem associated with the linear relaxation of P and π a feasible dual solution to D of cost $\underline{z} = \pi^\top b$. The reduced cost $\bar{c}_j$ of a generic variable x_j with respect to π is given by $\bar{c}_j = c_j - \pi^\top E_j$, where E_j denotes the coefficient column of x_j in matrix E. Variable x_j can be fixed to zero if $\bar{c}_j > \bar{z} - \underline{z}$ and can thus be removed from the problem.

In BPC algorithms, variable fixing is usually applied after solving the MP at each node of the search tree, when an optimal dual solution π becomes available. However, it is not directly applied to route variables λ_r, $r \in \Omega$, because, as discussed in Section 4.2.4, it would require employing complex mechanisms to forbid in the pricing problem the (re-)generation of the routes associated with the removed variables. For this reason, variable fixing is rather applied to implicit arc variables [93, 129, 130], yielding the simultaneous elimination of a large number of route variables, namely, all those associated with a route traversing an arc to be removed. Besides, removing arcs from the network makes the pricing problem easier to solve. However, there is an inconvenient to this practice: the reduced costs of the arc-flow variables x_{ij} are not directly available in a CG algorithm. Some approaches such as the addition of coupling constraints of the form (4.7) to the MP [15, 33] and the solution of the pricing problem directly as a linear problem [131] have been proposed in the literature to overcome this difficulty. In the following, we highlight the approaches of Irnich et al. [130] and Contardo and Martinelli [93], which are designed for VRPs.

Irnich et al. [130] fix arc-flow variables to zero as follows. Let $\Omega_{ij} \subset \Omega$ be the subset of feasible routes traversing arc $(i, j) \in A$. The reduced cost $\bar{c}_{ij}$ of arc-flow variable x_{ij} can then be computed as $\bar{c}_{ij} = \min_{r \in \Omega_{ij}} \bar{c}_r - \min_{r' \in \Omega} \bar{c}_{r'}$. Because $\min_{r' \in \Omega} \bar{c}_{r'} \leq 0$ at any CG iteration,

this result implies that, if $\tilde{c}_{ij} = \min_{r \in \Omega_{ij}} \bar{c}_r > \bar{z} - \underline{z}$, then arc (i, j) can be removed from G. Alternatively, if lb_{ij} is a lower bound on $\tilde{c}_{ij}$, then $lb_{ij} > \bar{z} - \underline{z}$ is a sufficient condition to remove arc (i, j). Such a lower bound can be computed for all arcs at once by solving the pricing problem using an exact monodirectional labeling algorithm (see [129, 130]). To compute $\tilde{c}_{ij}$ exactly and remove the maximum number of arcs, Irnich et al. [130] find feasible shortest paths from vertex 0 to all other vertices using an exact forward labeling algorithm as well as feasible shortest paths for all vertices to vertex $n+1$ using an exact backward labeling algorithm. Given that two full labeling passes need to be performed, this process might be time-consuming. Instead, relaxed shortest paths such as *ng*-paths can be computed, yielding only lower bounds on the arc reduced costs and possibly less removed arcs depending on the quality of these bounds. The state-of-the-art BPC algorithm of Pecin et al. [46] eliminates arc-flow variables in the same fashion.

As suggested by Pecin et al. [45], this technique can also be applied for fixing CI variables y_{ij}^{κ} instead of arc-flow variables x_{ij} when the BPC algorithm exploits an underlying CI compact formulation (see Section 4.3.2). In this case, the reduced costs $\tilde{c}_{ij}^{\kappa}$ of these CI variables (defined as $\tilde{c}_{ij}^{\kappa} = \min_{r \in \Omega_{ij}^{\kappa}} \bar{c}_r - \min_{r' \in \Omega} \bar{c}_{r'}$, where Ω_{ij}^{κ} is the set of all feasible routes traversing arc $(i, j)^{\kappa}$ in G_Q) are also computed by executing complete forward and backward labeling algorithms. The advantage of this new technique is that it allows to fix to zero more route variables. Indeed, when variable fixing is only based on the arc-flow variables x_{ij}, none of the route variables traversing an arc (i, j) can be fixed to zero if $\tilde{c}_{ij} \leq \bar{z} - \underline{z}$. However, some of them might be fixed when CI variables are involved, i.e., for all $\kappa \in \{0, 1, \ldots, Q - d_i\}$ such that $\tilde{c}_{ij}^{\kappa} > \tilde{c}_{ij} = \min_{\kappa \in \{0,1,\ldots,Q-d_i\}} \tilde{c}_{ij}^{\kappa}$ and $\tilde{c}_{ij}^{\kappa} > \bar{z} - \underline{z}$.

Although Contardo and Martinelli [93] developed a BPC algorithm for the multi-depot VRP under capacity and route length constraints, their algorithm is also very effective at solving CVRP instances. It proceeds in two phases and variable fixing is performed in each phase. First, the algorithm solves the linear relaxation of an edge-flow formulation to quickly generate a lower bound on the optimal value of the problem. This bound is then improved by generating cutting planes. When no more cuts are found, variable fixing is performed on the edge-flow variables, i.e., edges are removed from the network. In the second phase, a BPC algorithm is applied to solve the problem using the reduced network in the pricing problem. In this algorithm, variable fixing is performed as in Irnich et al. [130], except that the arc reduced costs are underestimated because the duals of the non-robust cuts are not fully considered in the labeling process.

Route enumeration.

Ideally, one would like to be able to enumerate all feasible routes and solve directly model (4.1)–(4.3) using a mixed integer programming solver. This is unpractical given that the number of feasible routes grows exponentially with the size of the instance. However, given an upper bound $\bar{z}$ on the optimal value obtained from a feasible solution s^* and a dual solution π providing a lower bound $\underline{z}$ like in variable fixing, enumerating the subset Ω'' of all the routes $r \in \Omega$ such that $\bar{c}_r < \bar{z} - \underline{z}$ and solving model (4.1)–(4.3) restricted to the routes in Ω'' is sufficient to either prove that s^* is optimal or find an optimal solution with a cost less than $\bar{z}$. This approach, called route enumeration, is presented by Baldacci et al. [99] and Baldacci et al. [2], who propose exact algorithms for solving the CVRP and the VRPTW. These algorithms are not BPC algorithms, as no branching is performed. In both algorithms, an additive bounding procedure which relies, among others, on CG and cutting planes is used to compute a high-quality lower bound. Route enumeration is then performed using a labeling algorithm similar to the ones applied for solving the pricing problem. Finally, if all routes can be enumerated, the MIP restricted to the subset of enumerated routes is solved by a commercial MIP solver, thus benefitting from all cutting-edge tools offered by this solver and tailored cutting planes such as SCCs and clique cuts (see, e.g., [45, 46, 93, 99]). These algorithms are very efficient when the gap $\bar{z} - \underline{z}$ is small. Otherwise, they might fail to enumerate all required routes and simply abort during the enumeration process due to a lack of memory.

To overcome this drawback, it is possible to employ a hybrid strategy which combines route enumeration and branching [45, 46, 93, 113]. In the BPC algorithm of Pessoa et al. [113], route enumeration is performed at each node of the BB search tree. Every time that the MP is solved, possibly after adding cuts, route enumeration is attempted. If the number of generated labels or the number of generated routes exceeds a predefined threshold at a node, enumeration is stopped and branching is performed. Otherwise, the model associated with this node and restricted to the set of enumerated routes is solved using a MIP solver, and no branching is necessary. Because the optimality gap tends to decrease with the depth in the tree, route enumeration can eventually work, reducing the size of the tree and the total computational time.

Contardo and Martinelli [93] perform route enumeration in a more aggressive way. Indeed, they allow to generate a large number of routes (e.g., up to 5 millions) that cannot be handled directly by a MIP solver. Once the MP is solved at a node of the search tree, the algorithm proceeds to route enumeration. If too many labels or too many routes are generated, the process is aborted. If the number of enumerated routes is small enough, the resulting model

is solved by a MIP solver. Otherwise, the routes are stored in a pool and the algorithm continues by adding cuts and fixing variables. At this point, when cuts are added, the MP is re-optimized by CG but the pricing problem is solved by inspecting the pool of routes. This allows to separate more non-robust cuts without impacting dramatically the complexity of the pricing algorithm. Pecin et al. [45, 46] implement route enumeration as in Contardo and Martinelli [93]. They, however, resort to branching when route enumeration is not possible.

4.3.5 Stabilizing dual variable values

CG algorithms are well known to suffer from convergence issues, which are mainly due to the degenerate nature of the MP and to the instability of the dual variable values from one iteration to the next [15, 50]. On the one hand, degenerate MPs often yield multiple optimal dual solutions and make it difficult to prove the optimality of the primal solution. Moreover, the algorithm may perform many iterations without any or almost any improvement of the objective function value (*tailing-off effect*). On the other hand, dual variable instability causes the algorithm to generate columns unlikely to be in an optimal solution, even if some good columns have already been generated.

Simple modifications to the MP may help the convergence of a CG algorithm. For instance, by avoiding redundant constraints in the MP or removing non-active constraints, the degree of degeneracy may be reduced. In addition, if one formulates the VRP as a set-covering problem instead of a set-partitioning problem, i.e., replacing equalities (4.5) in the MP by inequalities, the dual solution space is cut in half (the corresponding dual variables become non-negative) and the dual variables are typically more stable [21, 64].

Dual variable stabilization strategies have also been developed to overcome these convergence issues. In general, stabilization techniques seek to: i) limit the distance traveled by the variables in the dual space, either by restricting the dual space or penalizing long displacements of the dual points (see, e.g. [57, 60]); ii) avoid the generation of extreme dual solutions [64]; or iii) alleviate the impact of bad dual values [63]. The first family of stabilization strategies either force the dual variables to remain within relatively small intervals around the current dual values or allow to go outside these intervals at the expense of paying a penalty. The intervals and penalties are adjusted dynamically throughout the CG algorithm. Such methods can yield substantial speedups if good initial dual information is available. However, this is not the case, in general, for VRPs.

Extreme dual solutions may be harmful to CG algorithms because they can lead to unrealistic route reduced costs and the generation of routes that have little chance to be part of an optimal MP solution. Thus, it may be preferable to use interior dual points to reduce the

number of CG iterations. Naturally, these points can be obtained directly by solving at each iteration the RMP using an interior point method. Instead, Rousseau et al. [64] generate dual interior points by using convex combinations of several dual extreme points. At a given iteration, these extreme points are computed by solving the RMP several times after perturbing the right-hand side vector each time.

To reduce the impact of poor dual values and avoid large dual variable oscillations, Pessoa et al. [63] devise a dual price smoothing technique, which considers a transformed dual vector $\tilde{\pi} = \alpha\hat{\pi} + (1-\alpha)\pi$ when solving the pricing problem. In this expression, parameter $\alpha \in (0, 1]$ indicates the level of smoothing, and vectors π and $\hat{\pi}$ are, respectively, the current dual vector and a vector containing dual information from previous iterations. Vector $\hat{\pi}$ can be the best dual vector obtained so far [61] or a weighted sum of dual vectors computed in previous iterations [62]. This stabilization technique proceeds by solving at each CG iteration the pricing problem defined with the vector $\tilde{\pi}$ of dual values. Three outcomes may happen: the labeling algorithm finds 1) routes that have a negative reduced cost with respect to both $\tilde{\pi}$ and π; 2) routes that have a negative reduced cost with respect to π but not with respect to $\tilde{\pi}$; 3) no routes with a negative reduced cost with respect to π. In the first two cases, the computed routes can be added to the current RMP. In the second case, the value of α is decreased as the algorithm converges towards optimal dual values. Finally, in the third case, the value of α is also decreased and the pricing problem is solved again with the updated vector $\tilde{\pi}$.

As mentioned in Rousseau et al. [64] and Pessoa et al. [63], the use of a stabilization technique tends to increase the difficulty of solving the pricing problem, especially in the first CG iterations. Indeed, when no stabilization is used, the dual values are often badly distributed among the customers, yielding non-attractive customers that are rapidly disregarded during the search for negative reduced cost routes. In practice, this aspect is not a major issue because the quality of the generated routes outweighs this disadvantage.

4.4 Contributions to specific VRPs

In this section, we describe contributions that are specific to some VRP variants. Given the large number of contributions, we have chosen to summarize the general ideas of these specific contributions and to avoid mentioning every feature of each algorithm.

This section is divided according to the following problem features: heterogeneous fleet and multiple depots (Section 4.4.1), profits (Section 4.4.2), soft time windows (Section 4.4.3), multiple trips (Section 4.4.4), split services (Section 4.4.5), time dependency (Section 4.4.6),

cumulative costs (Section 4.4.7), environmental aspects (Section 4.4.8), uncertainty (Section 4.4.9), and pickups and deliveries (Section 4.4.10).

4.4.1 Heterogeneous fleet and multiple depots

One of the most direct variants of the CVRP is the heterogeneous fleet VRP (HFVRP) in which vehicles of different types are available. Let $\mathcal{K}$ be the set of vehicle types. These types may differ by their capacity, fixed cost and traveling costs. The HFVRP consists of determining the fleet of vehicles to use and designing feasible routes for them so as to service a set of customers at minimum total cost. This cost is given by the sum of the vehicle fixed costs and the variable traveling costs.

Given the differences between the vehicle types, a basic BP algorithm for the HFVRP requires one pricing problem for each type $k \in \mathcal{K}$, increasing the complexity of the pricing step. Choi and Tcha [132] develop the first CG-based algorithm for this problem which relies on the following procedure to reduce the number of pricing problems. Let z^* be the cost of a feasible solution obtained, e.g., by a heuristic. For each vehicle type $k \in \mathcal{K}$, compute a lower bound $\underline{z}_k$ on the optimal value of the problem when forcing the utilization of at least one vehicle of type k. If $\underline{z}_k > z^*$, then no vehicle of type k can be used in an optimal solution and the corresponding pricing problem can be discarded. To compute rapidly a lower bound $\underline{z}_k$, the authors solve a small-sized integer program that aims at minimizing the total fixed cost incurred by a fleet that has sufficient capacity to cover the total demand. This lower bound might be useful to discard some vehicle types only if the fixed costs are very large compared to the traveling costs.

Pessoa et al. [114] design a BPC algorithm for the HFVRP. When vehicle types may incur different traveling costs, multiple pricing problems are necessary. Otherwise, the authors show how to use a single pricing problem. They also introduce new families of robust cuts defined over CI variables, namely, homogeneous extended capacity cuts and strengthened RCCs, and also applied robust triangle clique cuts. Even if the HFVRP is usually defined over an undirected graph, the proposed algorithm relies on directed graphs in the pricing problems which are needed to handle the cuts defined over the CI variables. These networks introduce symmetry while solving the pricing problems because the same route can be traversed in both directions at the same reduced cost unless the contributions to the cuts differ with the direction. To break this symmetry, Pessoa et al. [114] impose that the index of the first customer visited along a route must be less than that of its last customer and adapt their labeling algorithm to handle this requirement.

Pessoa et al. [47] propose a new BPC algorithm for the HFVRP that combines the most re-

cent methodological advances for the CVRP and the VRPTW. By employing a new pseudo-polynomially large extended formulation, stronger extended capacity cuts are devised. Furthermore, they exploit the structure of the HFVRP to propose the concept of vehicle-type-dependent memory for the SRCs (see Section 4.3.2). This new idea yields a sharper definition of the concept of lm-SRCs by defining the coefficients of the route variables in the cuts according to their associated vehicle type. Each cut has, thus, a different memory for each vehicle type and this memory is smaller than the memory that would not depend on the vehicle type. Consequently, the impact on the time required to solve each pricing problem is more limited. On the other hand, these cuts are weaker and additional rounds of cut separation may be required to achieve the same final lower bound. Finally, Pessoa et al. [47] also exploit the structure of the HFVRP to develop a progressive route enumeration scheme. Indeed, given that enumerating routes for small-capacitated vehicles is easier, in general, than for large-capacitated vehicles, enumeration can be possible only for a subset of the vehicle types. Thus, when enumeration is successful for a type, the enumerated routes are added to the RMP or stored in a pool (see Section 4.3.4), the corresponding pricing problem is removed, and all the generated cuts can be lifted with respect to these route variables.

The multi-depot VRP (MDVRP) is also a direct extension of the CVRP, where the available vehicles are assigned to a set of depots. A route for a vehicle of a given depot must start and end at this depot. The MDVRP can be seen as a special case of the HFVRP by associating each depot with a vehicle type.

Baldacci and Mingozzi [133] design a general solution method for solving the HFVRP and several of its special cases including the MDVRP. This method extends the framework introduced by [99] for the CVRP (see Section 4.3.4) to deal with different vehicle types. It also incorporates mechanisms designed for the HFVRP, namely, a relaxation involving binary variables assigning customers to vehicle types and rules to resize the vehicle fleet (i.e., to update lower and upper bounds on the number of vehicles of each type). The first mechanism is especially effective when the fixed cost contribution to the total cost is significant. The overall algorithm consists of solving three different relaxations and keep the best lower bound obtained to perform route enumeration.

Bettinelli et al. [134] tackle a multi-depot heterogeneous VRPTW with limited route duration, where the duration of a route is computed as the difference between the return time to the depot and the departure time from it. The latter is a decision variable, as some waiting time along the route may be avoided by delaying the departure from the depot. To solve this problem, Bettinelli et al. [134] conceive a BPC algorithm with one pricing problem per vehicle type and depot. All pricing problems are, however, solved at once using a single

bidirectional labeling algorithm which handles two additional resources to impose a maximal route duration. This labeling algorithm relies on labels that contains, in theory, duplicated components for each depot. Given that two paths associated with different depots and covering the same customers in the same order only differ by their initial arcs, each label rather contains components for a single depot and, for each depot, a pointer to the initial arc of the path if it were to start at this depot. Since it is easy to retrieve the component values for each depot with these pointers, label storage requires much less memory. To deal with the multiple vehicle types that only differ by the fixed cost and the vehicle capacity, the labeling process is performed considering the largest vehicle capacity. Then, during the joining phase of the bidirectional search algorithm, the feasibility rules are verified for each vehicle type and the reduced cost is adjusted accordingly.

To solve the MDVRP with a route length constraint, Contardo and Martinelli [93] devise a BPC algorithm which includes variable fixing and route enumeration. In a post-processing phase, variable fixing is first performed based on the solution of the linear relaxation of a two-index arc-flow formulation augmented by cutting planes. Once the BPC algorithm is started, variable fixing is also carried out in the fashion of Irnich et al. [130]. Contrarily to Bettinelli et al. [134], Contardo and Martinelli [93] solve one pricing problem per depot. Their BPC algorithm comprises several families of robust and non-robust cuts. In particular, they exploit the similarity between the MDVRP and the CLRP to adapt non-robust CLRP cuts for the MDVRP, namely, the y-strong capacity cuts and the strong framed capacity cuts.

4.4.2 Profits (optional customers)

In some applications, it may not be possible to visit all the customers due to vehicle fleet limitations (e.g. a limited number of vehicles with limited capacity is available) or to a short planning horizon. In this case, the customers to serve are part of the decisions to make and some of them might not be serviced. To determine which ones to serve, a profit is associated with each customer and profit maximization is sought. For this class of routing problems, called VRPs with profits (VRPPs), only a few BP algorithms have been designed.

The team orienteering problem (TOP) is the simplest multi-vehicle variant of the VRPP. It consists of designing routes for a homogeneous fleet of vehicles in such a way that the total profit collected is maximized and the length (duration) of the routes does not exceed a given threshold. No routing costs are considered. When solving the TOP by means of a CG algorithm, the MP is defined as the linear relaxation of a set-packing model, with an additional constraint to limit the number of routes in the solution. The pricing problem is an

ESPPRC which has been solved by a labeling algorithm, except in [135] where it is solved by a procedure that enumerates subsets of customers and checks for each subset the existence of a feasible route by solving a TSP.

Boussier et al. [136] present a generic BP algorithm for the TOP and the selective VRPTW (SVRPTW). The SVRPTW is a generalized TOP where vehicle capacity and customer time windows are considered. To derive integer solutions, Boussier et al. [136] apply two branching strategies. They branch first on a customer $i \in V'$ that is visited a fractional number of times. If this is not possible, i.e., the flow through each customer vertex is integer, an alternative branching on an arc (i, j) is performed. Two cases are considered. If one of the customers i or j has been constrained to be served, two branches are created: one imposing arc (i, j) to be traversed, and the other forbidding it. Otherwise, three branches are created: one forbidding the visit to i, and two enforcing the visit to i followed or not by j. Keshtkaran et al. [137] enhanced the algorithm of Boussier et al. [136] by incorporating to the pricing algorithm several acceleration techniques discussed in Section 4.3 and by including SRCs to reinforce the MP. These cuts are also valid for a set-packing formulation.

Archetti et al. [138] introduce the capacitated TOP (CTOP), a variant of the TOP that involves a vehicle capacity constraint. They also study the capacitated profitable tour problem (CPTP) which aims at minimizing the difference between the total collected profits and the total traveling cost and considers a vehicle capacity constraint but no maximum route duration constraint. To solve both the CTOP and the CPTP, the authors adapt the BP algorithm of Boussier et al. [136]. Archetti et al. [139] enhance the BP algorithm of Archetti et al. [138] by improving the pricing labeling algorithm with some acceleration techniques and revising the branching strategies used. The new branching scheme involves branching on the number of vehicles used, on the flow through a customer, and on the flow on an arc. In the latter case, imposing a flow of one on an arc is performed by adding a constraint in the MP. This new algorithm employs a column-based restricted master heuristic to generate primal solutions throughout the search tree. Finally, Archetti et al. [140] address the CTOP with incomplete services, where a customer may be partially served. The problem is then formulated as a set-packing problem. The routes, possibly performing incomplete services, are generated by means of the labeling algorithm described in Archetti et al. [141], which exploits an expanded network that contains vertices associated with every possible pair of customer and quantity that can be delivered to this customer.

Another problem belonging to the class of VRPPs is investigated by Bulhões et al. [142], who introduce the VRP with service level constraints (VRP-SL). This problem arises in situations where the total customer demand exceeds the general capacity of supply. However, due to

commercial agreements, a minimum service level to the customers must be ensured as follows. The set of customers V' is partitioned into m disjoint groups $\mathcal{V}_k$ (i.e., $V' = \cup_{k=1}^{m} \mathcal{V}_k$ and $\mathcal{V}_k \cap \mathcal{V}_{k'} = \emptyset$ for any $k \neq k'$) where a group may represent the deliveries to a same company. Moreover, with each customer $i \in V'$ are associated a demand q_i, a profit ρ_i and a service weight w_i. This latter represents the importance of the customer in its group. The VRP-SL consists of designing profitable routes that visit some of the customers in V' (to deliver their full demands) such that, for each subset $\mathcal{V}_k$, $k = 1, \ldots, m$, a minimum service level $\phi_k \in [0, 1]$ is reached. The service level of a group k is computed as $\sum_{i \in \mathcal{V}_k^*} w_i / \sum_{i \in \mathcal{V}_k} w_i$, where $\mathcal{V}_k^* \subseteq \mathcal{V}_k$ denotes the subset of customers in $\mathcal{V}_k^*$ that are serviced in the solution. The objective function aims at minimizing the total cost and the lost profits incurred by not servicing some of the customers. Bulhões et al. [142] formulate the VRP-SL using a set partitioning model that contains binary slack variables to identify which customers are not serviced and the service level constraints. For solving the problem, they devise a rather straightforward BP algorithm.

Other VRPP variants are studied by Azi et al. [143], Archetti et al. [141, 144], Parragh et al. [145], Luo et al. [146], and Gutierrez-Jarpa et al. [147]. Given that the focus of these works is on the treatment of multiple trips, split services or pickups and deliveries, they are reviewed in Sections 4.4.4, 4.4.5 and 4.4.10.

4.4.3 Soft time windows

Soft time windows, unlike traditional time windows that pose structural restrictions on the feasible solution space, do not impact the feasibility of the routes. These restrictions are thus relaxed in the soft variant and transferred as penalties in the objective function [148]. More precisely, in the VRP with soft time windows, a vehicle can serve a customer $i \in V'$ without paying a penalty if the service is performed within $[e_i, l_i]$, but it can also serve a customer before e_i, or after l_i by paying a linear penalty proportional to the anticipation or the delay.

From a computational viewpoint, soft time windows are harder to handle than the hard time windows because the feasible solution space in the former is larger. Liberatore et al. [148] present the first exact BP algorithm that handles soft time windows in an efficient manner. In their algorithm, the labeling algorithm considers in each label a piecewise linear reduced cost function $\bar{c}(t)$ of the start of service time t. This function is equal to 0 at the source vertex and updated with each extension. When performing an extension along an arc (i, j), the reduced cost function of the new label is obtained by summing up the function from the previous label and the penalty function associated with the possible start of service times at j. In order to reduce the number of feasible extensions, the authors introduce the notion of *profitable time windows*, which are the reduced intervals of time in which a visit to a customer

may be beneficial. Arrival times out of this interval are deemed unpromising for the current pricing problem and discarded.

Bettinelli et al. [149] adapt this algorithm to solve the multi-depot heterogeneous fleet pickup and delivery problem with soft time windows. Abdallah and Jang [150] consider a hybrid variant of this problem in which penalties are accepted only in a limited zone around the hard time windows, out of which arrivals are deemed infeasible. Thus, structural constraints are not completely lost. These authors develop a tailored labeling algorithm that is shown to be efficient to limit the number of non-dominated labels.

Taş et al. [151] study the VRP with soft time windows and stochastic travel times. This work is reviewed in Section 4.4.9.

4.4.4 Multiple trips

In some applications, vehicles may return to the depot in the middle of their route to be replenished before performing additional deliveries. Such returns may be caused by a limited vehicle capacity compared to the volume of the demands, by some duration constraints imposed by the drivers' regulations or because the products delivered/collected are perishable. This feature gives rise to a class of problems, called the multi-trip VRPs (MTVRPs), where a trip is defined by a sequence of visits to customers between two visits to the depot and a route contains one or several trips. These problems are worthy of investigation if the number of vehicles is limited (or they incur fixed costs) and some constraints (such as a maximal route length or time windows) restrict the feasibility of the trip sequences. Otherwise, there exists an optimal solution with only single-trip routes and the problem becomes equivalent to the VRP.

Despite the practical importance of the MTVRP, no exact solution algorithms have been proposed for this problem prior to the recent work of Mingozzi et al. [152]. The MTVRP they consider is a direct extension of the CVRP with a limited number of vehicles and a maximum driving time per route. Their algorithm extends the one of Baldacci et al. [99], as it relies on bounding procedures to solve relaxations of two set-partitioning formulations and trip/route enumeration. In the first set-partitioning formulation, denoted SPF_1, the variables are associated with feasible trips; in the second, denoted SPF_2, they represent feasible routes. The proposed algorithm exploits the strengths of both formulations. On the one hand, generating trips is easier than generating routes. On the other hand, the bounds provided by SPF_2 are better than those yielded by SPF_1. The algorithm starts by computing a lower bound from SPF_1 by CG before performing trip enumeration and solving the resulting reduced model. If this approach does not succeed (e.g., when too many trips are

enumerated), a lower bound from SPF_2 is computed by CG using only the trips enumerated in the first phase. Route enumeration is then performed to derive a reduced SPF_2 model that is solved by a MIP solver. Both set partitioning formulations are reinforced with non-robust cuts. In particular, a strengthened version of the knapsack inequalities, called working time inequalities, is introduced for SPF_1.

Considering the delivery of perishable products, Azi et al. [143] tackle a MTVRP with time windows and a limited trip duration (MTVRPTW-LD) in which customers' service is optional. The problem can also be seen as a VRPP variant. It is formulated as a set packing problem where the variables are associated with feasible routes and the objective consists of minimizing the total profit collected from the serviced customers minus the total traveling cost. To solve this problem, the authors propose a two-phase method where a complete enumeration of all the feasible trips is carried out in the first phase and routes obtained by concatenating feasible trips are generated by CG in the second phase. Complete trip enumeration is achievable in the first phase because the number of customers in a trip is small due to the maximum trip duration. This enumeration is performed using the algorithm of Azi et al. [153] for the single-vehicle case and in which a dominance rule is applied to eliminate non-Pareto-optimal trips. Each trip is associated with a (minimum) duration, a loading time that needs to be taken into account before starting the trip, and a starting time window which preserves the trip duration. A BP algorithm is applied in the second phase. The pricing problem is an ESPPRC defined over a directed graph where the vertices represent the trips enumerated in the first phase.

Hernandez et al. [154] address the MTVRPTW-LD with mandatory services to the customers. They model the problem as a set-covering problem where the variables are associated with feasible trips and so-called *mutual exclusion constraints* (MECs) ensure that the number of available vehicles is met at all times. If the discretization of the planning horizon is too fine-grained, the number of MECs explodes and the model becomes intractable. In the proposed BP algorithm, this issue is dealt with by solving first a relaxation that considers a coarser discretization (thus, less MECs) and, if necessary, subsequent stronger relaxations obtained by refining the discretization. The pricing problem consists of finding trips with negative reduced costs or proving that none exist. This includes determining the exact schedule of each trip. To solve this problem, Hernandez et al. [154] devise a two-phase algorithm. In the first phase, trips without schedule are enumerated using a corrected version of the enumeration algorithm of Azi et al. [143]. Each trip is defined by a sequence of customers, a minimum duration, a loading time, and a starting time window. In the second phase, a minimal reduced cost schedule is determined for each enumerated trip. Given that imposing integrality requirements on the arc flow between two customers is not sufficient to derive an

optimal integer solution (a convex combination of schedules can be chosen for a given trip), the authors also develop a specific branching scheme. Given a fractional solution, branching on the arc flow between two customers is favored. If this is not possible (i.e., all arc flows are integer), then a special case of a VRPTW is defined from this solution (each trip in the solution represents a customer) and solved by an ad hoc BP algorithm to find a feasible solution to the MTVRPTW-LD if one exists. If so, the cost of this solution is equal to the current RMP solution cost and the node can be pruned. Otherwise, multiple branching nodes are created, each forbidding the usage of an arc in the current solution.

Contrarily to the previous works, Hernandez et al. [155] study the MTVRP with time windows in which no limitation on the trip duration is imposed. They present two different formulations and two BP algorithms. In the first formulation, variables are associated with routes, whereas in the second, they are associated with trips. Despite the similarity with previous methods, Hernandez et al. [155] do not benefit from trips with limited duration and, therefore, complete trip enumeration becomes impractical. Thus, the authors develop tailored labeling algorithms to solve the pricing problem ensuing from each formulation. In the BP algorithm for the first model, routes are generated by solving an ESPPRC with possible returns to the depot. Mono-directional backward labeling is performed to handle the loading time before each trip which depends on the total quantity to be delivered along the trip. In the BP algorithm for the second model which includes MECs in the MP, trips are also generated by solving an ESPPRC but, in this case, the trip schedule impacts its reduced cost because of the dual values of the MECs. To reduce the number of generated labels, Hernandez et al. [155] establish some theoretical results which allow to group together labels corresponding to the same sequence of visited customers and keep a single representative label for each group. Here again, backward labeling is used.

Finally, Muter et al. [156] consider a generalization of the MTVRP and the MDVRP called the MDVRP with inter-depot routes (MDVRPI). In the MDVRPI, a route starts and ends at the same depot, and can also stop at any intermediate depot to replenish. The routes can be seen again as sequences of trips where the starting depot of a trip may differ from its ending depot. Route duration is limited but not trip duration. Route duration is computed as the sum of the traveling times, customer service times and fixed loading times required before starting the trips. The MDVRPI is modeled as a set covering problem where the variables are associated with routes and two different BP algorithms are developed to solve it. They differ by the algorithm used to solve the ESPPRC pricing problems, one for each depot. The first algorithm is a labeling algorithm similar to that of Feillet et al. [86], which is applied on a network that contains additional vertices to represent intermediate visits at a depot. Because a vehicle can be replenished along its route, a customer can be temporarily

considered unreachable if this status is only due to vehicle capacity. The second algorithm is a two-phase algorithm. In the first phase, all non-dominated trips are enumerated like in the work of Azi et al. [143]. Furthermore, among the trips that link the same depot, only those with a negative reduced cost are kept. In the second phase, a labeling algorithm is applied on a network where the vertices represent the depots and the arcs the enumerated trips.

In Section 4.4.10, we review the work of Luo et al. [146] that considers a rich dial-a-ride problem in which multiple trips per route are possible.

4.4.5 Split services

In routing problems with split services (split deliveries or split pickups), customers are allowed to be visited more than once in order to have their demand satisfied. The fact that each demand can be split among several vehicles can yield substantial economic savings, operational flexibility, not to mention, the possibility of customers with a demand larger than a vehicle capacity to be served Archetti and Speranza [157]. Taking this option into account, the split delivery VRP (SDVRP) and the split delivery VRPTW (SDVRPTW) extend the CVRP and the VRPTW, respectively. The design of BP algorithms for these problems is not straightforward because the amount to deliver at each customer visit is also a variable decision. In the following, we start by discussing works on the SDVRPTW which is the first split delivery VRP variant to be solved by means of a BP algorithm. Note that, unless otherwise specified, the service time at a customer does not depend on the quantity delivered.

Gendreau et al. [158] propose the first BPC algorithm for the SDVRPTW and, assuming that travel costs and times satisfy the triangle inequality, prove the following properties for the SDVRPTW (the first two ensue from identical properties for the SDVRP): 1) two routes cannot have more than two customers in common; 2) no arc can appear more than once in a solution; and 3) each route visits a customer at most once. In the proposed BPC algorithm, the pricing problem only determines the vehicles routes, not the quantities to deliver. These decisions are made at the MP level using additional quantity variables and constraints, which depend on the routes generated. Therefore, the MP involves an exponential number of constraints that are generated dynamically with the generated route variables. The authors develop sufficient conditions on the optimality of a MP solution in this context. These conditions are used to define the objective function of the pricing problem which is an ESPPRC that is solved by an adapted version of the labeling algorithm of Feillet et al. [86]. To speed up the solution process, Gendreau et al. [158] solve first a relaxed MP which allows to generate columns more rapidly. Because customers may be visited more than once in the SDVRPTW, a solution may be fractional even if the arc flows are integer.

When this occurs, a sophisticated branching rule on a subset of variables described in Feillet et al. [159] is applied. Two other branching rules can be applied, namely, on the number of visits to a customer and on an arc flow.

Desaulniers [160] designs a BPC algorithm for the SDVRPTW where the quantities to be delivered to the customers visited in a route (forming a so-called *delivery pattern*) are determined in the pricing problem, which corresponds to an ESPPRC combined with the linear relaxation of a bounded knapsack problem (LP-BKP). To solve this pricing problem, a tailored labeling algorithm is developed. It exploits the property of an optimal basic solution to the LP-BKP: all variables are either at their lower or upper bound except possibly one. In the SDVRPTW context, this property translates to: in a delivery pattern of a route yielding an optimal solution to the pricing problem (called an *extreme delivery pattern*), all customers receive a full or a zero delivery, except at most one customer that can receive a partial delivery. Because convex combinations of extreme delivery patterns for the same route are allowed in the MP, all customers in a route receive a positive quantity in an optimal solution and multiple partial deliveries are possible. In the labeling algorithm, a label can be extended up to three times along an arc reaching a customer, namely, for a full delivery at this customer, a zero delivery, or a partial delivery. Because the quantity delivered in a partial delivery is a decision variable that is computed only once the route is completed, the reduced cost of a partial path is a linear function of this quantity and the dominance rule is devised to handle such reduced cost components. The MP is reinforced with adapted kPCs exploiting the structure of the SDVRPTW and with edge-flow cuts which ensue from the application to a single edge of the first property stated by Gendreau et al. [158]. Four branching strategies are applied, namely: on the total number of vehicles, on the number of times a customer is visited, on an arc flow, and on the possibility of using or not two arcs consecutively. Decisions of this last type are non robust in the sense that they require additional resources in the labeling algorithm. They are, however, necessary to ensure an exhaustive exploration of the search tree.

Archetti et al. [103] improve the BPC algorithm of Desaulniers [160]. They design a tabu search algorithm that combines the procedure of Desaulniers et al. [44] with an LP-BKP solver to generate negative reduced cost columns. They also develop three efficient separation heuristics for the kPCs. Finally, to further strengthen the MP lower bounds, they introduce three new classes of non-robust valid inequalities for the SDVRPTW: strong k-path cuts (SkPCs), strong minimum number of vehicles inequalities (SMVCs), and adapted SRCs. The SkPCs are a strengthened version of the kPCs which are not weakened by paths entering a given subset of customers more than once. The SMVCs are based on the idea that, if a customer does not receive a full delivery, it has to be visited at least twice. Finally, even if

the MP does not include set partitioning constraints, SRCs can be defined by observing that each customer cannot receive more than one full delivery.

Salani and Vacca [161] study an extension of the SDVRPTW, called the discrete SDVRPTW (DSDVRPTW), in which the demand of the customers is composed of a discrete set of items which can be delivered separately. The items delivered by a vehicle to a customer form an order and the service time at a customer depends on the delivered order. Because of this feature, several properties defined for the SDVRP(TW) are no longer valid for the DSDVRPTW. For instance, two routes can have more than one split customer in common. The authors formulate the DSDVRPTW as a set partitioning model where the set partitioning constraints are associated with the items of each customer. The pricing problem is an ESPPRC defined on a network identical to that used for the VRPTW except that the vertices are associated with feasible orders instead of customers. This network structure permits to include the service time for each possible order on the arcs.

Luo et al. [162] introduce an extension of the SDVRPTW called the SDVRPTW with linear weight-related cost, in which the travel costs per unit distance are charged based on a linear function of the load weight. To solve this problem, they devise a BPC algorithm that borrows ideas from Desaulniers [160] and Archetti et al. [103] and relies on an efficient labeling algorithm involving an aggressive dominance rule. In this labeling algorithm, each label contains a reduced cost component, which is also a function of the vehicle load but encodes information regarding the extreme delivery patterns for the corresponding route. With this reduced cost function, each label is extended once along each arc contrarily to the algorithm of Desaulniers [160] that extends it up to three times. The proposed dominance rule allows to dominate partially or fully a label by comparing it with a set of labels. In fact, all labels are stored in a dominance graph that permits efficient label comparisons.

Archetti et al. [163] present the first BPC algorithm for solving the SDVRP. The approach is similar to the one proposed by Desaulniers [160] since the pricing problem generates simultaneously the routes and their delivery patterns. The main difference is that the pricing problem is defined over an expanded network, where each customer is represented by a set of vertices, one for each possible quantity that can be delivered to it. With this network, the dual values of the customer demand constraints can be easily incorporated into the arc costs and the labeling algorithm is much simpler than that of Desaulniers [160], especially a standard dominance rule can be used. On the other hand, the performance of the algorithm heavily depends on the size of the customer demands.

Moreno et al. [164] focus on improving the lower bounds for the SDVRP. They design a robust CG algorithm that employs cutting planes defined in terms of CI variables. The

algorithm considers three families of valid inequalities: the homogeneous extended capacity cuts of Pessoa et al. [114], edge-flow cuts and split delivery cuts. The edge-flow cuts are the same as those proposed independently in Desaulniers [160]. The split delivery cuts are Chvátal-Gomory cuts derived from a single customer demand cut. The authors point out that, by formulating the SDVRP using CI variables, the knapsack structure of the problem can be exploited, enabling stronger cuts to be devised.

Archetti et al. [165] address the commodity constrained SDVRP (C-SDVRP) in which a customer can be visited more than once if its demand is composed by multiple commodities that must be treated separately. This situation occurs, for example, when different products require specific temperatures (e.g., frozen, fresh and dry food), or when different commodities must be transported in separated compartments (e.g., food and hazardous products). Like in the DSDVRPTW, the C-SDVRP allows multiple visits to a customer, but does not allow a specific commodity to be split. The authors model the C-SDVRP as a CVRP where the set of customers is replaced by the set of all requested commodities for all customers. They design a BPC algorithm where the MP ensures that all commodities are delivered and the pricing problem is defined over an expanded network, where most vertices represent a commodity/customer pair. In fact, for each customer, a subnetwork containing the related vertices is created to reduce symmetry. Furthermore, at each column generation iteration, a preprocessing step is applied to each subnetwork to deduce non-dominated subpaths with respect to vehicle capacity and the current dual values. Besides the branching strategies proposed by Desaulniers [160], a new strategy inspired by Ryan and Foster [166]'s rule is presented. It branches on whether or not two pairs of commodity/customer must be delivered in the same route. Each such decision requires adding a new resource in the label definition.

Some authors study the impact of split services in VRPP variants. Archetti et al. [141] develop a BP algorithm to solve the split delivery CTOP (SDCTOP), where each customer can be visited multiples times in order to have its demand completely fulfilled. This algorithm incorporates features from the algorithms of Archetti et al. [138] and Archetti et al. [103], in particular, it uses the expanded network defined in Archetti et al. [103]. Archetti et al. [144] investigate the SDCTOP with incomplete service, where a customer may be served only partially and can be visited by more than one vehicle. This problem can be modeled as the SDCTOP except that the constraints enforcing the complete service of the customers are replaced by constraints bounding the total quantity delivered to each customer by its demand. Therefore, the authors present an algorithm similar to the one described in Archetti et al. [141].

Parragh et al. [145] address a pickup and delivery routing problem with split services that is

discussed in Section 4.4.10.

4.4.6 Time dependency

Classical VRP variants do not capture some important features related to real-life problems such as the presence of congested routes at different times of a day. In the literature, road congestion has been represented by the consideration of time-dependent travel times, i.e., by making the travel time on an arc depend on the moment at which it is traversed. In this context arises the time-dependent VRPTW (TDVRPTW) in which the planning horizon is divided into time zones (e.g., morning, afternoon, and night) and, for each time zone, travel times are assumed to be constant on every arc, leading to a stepwise speed function over the horizon. This speed function is then used to determine a travel time piecewise linear function that satisfies the first-in-first-out (FIFO) principle [167].

Dabia et al. [168] propose the only BP algorithm to solve a variant of the TDVRPTW that minimizes the total duration of the routes. Given that the cost of a route is its duration, the reduced cost of a route cannot be computed as the sum of arc costs in the pricing problem. Therefore, in the labeling algorithm developed by Dabia et al. [168], a label associated with a customer vertex $i \in V'$ stores a piecewise linear function $\delta_i(t)$ to represent the time at which the service is completed (ready time) at i if the vehicle leaves the depot at time t. This function is non-decreasing and allows to compute the route duration in function of this departure time as $\delta_i(t) - t$ as well as the route reduced cost as $\delta_i(t) - t - c_\pi$, where c_π is the sum of the dual variables associated with each vertex visited along the route. The authors introduce a sharp dominance rule that takes into account the possibility to use different departure times from the depot for each compared label.

4.4.7 Cumulative costs

The real cost of a vehicle traversing an arc depends on many variables. Several of them are proportional to the arc distance (or travel time). Some others like the elapsed time up to each customer and the vehicle load, which impacts the fuel consumption rate, cannot be represented by the distance. In some cases, they are, however, proportional to a flow on the arc, e.g., the remaining quantity to deliver and the number of customers still to be visited, and the cost function can be defined as a product of this flow and the distance traveled. This gives rise to a class of VRPs, called the cumulative VRPs (CumVRPs) [169].

Lysgaard and Wøhlk [170] design a BPC algorithm to solve a CumVRP in which the objective aims at minimizing the sum of the arrival times at the customers. In fact, their algorithm is

very similar to the algorithm of Fukasawa et al. [43] for the CVRP. However, to assess the cost of traversing an edge e, one not only needs to know the traveling time t_e along e but also the number of customers remaining to visit after e, because t_e contributes to the arrival time of all the remaining customers. Given that this information is unavailable when building a route from the source vertex, Lysgaard and Wøhlk [170] use a backward labeling algorithm to solve the pricing problem. In this way, the customers remaining to service after traversing an edge are those that have been visited in the current backward path and, thus, the reduced cost of traversing an edge can easily be computed.

Another CumVRP variant where the cost of traversing an arc depends on the vehicle load is tackled by Fukasawa et al. [171]. This work is reviewed in the next section.

4.4.8 Environmental aspects

Recently, it has become critical to make transportation and logistics environmentally sustainable, and thus VRPs that include environmental aspects have gained considerable attention in the literature. These VRP variants often aim at minimizing greenhouse gas emissions/fuel consumption [172], or at managing the use of electric vehicles [173]. Green VRPs (GVRPs) denote VRP variants in which greenhouse gas emission or fuel consumption is considered either in the objective function or in the operational constraints [172].

Fukasawa et al. [171] design a BPC algorithm to solve the energy minimization vehicle routing problem (EMVRP). This problem belongs to the class of CumVRPs [169] and extends the CVRP by defining the arc cost as the product between the arc length and the current vehicle weight when traversing the arc. The proposed algorithm is similar to that of Fukasawa et al. [43]. These algorithms differ by the network used to generate q-routes. Because in the EMVRP, the cost of traversing an arc depends on the carried load, Fukasawa et al. [171] use an expanded network G_Q where each arc appears $Q+1$ times in the network. In G_Q, an arc (i, j, q) represents a traversal of the arc (i, j) with a load of q units. The labeling algorithm is adapted to be executed on G_Q.

The pollution routing problem (PRP, [174]) is a GVRP variant in which one must not only design vehicle routes but also determine at which speed each arc in each selected route should be traveled. The problem aims at minimizing the vehicle consumption and the drivers' wages that are proportional to route duration. Both objectives are conflicting since higher speeds yield shorter routes but larger fuel consumptions. To address a variant of the PRP which assumes constant speed along a route and variable departure time from the depot, Dabia et al. [175] adapt the BP algorithm of Dabia et al. [168] (see Section 4.4.6). The modifications concern the labeling algorithm which also considers a piecewise linear function to represent

the ready time $\delta_i(t, v)$ at a customer $i \in V'$, that does not only depend on the departure time t from the depot, but also on the vehicle speed v along the route. Because computing this function dynamically is not straightforward, the authors rather generate two functions, obtained by fixing $t = 0$ and $v = v_{max}$ (the maximum speed), that can be combined to provide all necessary information. The labeling algorithm requires new resources to compute the fuel consumption and a tailored dominance rule.

In addition to the GVRPs discussed above, another VRP variant that has environmental motivations is the electric VRP (EVRP), in which vehicles are powered by electricity and suffer from limited battery autonomy constraints [173]. When time windows are associated with the customers, the EVRP with time windows (EVRPTW) arises. In the EVRPTW, one considers all features of the VRPTW plus the following ones: additional vertices representing recharging stations, a recharging time depending on the amount of energy recharged, a battery capacity for each vehicle and, for each arc, the energy consumed by a vehicle traversing it.

Depending on the number of allowed visits to the recharging stations in each route —single (S) or multiple (M)—, and the type of battery recharging policy —full (F) or partial (P)—, four VRP variants of the EVRPTW may arise, namely, the EVRPTW-SF, the EVRPTW-MF, the EVRPTW-SP, and the EVRPTW-MP. Desaulniers et al. [176] study all these variants and propose different BPC algorithms that differ by the labeling algorithm used to generate routes. Assuming that the energy consumed along each arc of a route can be converted into a required recharging time, the battery capacity constraint is expressed in terms of the time required to recharge the consumed energy. For the EVRPTW-SF and the EVRPTW-MF, the labeling algorithms are similar (the former problem requires an additional resource to impose at most one recharge per route). In the forward algorithms, a single resource is needed to cumulate the required recharging time which also monitors the battery capacity. Given that the time required for a recharge at a station depends on the energy consumed, this time is unknown in a backward labeling algorithm. Nevertheless, Desaulniers et al. [176] devise a bidirectional labeling algorithm where the backward algorithm relies on additional resources to ensure that this time is well computed. When partial recharges are allowed in the EVRPTW-SP and the EVRPTW-MP, the trade-off between the amount recharged and the time spent for recharging leads to express the start of service time at a customer visited after a recharge as a linear function of the recharging time. This relation is modeled using three resources and requires an ad hoc dominance rule. In this case, a backward labeling algorithm symmetric to the forward one can be devised and used in a bidirectional search.

Another VRP variant that incorporates environmental aspects and exhibits similarities with the EVRPTW is studied in Andelmin and Bartolini [177]. In this variant, the vehicles

are powered by an alternative fuel (e.g., biodiesel or electricity), have limited fuel/battery autonomy, and might need to stop for refueling/recharging along their route. Each refueling stop has a fixed duration and each route is subject to a maximum duration. To model this problem, the authors use a multi-graph which contains additional arcs between every pair of vertices. For a given pair of vertices $i, j \in V'$, these extra arcs represent non-dominated refueling paths (containing one or multiple refueling stops) between i and j. The proposed solution algorithm follows the scheme of Baldacci et al. [2] (see Section 4.3.4).

4.4.9 Uncertainty

All VRPs discussed so far assume that all required input data is known in advance. However, when solving real-life problems, this is not always the case as several data may be subject to different sources of uncertainty coming from expected variations and/or unexpected events. The demands, the service times, the traveling times, and the presence of the customers are commonly considered to be stochastic according to Gendreau et al. [178]. In the last two decades (see, e.g., [179]), many researchers have turned their attention to the exact solution of stochastic VRPs (SVRPs) and robust VRPs (RVRPs).

Stochastic VRPs.

SVRPs are often considered as *a priori* optimization (or two-stage optimization) problems. In the first stage when complete information is unknown, *a priori* decisions (e.g., planned routes) must be taken. Once the uncertain information is revealed in the second stage, the planned routes may become infeasible in which case they can be revised according to predefined recourse actions or they can simply be deemed failures. In the literature, two main modeling approaches have been proposed for the SDVRPs: chance-constrained programs (CCP) or stochastic programs with recourse (SPR) [179, 180]. In a CCP, a lower bound on the probability that a given plan will be feasible once the stochastic information becomes known is imposed. This probability can be considered while solving the pricing problem [181] or by adding chance constraints directly in the MP [178]. In turn, in a SPR, the objective function consists of minimizing the expected costs, namely, the travel costs of the planned routes plus the expected recourse costs ensuing from the application of recourse actions when the routes become infeasible in the second stage. Note that the recourse actions are not arbitrary as they must obey to the chosen recourse policy for the problem at hand. Even if the SPRs are typically more difficult to solve than the CCPs, their objective function is more meaningful [180], as it considers the cost of turning the infeasible planned routes into feasible ones. On the other hand, the main advantage of the CCPs is that they allow to control

directly the probability of failure which may be important in certain cases to maintain the image of the company.

The most studied SVRP variant is the VRP with stochastic demands (VRPSD, [180]), where the demand of each customer $i \in V'$ is expressed by a random variable with an expected value $\xi(q_i)$ and a variance $\vartheta(q_i)$. Normally, the demands are assumed to be independent and to follow an additive probability distribution such as the Normal or the Poisson distribution [179]. For a route $r = (0, i_1, \ldots, i_k, n+1)$, one can thus define the cumulative expected demand $\mu_h = \sum_{\ell=1}^{h} \xi(q_{i_\ell})$ and the cumulated variance $\sigma_h^2 = \sum_{\ell=1}^{h} \vartheta(q_{i_\ell})$ at any given customer i_h, $h \in \{1, \ldots, k\}$. In the VRPSD, route r is said to be feasible if it is elementary and $\mu_h \leq Q$ for all $h \in \{1, \ldots, k\}$. This latter condition prevents routes from systematically *failing* when their feasibility is assessed. By considering a cumulative probability distribution with mean μ_h and variance σ_h^2 for customer i_h, one can determine the probability $P(\mu_h > Q)$ that a *failure* occurs at this customer. In this case, if the problem is modeled as an SPR, a recourse action (e.g., returning to the depot before continuing the route) should be applied. The objective of the VRPSD consists of minimizing the total expected cost of the routes, that can be decomposed in two parts: the deterministic cost and the *expected failure cost* (EFC). This latter is given by the probability that a route fails at a customer multiplied by the cost of returning to the depot before continuing the route (or of another recourse action).

Christiansen and Lysgaard [127] introduce the first BP algorithm for a SVRP, namely, the VRPSD. The pricing problem is a shortest path problem with 2-cycle elimination defined on an expanded network $G_S = (V_S, A_S)$. The vertex set V_S is defined as $V_S = \{(0,0,0)\} \cup \{(\mu, \sigma^2, i) : i \in V \cup \{0\}, \mu = 1, \ldots, Q, \sigma^2 = 1, \ldots, \vartheta_{\max}\}$, where $(0,0,0)$ is the source vertex and each vertex (μ, σ^2, i) can only be reached by paths representing a (partial) route ending at customer i (or at the depot if $i = 0$) and whose expected cumulative demand and variance are μ and σ^2, respectively. In this definition, $\vartheta_{\max}$ is the maximum total variance of a feasible route, i.e., with a mean demand $\mu \leq Q$, which is pre-computed by solving a knapsack problem. In arc set A_S, there exists an arc between two vertices (μ_i, σ_i^2, i) and (μ_j, σ_j^2, j) only if $\mu_j = \mu_i + \xi(q_j)$ and $\sigma_j^2 = \sigma_i^2 + \vartheta(q_j)$ with $\xi(q_j) = \vartheta(q_j) = 0$ if $j = 0$. This network structure allows to incorporate the EFCs directly in the arc costs, giving a huge advantage over the approaches requiring dynamic calculations of the EFCs. Note, however, that the construction of network G_S requires a finite number of possible expected cumulative demands and variances. For their computational tests, Christiansen and Lysgaard [127] assume that the demands are Poisson distributed, which implies $\mu = \sigma^2$. This allows to considerably reduce the size of the network G_S by eliminating all vertices (μ, σ^2, i) for which $\mu \neq \sigma^2$. The authors propose a branching strategy inspired by that of Gelinas et al. [125] (see Section 4.3.3). Instead of branching on time windows, they branch on the expected cumulative demand at a customer

$i \in V'$ that is visited by two routes containing the vertices (μ_1, σ_1^2, i) and (μ_2, σ_2^2, i), where $\mu_1 < \mu_2$. Choosing a value $\delta \in [\mu_1, \mu_2)$, one can impose that expected cumulative demand μ at customer i is such that $\mu > \delta$ in one branch and $\mu \leq \delta$ in the other.

Addressing the same VRPSD, Gauvin et al. [182] develop a BPC algorithm that improves the algorithm of Christiansen and Lysgaard [127] by adding several features (*ng*-routes, heuristic pricing, bidirectional labeling, cuts) found in state-of-the-art algorithms for VRPs. Moreover, they introduce a new dominance rule which allows the comparison of labels associated with two different vertices, as long as these vertices represent the same customer i. Finally, to derive integer solutions, they branch on customer sequences or cutsets (see Section 4.3.3).

Dinh et al. [181] study the chance-constrained VRPSD (CCVRPSD) and develop a BPC algorithm for solving it which is derived from that of Fukasawa et al. [43]. They introduce the concept of *chance-constraint feasible route* (CC route), which is a route for which the probability of satisfying the capacity constraint is greater than or equal to a threshold $1 - \epsilon$ for a given $\epsilon \in (0, 1)$. Contrary to previous methods used to solve the VRPSD, the algorithm of Dinh et al. [181] does not require the customer demands to be independent, though it needs the quantiles of the random variable defined by the sum of the demands in any subset of customers. Because the pricing problem remains strongly $\mathcal{NP}$-hard, even if the q-route relaxation is used, the application of the BPC algorithm of Fukasawa et al. [43] to solve the CCVRPSD is not straightforward. To circumvent this issue, the authors propose to relax the capacity chance constraint in the pricing problem and to handle it through capacity constraints in the MP. To strengthen the relaxed pricing problem, they add a knapsack constraint which ensures that all CC routes remain feasible. Regarding the capacity inequalities added to the MP, Dinh et al. [181] discuss how strong lower bounds on the number of vehicles required to serve a subset of customers can be computed cheaply considering the quantiles of the probability distribution of the total demand for this customer subset.

Noorizadegan and Bo [183] devise a BP algorithm for the CCVRPSD, assuming that every customer demand follows a Poisson distribution. They, however, mention that, if verifying the satisfaction of the chance capacity constraint for a route is possible, the algorithm can be adapted to other distributions for which the sum of their random variables follows a known distribution. These requirements are necessary because the pricing problem is solved by means of a labeling algorithm, in which the chance capacity constraint is verified after every label extension. In the case of the Poisson distribution, a single additional resource specifying the average demand along the route (which is equal to its variance) is required. Other resources may be required for other distributions. Standard dominance rule is then applied.

Taş et al. [151] address a VRP with soft time windows and stochastic travel times. In this problem, the travel times follow Gamma probability distributions and the customers may be served outside their time windows under the payment of a penalty. The objective consists of minimizing a weighted cost function of the total transportation costs and the service costs. The transportation costs depend on the total distance traveled, the number of vehicles used, and the total expected overtime of the drivers, whereas the service costs are due to early and late arrivals at the customers. Waiting at customers is forbidden. The authors developed a BP algorithm where the pricing problem is solved by a labeling algorithm derived from that of Feillet et al. [86] and in which routes cannot be deemed infeasible because of a time window violation. A label includes an expected reduced cost component which is computed by adjusting the departure time from the depot using a golden section search method. The dominance rule is standard except that it includes an additional condition which compares the expected reduced cost of the routes if they were both starting from the depot at the optimal departure time of the potentially dominated route.

Errico et al. [184, 185] study the VRPTW with stochastic service times (VRPTW-ST) which is defined like the VRPTW except that the customer service times are expressed by mutually independent probability distributions. Both works address the problem of dispatching technicians to perform maintenance operations, where vehicle capacity is not a concern. Nevertheless, the proposed BPC algorithms can easily be adapted to handle vehicle capacity if required. The works of Errico et al. [184, 185] differ in the way the VRPTW-ST is modeled, namely, as a CCP in the former and as a SPR in the latter.

In Errico et al. [184], the VRPTW-ST is formulated as a set partitioning model with an additional constraint imposing a lower bound on the success probability of the whole route plan. Applying logarithm properties and by taking advantage of the service times being mutually independent, the authors show how this non-linear constraint can be linearized. Errico et al. [184] develop a BPC algorithm where the pricing algorithm must handle the dual from this additional constraint considering a stochastic time component. Indeed, the coefficient of a route variable λ_r in this constraint depends on its success probability. To determine this probability as the route is being built in the labeling algorithm, the mass function of the earliest start of service time at each vertex along the route must be computed. Consequently, the authors replace the usual time component in a label associated with a vertex $i \in V'$ by $l_i - e_i + 1$ components, namely, one for each integer time t in the time window $[e_i, l_i]$ which is denoted $\bar{M}_i(t)$. These components define the cumulative probability distribution of the earliest start of service time at i restricted to $[e_i, l_i]$. They also allow to decompose the reduced cost of a route into single arc contributions obtained from conditional probabilities. In the labeling algorithm, the dominance rule combines the traditional dominance criteria

with the concept of *stochastic dominance* which compares the components $\bar{M}_i(t)$ for each time $t \in [e_i, l_i]$.

When formulating the VRPTW-ST as a SPR, Errico et al. [185] propose two alternative recourse policies. Both policies assume that the real service time at a customer is first evaluated just before starting the service at a customer. If this evaluation indicates that the time window at the next customer will be missed, then the service is skipped at the current customer in the first policy, denoted `C`, or at the next customer in the second one, denoted `N`. In both policies, a penalty is paid for skipping the service at a customer. To ensure a high-level service, a route is considered feasible if it requires at most one recourse action and its success probability is larger than or equal to a given threshold. Each recourse policy induces a different pricing problem, which is solved by a tailored labeling algorithm adapted from that devised in Errico et al. [184]. To handle policy `C`, two new resources are added to the label definition. Policy `N` is more complex because skipping the service at the next customer i_N due to a long service time at a customer i_C implies taking a shortcut in the route from i_C to the customer i_F (or depot) visited after i_N, yielding a smaller traveling cost. Given that this situation is observed at vertex i_N in the labeling algorithm, vertex i_F is unknown at that time and the cost saving realized by cutting short from i_C to i_F cannot be evaluated at vertex i_N. Consequently, Errico et al. [185] replace the reduced cost component of a label by an *incomplete reduced cost* component, and compute lower and upper bounds on the expected reduced cost of any extension of the corresponding route that reaches the depot. In the dominance rule, these bounds are used to modify the reduced cost criterion. The rest of the labeling algorithm is similar to the algorithm used for policy `C`.

Robust VRPs.

Stochastic programming approaches require the knowledge of the probability distributions of the uncertain data. These distributions are not always available and, when they are, they must respect certain assumptions to yield tractable SVRPs. Robust optimization is an alternative approach that expresses the uncertainty of the data by means of an uncertainty set Ψ. This set is generally parameterized by a perturbation vector that indicates how the problem data diverge from their nominal values. It can be defined by taking into account past observations or considering some existing knowledge about the probability distributions. Given a set Ψ, a solution is said to be *robust feasible* if it satisfies all the realizations of the constraints defined over the uncertainty set. The problem of finding the best robust feasible solution is called the *robust counterpart problem* [186]. It is shown by Bertsimas et al. [187] that the robust counterpart problem of a linear problem is also a linear problem of

polynomially-bounded size.

Very few BP algorithms exist for solving RVRPs. Lee et al. [188] consider the VRP with customer deadlines and travel time/demand uncertainty, and develop a BP algorithm in which the robust aspect is embedded in the pricing problem. Two uncertainty sets Ψ_t and Ψ_d associated with the travel times and the demands, respectively, are defined as follows. For each arc $(i,j) \in A$, the travel time can take a value in the interval $[\hat{t}_{ij}, \hat{t}_{ij}+\delta_{ij}]$, where $\hat{t}_{ij}$ is the nominal travel time along this arc and δ_{ij} is the maximum deviation from this nominal value. Because it is unlikely that a vehicle will be delayed on every arc of a route, the uncertainty set is limited by the maximum number of delayed segments Γ_t which ensures a certain degree of robustness of the routes. Set Ψ_t is defined as: $\Psi_t = \{\left(\tilde{t}_{ij}\right)_{(i,j)\in A} \,|\, \tilde{t}_{ij} = \hat{t}_{ij} + \delta_{ij}\nu_{ij}, 0 \leq \nu_{ij} \leq 1,\ \forall (i,j) \in A,\ \sum_{(i,j)\in A} \nu_{ij} \leq \Gamma_t\}$. Set Ψ_d is defined similarly. To ensure that the generated routes are feasible for all possible realizations of the travel times and demands in these uncertainty sets, the labeling algorithm relies on $\Gamma_t + \Gamma_d$ additional resources. The Γ_t resources store the sums of the k largest travel time deviations δ_{ij} associated with the arcs traversed in a route, for $k = 1, \ldots, \Gamma_t$. The other Γ_d resources play the same role for the demands. With these resources, a standard dominance rule can be used.

Souyris et al. [189] also suggest a robust approach to tackle a real-life problem of designing routes for maintenance technicians where the customer service times are uncertain and a soft start of service time deadline is imposed at each customer. Furthermore, it is assumed that the technicians have different initial locations and do not have to return to a depot at the end of the day. Given that the routes are subject to a maximum duration, some customers might not be serviced. The objective function is a weighted sum of the traveling costs, penalties for delayed customer service, and penalties for unserviced customers. The authors present two models. The first assumes that the customer service times are independent, resulting in a VRP with soft time windows where all service times are replaced by their worst case. The second model assumes that the service times of the customers visited by each technician are correlated, i.e., deviations will not occur at all customers visited along a route. Assuming that the service time at a customer $i \in V'$ belongs to the interval $[\hat{s}_i, \hat{s}_i + \delta_i]$, the uncertainty set is given by $\Psi^k = \{\left(\tilde{s}_i\right)_{i\in V'} \,|\, \tilde{s}_i = \hat{s} + \nu_i, 0 \leq \nu_i \leq \delta_i,\ \forall i \in V',\ \sum_{i\in V'} \nu_i \leq \Gamma^k\}$, where Γ^k is an upper bound on the total service time deviation for technician k. To solve this RVRP, Souyris et al. [189] develop a BP algorithm. They formulate the resulting pricing problem as a mixed integer linear program but solve it using constraint programming.

Dinh et al. [181] and Noorizadegan and Bo [183] mention how their BP algorithms discussed in Section 4.4.9 can be adapted to solve a *distributionally robust* extension of the CCVRPSD for which the probability distributions are not known precisely, but they are assumed to

belong to an ambiguity set of possible distributions. Routes must be robust in the sense that they must satisfy the chance capacity constraint for all probability distributions in this set.

4.4.10 Pickups and deliveries

Pickup and delivery problems (PDPs) consist of finding least-cost vehicle routes to satisfy a set of requests. Each request is defined by a pickup location, a delivery location, and a volume of merchandise (or a number of persons) to be transported from the pickup location to the delivery location. Several requests can be transported simultaneously by a vehicle as long as the onboard load does not exceed vehicle capacity. A large number of PDP variants have been studied in the literature and classified by Berbeglia et al. [190]. So far, BP algorithms have been devised for *one-to-one PDPs* and *one-to-many-to-one PDPs*. In the former class, each request corresponds to a specific commodity and, in general, the pickup and delivery locations differ from the depot. In the latter class, commodities originating from the depot (e.g., filled beer bottles) must be delivered to a set of customers, while other commodities destined to the depot (e.g., empty beer bottles) must be picked up at a possibly different set of customers.

Given the large number of works on PDPs, we divide our review in the following five subsections: time windows (Subsection 4.4.10), ride time constraints (Subsection 4.4.10), transfers (Subsection 4.4.10), loading constraints (Subsection 4.4.10), and simultaneous pickups and deliveries (Subsection 4.4.10). The first four subsections are devoted to one-to-one PDPs, whereas the last one concerns one-to-many-to-one PDPs.

Time windows.

Most BP algorithms for PDPs have been developed for PDPs subject to time window constraints. Here, we focus our attention on the one-to-one PDP with time windows (PDPTW), which can be defined on the following network $G = (V, A)$. Let n be the number of requests to satisfy. Let $P = \{1, \ldots, n\}$ be the set of pickups and $D = \{n + 1, \ldots, 2n\}$ the set of deliveries such that $i \in P$ and $n + i \in D$ are associated with the same request. Set V is then given by $V = P \cup D \cup \{0, 2n + 1\}$, where the vertices 0 and $2n + 1$ denote the origin and the destination depot, respectively. Set A is formed by the feasible links connecting the vertices in V. In particular, for $i \in P$, there is no arc $(n + i, i)$ given that the pickup of a request must be performed before its delivery. Note that set P is often used to represent the set of requests. When designing BP algorithms to solve the PDPTW, one of the most complicating aspects is the presence of *pairing* and *precedence* constraints. The pairing constraints specify that, if a vehicle performs the pickup of a request, it must also perform its delivery. The

precedence constraints impose that pickups be performed before the corresponding deliveries. These constraints must be dealt with when constructing the routes in the pricing problem. Note that the MP does not need to include a set partitioning constraint for each pickup and each delivery, as one for each request suffices. This allows some flexibility when defining the modified arc costs. Indeed, the dual cost for fulfilling a request can be associated with its pickup vertex i, its delivery vertex $n + i$, or be arbitrarily distributed among both of them (see [191]).

The first BP algorithm for the PDPTW is due to Dumas et al. [192] who design a labeling algorithm in which the label definition is extended by adding a set of visited vertices $V(L)$ and a set of open requests $\mathcal{O}(L)$, i.e., pickup vertices whose corresponding delivery vertices have not been visited yet. Strong dominance rules can be obtained if one assumes that the arc modified cost matrix in the pricing problem respects the delivery triangle inequality (DTI), i.e., a visit to a delivery vertex never lowers the path reduced cost. To ensure that this condition is satisfied, the dual variables of the request covering constraints are transferred to the arcs exiting a pickup vertex. When comparing two labels L_1 and L_2 for dominance, the previous property allows to discard label L_2 if the condition $\mathcal{O}(L_1) \subseteq \mathcal{O}(L_2)$ holds (together with the usual conditions), rather than $\mathcal{O}(L_1) = \mathcal{O}(L_2)$. Dumas et al. [192] also propose preprocessing techniques to tighten the time windows and eliminate infeasible arcs, as well as unreachability criteria that can be applied while solving the pricing problem. Finally, they devise a branching strategy based on request ordering. Savelsbergh and Sol [193] improve this BP algorithm by developing two heuristics for generating feasible routes: one considering a reduced network and another based on construction and improvement algorithms.

Ropke and Cordeau [194] present the first BPC algorithm for the PDPTW that adds valid inequalities to the algorithm of Dumas et al. [192]. This practice, however, yields a modified cost matrix that no longer satisfies the DTI, which is required to apply the enhanced dominance rule. The authors then propose a procedure to transform any arbitrary cost matrix into an equivalent matrix where the DTI is ensured.

Baldacci et al. [195] extend the exact solution framework of Baldacci et al. [99] to deal with the PDPTW. They develop two new bounding procedures that exploit the properties of the PDPTW.

The application of bidirectional search in the context of the PDPTW is not straightforward. The strong dominance rules of Dumas et al. [192] are not compatible with a backward labeling. When the direction of the labeling changes from forward to backward, the cost matrix must respect the pickup triangle inequality (PTI). Two techniques are suggested to overcome this issue. First, Bettinelli et al. [149] redistribute the weights over the arcs of the network, so

that the two above properties hold simultaneously. It requires solving a linear program for every label extension in the labeling algorithm, which is computationally costly. Second, Gschwind et al. [191] use two modified cost matrices when employing bidirectional search: one for the forward labeling, that respects the DTI, and another for the backward labeling, where the PTI holds. With these matrices, the bidirectional search is performed normally except that, when merging the partial paths, the reduced cost of the complete route must be adjusted.

Ride time constraints.

The dial-a-ride problem (DARP) is an important variant of the PDP in which the hard time windows are replaced totally or partially by ride time constraints that prevent the commodities from staying too long inside the vehicle [196]. These constraints are also called *dynamic time windows*, in contrast to the *static time windows* used in the PDPTW. Important applications of the DARP can be found in the context of people transportation, or in the delivery of perishable products. Ropke [197] proposes a simple way to adapt a BPC algorithm for the PDPTW to the DARP. It consists of adding *infeasible path elimination constraints* (IPECs) to the MP to handle the ride time constraints.

Gschwind and Irnich [198] develop a BPC algorithm for the DARP that handles the ride time constraints in the pricing problem. They observe that although harder subproblems need to be solved, much better bounds can be achieved, resulting in a positive trade-off. The additional burden in the pricing problem arises from the following two observations: 1) to ensure time window feasibility, every vertex should be visited as early as possible; 2) to ensure ride time feasibility, every pickup vertex should be visited as late as possible. These two observations are clearly in conflict with each other. To address this issue, the authors design two labeling algorithms. In the first, the label definition is the same as for the PDPTW but a label L_1 can dominate another label L_2 only if $|\mathcal{O}(L_1)| \leq 1$. In the second labeling algorithm, each label keeps track of the *latest possible delivery time* for each open request in function of the start of service time at the vertex associated with the label. This allows to define a strong dominance rule that can be applied for any value of $|\mathcal{O}(L_1)|$. Several valid inequalities devised for the PDPTW [199] and for the DARP [200] are used to reinforce the MP.

Gschwind [201] introduces the synchronized PDP (SPDP), a generalization of the DARP in which a delivery vertex must be visited within a time interval defined in terms of minimum and maximum ride times associated with its corresponding pickup vertex. Because the pickup and the delivery vertices have to be visited by the same vehicle, these additional constraints

are also called *intra-route synchronization constraints* (general synchronization aspects are discussed in [202]). Due to the complexity of the resulting pricing problems when considering simultaneously minimum and maximum ride times, the author proposes four BPC algorithm variants which consider different pricing problems obtained by relaxing none or some features: the $\text{PP}^{\text{max}}_{\text{min}}$ that handles all the route constraints of the SPDP; the PP_{min} that only addresses minimum ride times; the PP_{max} that is similar to the one proposed by Gschwind [198] and consider only maximum ride times; and the PP_{relax} that relaxes both ride time constraints. Contrarily to the PP_{max}, in which maximum ride times may yield conflicting decisions with customer starting service times, minimum ride times are in conformity with the decisions of visiting customers as early as possible in the PP_{min}. Thus, the proposed labeling algorithm is similar to the one used to solve the pricing problem of the PDPTW. Regarding the $\text{PP}^{\text{max}}_{\text{min}}$, the presence of both minimum and maximum ride time constraints complicates considerably the pricing problem and a complex dominance rule is devised for the labeling algorithm. Except for the first BPC algorithm in which all routing constraints are handled in the pricing problem, the other three algorithms use IPECs in the MP to enforce the relaxed constraints as proposed by Ropke [197].

Parragh et al. [145] tackle the DARP with split requests and profits (DARPSRP) in the context of people transportation. In the DARPSRP, a revenue is associated with each request $i \in P$ which might remain unserved. If served, all the associated people must be transported to their destination, though they can be split into different vehicles. The objective aims at maximizing the total profit which is expressed by the total revenues collected minus the traveling costs. Due to the presence of paired pickups and deliveries in the DARPSRP, some of the known SDVRP solution properties [158, 160] are no longer valid and cannot be exploited. In particular, an optimal solution might necessarily contain a route which visits more than once the same pickup vertex. In the BP algorithm of Parragh et al. [145], the pricing problem consists of generating routes with positive reduced cost, in which all route feasibility constraints are respected, namely: vehicle capacity, customer time windows, maximum ride time, precedence and pairing. The authors devise a labeling algorithm inspired by that of Ropke and Cordeau [194]. A set of resources is defined to take into account the number of people associated with the onboard requests. Besides, two additional resources are used to compute the minimum possible duration of a route that would complete all open requests and, consequently, to check the feasibility of a label extension. When performing an extension over an arc (i, j), two cases are analyzed: if $j \in P$ (pickup vertex), as many labels as different possible splits of the people remaining to pick up at j may be generated. If $j \in D$, only a label associated with the number of people to be delivered is created. Moreover, unpromising extensions are avoided by applying a tailored dominance rule along

with completion bounds.

Qu and Bard [203] study a variant of the PDPTW that arises in the context of people transportation and considers a heterogeneous fleet, multiple shipment types and configurable vehicle capacities. No ride time constraint are imposed but the objective function penalizes each minute of ride time and of waiting before a pickup. In this problem, each pickup point (customer) is associated with a set of load requirements, i.e., specific conditions for transporting the items. For example, such requirements may ask for enough space for transporting a passenger and its wheelchair or walker. In turn, each vehicle type can be set a priori in different configurations to accommodate different types of demand. For example, a given vehicle can transport three seating passengers or only two if they bring wheelchairs. The problem consists of designing feasible routes such that the number of used vehicles, the total traveled distance and the customers traveling times are minimized. The decisions also include selecting the configuration of each vehicle to be set up before starting its route. There is a pricing problem for each vehicle type. Each problem is solved by a labeling algorithm where a label includes the list of feasible vehicle configurations. To determine the list of feasible configurations, a generalized assignment problem is solved.

Finally, Luo et al. [146] solve a rich variant of the DARP (R–DARP) that considers, among other attributes, multiple trips, heterogeneous vehicles, multiple request types, configurable vehicle capacities, optional services, and manpower planning. The R-DARP consists of designing routes such that the number of satisfied requests is maximized, the total distance traveled by the vehicles is minimized, and work regulations are respected. To solve this problem, the authors develop a BPC algorithm that relies on a new trip-based model similar to that of Azi et al. [143] and Hernandez et al. [154] (see Section 4.4.4). In this model which considers only non-dominated trips that are identified a priori by a label-setting algorithm, trips are combined to generate working shifts for the drivers. The proposed BPC algorithm includes two types of cuts, namely, IPECs and Benders cuts that forbid vehicle routes for which no feasible worker schedules exist.

Transfers.

Classic PDPs are subject to pairing and precedence relations which impose that the pickup and the delivery of a request be performed by the same vehicle. However, in some applications, some requests can be transferred from one vehicle to another at predetermined locations called *transfer points*. This extension of the PDP is called the PDP with transfers (PDPT) and is particularly challenging when the pickup and delivery points have time windows. Indeed, in this case, the PDPT involves a new type of precedence constraint stipulating that

any transferred request must be delivered to its transfer point by a first vehicle before being picked up by a second vehicle which will transport it to its final destination.

Masson et al. [204] address a special case of the PDTP with time windows called the PDP with shuttle routes (PDPS), where it is assumed that the requests have individual pickup points but they share a limited number of delivery points. In the PDPS, *pickup routes* visit first pickup locations and end by a visit to a single delivery point which may not be the final destination of the picked up requests, i.e., it may be a transfer point for some of these requests. In this case, direct *shuttle routes* are used to transport the transferred requests to their delivery point. The PDPS arises when, e.g., people need to be transported from their home to schools or rehabilitation centers. Masson et al. [204] design two different set partitioning formulations with additional constraints for the PDPS. Both formulations involve binary variables associated with the pickup routes. In the first, nonnegative integer variables indicate the number of shuttles required between each pair of delivery points without specifying the transported requests. In the second which provides better lower bounds, binary variables associated with shuttle routes and their transported requests are used. A BPC algorithm is designed for each formulation. In both algorithms, there is one pricing problem for each possible delivery (transfer) point. It is an ESPPRC that involves a resource indicating the latest time at which the vehicle can arrive at the delivery vertex without violating the time windows of the picked up requests. In the second algorithm, a pricing problem is also needed to generate the shuttle routes. This problem corresponds, in general, to a binary knapsack problem. However, if a single person is associated with each request, it can be solved more efficiently using a sorting algorithm for each pair of delivery points.

Motivated by situations in which part of the operations can be performed at a certain cost by the public transit available, Ghilas et al. [205] introduce a generalization of the PDPT called the PDPTW with scheduled lines (PDPTW-SL). In this problem, each request can be served directly by a single vehicle or indirectly using two vehicles and a scheduled public transit line. In the former case, the request is picked up by a vehicle which brings it to a transfer point that is serviced by a scheduled line. This request is then transported by the public transit to another transfer point. Finally, a second vehicle picks up the request at this second transfer point to deliver it to its final destination. Such indirect trips may be beneficial for transportation companies when pickup nodes are located far from the delivery nodes. To solve the PDPTW-SL, Ghilas et al. [205] present a BPC algorithm that relies on a set partitioning MP with several sets of additional constraints that enforce the capacity of the scheduled lines and proper synchronization between the vehicle routes and the scheduled lines transporting the transferred requests. Furthermore, the authors develop a forward labeling algorithm for solving the ESPPRC pricing problems, one for each vehicle type and depot.

This labeling algorithm uses labels that store three sets of requests: the open requests that have been picked up but not yet delivered, the completed requests, and the requests that have been pickup up at a transfer point. Note that a picked up request may be delivered to its final destination or to a transfer point. Finally, the authors devise a bidirectional search labeling algorithm for these pricing problems.

Loading constraints.

In some contexts, the order in which the requests are collected or the shape (volume and/or size) of their items may have operational implications that must be taken into account by the models and algorithms. This feature is, in general, referred to as *loading constraints*. The following works concern loading constraints induced by the order of the collected requests.

Cherkesly et al. [206] address a PDPTW subject to a LIFO (last-in-first-out) loading constraint, denoted PDPTWL, which arises when handling operations should be avoided, for instance, in the transportation of heavy, dangerous or large items. This LIFO constraint assumes that, in a vehicle, the items of the requests are stored in a single stack (e.g., from the front of the vehicle towards the access door at the back). Consequently, when requests are picked up, they are put on top of the stack. Furthermore, a delivery point can only be visited if its corresponding request is on top of the stack. The authors propose three BPC algorithms that differ in the way the LIFO constraint is handled. In the first, the LIFO constraint is imposed through *LIFO-infeasible path cuts* in the MP. In the second algorithm, this constraint is enforced directly in the pricing problem which requires the development of a new labeling algorithm. In particular, new label components indicating the position in the stack of each onboard request are considered and a specialized dominance rule is devised. This second approach yields better lower bounds than the first one, but the pricing problem is much harder to solve. Seeking a trade-off between these two algorithms, the authors introduce a third algorithm that incorporates LIFO-infeasible path cuts in the MP whenever needed and a labeling algorithm which partially imposes the LIFO policy. Indeed, given a positive integer κ, the LIFO constraint must be respected by all onboard requests that have remained in the top κ positions in the stack. Consequently, as soon as a request falls below the top κ positions, it is ejected from the stack and can be delivered in any order with respect to the other requests that have also been ejected.

The last two algorithms of Cherkesly et al. [206] mentioned above are adapted by Cherkesly et al. [207] to solve the PDPTW with multiple stacks (PDPTWMS). In this problem, each stack has a limited capacity and the LIFO loading constraint applies to each individual stack. The presence of multiple stacks is addressed in the labeling algorithms by means of

components that indicate: the accumulated load under each request in a given stack, the relative position between requests inside the same stack, and the simultaneous presence of two requests in the same vehicle, but not in the same stack. Besides handling the loading constraints correctly, the use of these label components favors the elimination of symmetry between the identical stacks as a stack can be identified by its top request. When extending a label to a delivery vertex, a single label is created. However, when it is extended to a pickup vertex, multiple labels can be created, namely, one for each stack on which the request can be put. In the algorithm where the LIFO loading constraints are partially imposed in the pricing problem, it is possible to generate a path that does not respect these constraints with the proposed loading plan (assignments of the requests to the stacks). Nevertheless, with a different loading plan, the path may be feasible. Therefore, once a path with an infeasible loading plan is generated, the algorithm first checks if there exists a feasible loading plan for this path by solving a shortest path problem with resource constraints. This allows to reduce the number of LIFO-infeasible path cuts added to the MP. Finally, let us mention that these algorithms rely on valid inequalities including the RCCs that were adapted by Côté et al. [208] for the pickup and delivery TSP with multiple stacks.

Introduced by Iori and Riera-Ledesma [209], the double VRP with multiple stacks differs from the PDPTWMS by the fact that pickups and deliveries are carried out in two different regions which are far apart. Therefore, one can assume that, in each route of an optimal solution, all pickups are performed before the deliveries. Nevertheless, the routes are still subject to the pairing constraints between the pickup and the delivery points as well as the LIFO policy for each stack. For this problem, the authors devise two set partitioning formulations: one that considers complete routes, i.e., routes containing pickup and delivery operations; and another that relies on partial routes dedicated to a single region (either all pickups or all deliveries) and ensures the compatibility of the selected partial routes through additional constraints. The former formulation is solved by means of a BP algorithm that employs a pricing algorithm similar to that of Ropke and Cordeau [194]. In this labeling algorithm, however, the arcs traversed by a route in each region are stored in the label and are later considered when performing dominance. The FIFO policy is only checked once a route is completed. For the latter formulation, a BPC algorithm is developed in which only the capacity constraint is handled in the pricing problem and the pairing and LIFO loading constraints are managed by adding IPECs in the MP when needed.

It may be beneficial in some cases to allow the rehandling of the load at intermediate points of a route, if this brings extra savings when compared with a strict LIFO policy. In this context, Veenstra et al. [210] introduce the PDPTW with handling operations as an extension of the PDPTWL. In their study, rehandling operations are only allowed at delivery points.

Two rehandling policies are analyzed: one that only allows compulsory rehandling, i.e., only the requests on top of the delivered request must be rehandled; and another that accepts compulsory and preventive rehandlings, meaning that all the requests in a vehicle can be rehandled at once. No cost is incurred for rehandling but extra service time, which depends on the number of items rehandled, must be accounted for each rehandling operation, yielding infeasible extensions due to the time windows. Veenstra et al. [210] propose a BPC algorithm for each handling policy. In both cases, the labeling algorithm takes into account the relative position between any pair of requests like in Cherkesly et al. [207]. However, two requests are considered to be at the same position if their most recent rehandling operation was carried out at the same location.

Simultaneous pickups and deliveries.

The VRP with simultaneous pickups and deliveries (VRPSPD) is an extension of the CVRP, where each customer requires a delivery, a pickup, or both. This problem belongs to the class of one-to-many-to-one PDPs because all delivered items originate from the depot, whereas all picked up items must be transported to the depot. It models real-life situations related to reverse logistics activities such as the distribution of beverages and the collection of empty cans and bottles.

Angelelli and Mansini [211] develop the first BP algorithm to tackle the VRSPD with time windows (VRPTWSPD). They assume that each customer $i \in V'$ is associated with a non-negative delivery demand q_i^D and a nonnegative pickup demand q_i^P. They notice that, if either $q_i^D > q_i^P$, $\forall i \in V'$, or $q_i^P > q_i^D$, $\forall i \in V'$, then an optimal solution to the VRPTWSPD can be found by solving a VRPTW in which the demand of every customer is given by its net demand. In the algorithm proposed for the general case, the pricing problem is solved by a labeling algorithm which relies on two dependent resources to enforce vehicle capacity: one stores the total demand collected in the route and another indicates the minimum capacity required along the route, i.e., the maximum load carried simultaneously. These resources were previously stated in [8].

Dell'Amico et al. [126] design a BP algorithm to solve the VRPSPD, where the pricing problem is also solved by a labeling algorithm. To manage vehicle capacity in this algorithm, they rely on two resources that are different but equivalent to those used by Angelelli and Mansini [211]. Dell'Amico et al. [126] propose various strategies to accelerate the search for negative reduced cost routes. In particular, they apply bidirectional search and limit the number of customers that can be visited in each forward and backward route to $\lceil \bar{\sigma}/2 \rceil$, where $\bar{\sigma}$ is an upper bound on the number of customers that can be visited in a route. This

upper bound is computed by solving two binary knapsack problems, namely, one for the pickup demands and one for the delivery demands. Furthermore, they use a state-space relaxation obtained by replacing in the label definition the customer resource vector $\mathcal{E}$ with a single component $|\mathcal{E}|$ counting the number of customers visited. This relaxation allows the generation of routes with cycles. To keep the RMP tractable, variables associated with routes having a reduced cost greater than a given threshold are removed from the RMP when it reaches a maximum size. These columns are then stored in a pool for a maximum number of iterations. While in the pool, these routes are priced directly and added to the RMP if their reduced cost becomes negative. Finally, two new branching strategies are proposed: branching on cycles and branching on resource windows. When a cycle occurs in a fractional solution, it can be eliminated by creating multiple child nodes obtained by imposing/forbidding the use of some arcs in the cycle. The second type of branching is only applied when a customer vertex is involved in two cycles of two different routes. It is an adaptation of the branching proposed by Gelinas et al. [125] and corresponds to branching on the resource windows of the two resources handling the vehicle capacity constraint. Using a label definition similar to the one considered in Dell'Amico et al. [126], [42] show how to apply bidirectional search for solving the pricing problem arising in the VRPSPD.

Subramanian et al. [212] extend the BPC algorithm of Fukasawa et al. [43] for the CVRP to the VRPSPD. They introduce the concept of *pd*-routes. A *pd*-route starts and ends at the depot, and at any visited customer, the sum of all collected items plus the sum of all items to be delivered does not exceed vehicle capacity. In the BPC algorithm, the pricing problem consists of finding *pq*-routes with negative reduced cost. To speed up the labeling algorithm, the authors use completion bounds that are computed by finding the least-cost path from each customer to the depot, but only considering deliveries.

Finally, Gutierrez-Jarpa et al. [147] study the VRP with deliveries, selective pickups and time windows (VRPDSPTW), where all deliveries are mandatory whereas pickups are optional and partial pickups are not allowed. Performing a pickup at a customer yields a revenue. The authors propose a BP algorithm that not only solves the VRPDSPTW, but also five other variants of the problem involving: single or combined (pickup and delivery) demands, single or multiple visits; and backhauls. The MP includes set partitioning constraints for the deliveries and set packing constraints for the pickups. The labeling algorithm handles the vehicle capacity using the two resources proposed in Desaulniers et al. [8]. To address the variants allowing combined demands and backhauls, changes in the underlying network are required: in particular, one pickup and one delivery vertex is created for each customer with a combined demand. In the variants where the combined demands of a customer may be satisfied with multiple visits, the elementarity of the routes is ensured by considering two

resources for each customer, i.e., one per vertex.

4.5 Conclusion

The VRP was introduced in 1959 by Dantzig and Ramser [16]. Since then, a large number of VRPs have been studied, leading to one of the most prolific research areas in operations research. VRPs are, in general, $\mathcal{NP}$-hard and large-sized practical instances are usually solved using heuristics. Nevertheless, the advances achieved in the past decades on the exact BPC algorithms have led to the exact solutions of instances with more than 300 customers for the CVRP and 200 customers for the VRPTW. In this paper, we surveyed the methodological and modeling contributions made on the BP/BPC algorithms for VRPs since the 1990s. In Section 4.3, we discussed in details the proposed generic tools that can be applied for most VRP variants. In Section 4.4, we reviewed the main contributions that are specific to a VRP variant. We believe that this survey paper will help the researchers to identify more easily the contributions made on BP/BPC algorithms for vehicle routing so that they can further improve state-of-the-art algorithms and work on unexplored ideas and topics. Given the large number of papers that have recently been published on this subject, we can only expect that the literature will continue to grow in this domain, especially to address complex VRP variants including, e.g., synchronization constraints or further uncertainty factors.

Acknowledgments

This work was supported by Fonds de recherche du Québec – Nature et technologies (FRQNT) under the grant 181909 and the Natural Sciences and Engineering Research Council (NSERC) of Canada under the grants 2017-05683 and 2013-435824. This support is gratefully acknowledged.

CHAPTER 5 ARTICLE 2: SELECTIVE ARC-NG PRICING FOR VEHICLE ROUTING

Authors: Luciano Costa, Claudio Contardo, Guy Desaulniers, and Diego Pecin
Submitted to International Transactions in Operational Research, 2020 [2].

Abstract. Column generation algorithms for solving vehicle routing problems often rely on a relaxed pricing subproblem where routes may be non-elementary and which is solved by a labeling algorithm. This pricing algorithm is said to be selective if it can discard non-elementary paths that may be Pareto-optimal but guarantees finding at least one (not necessarily elementary) path of negative reduced cost when there exists at least one elementary path of negative reduced cost. In this paper, we propose a selective pricing mechanism for a recently introduced variant of the *ng*-route relaxation, in which the neighborhoods are associated with arcs instead of nodes. We extend a state-of-the-art set-based dominance criterion for this problem and show, by means of an exhaustive computational campaign, that the resulting mechanism is effective at reducing the number of non-dominated labels as compared to the original pricing algorithm for the same problem. As a result, typically shorter computing times are required to compute similar lower bounds when the proposed mechanism is embedded within a column generation solver.

Keywords. *Column generation, shortest path problem with resource constraints, selective pricing, vehicle routing.*

5.1 Introduction

The vehicle routing problem (VRP) is a classic optimization problem arising in many applications in logistics. Since the seminal work of Dantzig and Ramser [16], many researchers and practitioners have attempted to tackle this problem as efficiently as possible. Unfortunately, solving the VRP is $\mathcal{NP}$-hard as the traveling salesman problem (TSP, [110]) can be reduced to a VRP in polynomial time. Over the years, several variants of the VRP have been proposed to model the different attributes associated with real-life problems, namely: heterogeneous fleet, multiple depots, combined pickup and delivery, split services, and vehicles with special capabilities (e.g., electric vehicles), among others. For recent compendiums and a broad view of the recent trends in vehicle routing applications, the reader is referred

[2] Available at [213].

to [3] and [17].

Nowadays, the dominant methodology for solving many variants of the VRP is branch-price-and-cut applied to a set-partitioning-type formulation [9], which can be described generically as follows. Let N^+ be a set of customers that need to be serviced by exactly one vehicle each. Let Ω be the set of feasible vehicle routes, where the feasibility of a route depends on the VRP variant considered. With each route $r \in \Omega$, associate a cost c_r and, for each customer $i \in N^+$, a binary parameter a_i^r that indicates whether or not customer i is visited in route r. Furthermore, define a binary variable ω_r, which is equal to 1 if route r is selected in the solution, or 0 otherwise. With this notation, a pure set-partitioning formulation for a VRP is as follows:

$$\min \quad \sum_{r \in \Omega} c_r \omega_r \tag{5.1}$$

$$\text{s.t.} \quad \sum_{r \in \Omega} a_i^r \omega_r = 1 \qquad \forall i \in N^+ \tag{5.2}$$

$$\omega_r \in \{0, 1\} \qquad \forall r \in \Omega. \tag{5.3}$$

The objective function (5.1) aims at minimizing the total routing cost. Constraints (5.2) impose a single visit to each customer $i \in N^+$. Finally, constraints (5.3) define the domain of the variables. For certain problem variants, additional constraints and variables may be required, yielding a set-partitioning-type model. Note also that, under certain conditions, model (5.1)–(5.2) can be replaced by a set-covering model, obtained by replacing the equality in (5.2) by a greater-than-or-equal relation.

Formulations like (5.1)–(5.3) have an exponential number of variables with respect to the number of customers. Handling these variables all at once may be computationally prohibitive. This drawback can be overcome using branch-and-price, that is a branch-and-bound algorithm applying column generation to solve the linear relaxations encountered throughout the search tree. Column generation is an iterative algorithm that solves at each iteration a restricted master problem (RMP) and a pricing subproblem. The RMP is defined as the linear relaxation of (5.1)–(5.3) considering only a subset of its variables. It is solved by a linear programming solver to yield a pair of primal and dual solutions. The pricing subproblem looks for negative reduced cost variables to add to the RMP. For most VRPs, it is cast as an elementary shortest path problem with resource constraints (ESPPRC) defined over an application-dependent network and solved by a labeling algorithm. When negative reduced cost columns are identified, they are added to the RMP before starting a new iteration. Otherwise, the column generation algorithm stops and the current RMP solution yields a lower bound. To strengthen the linear relaxations, cutting planes can be added dynamically to

yield a branch-price-and-cut algorithm.

As mentioned above, the pricing subproblem is often an ESPPRC, which is strongly $\mathcal{NP}$-hard [38]. At the opposite, the subproblem arising from completely dropping the elementarity constraints, called the shortest path problem with resource constraints (SPPRC), is solvable in pseudo-polynomial time [20], but proves to be weak when the resource constraints are not sufficiently tight. To overcome this issue, several authors have focused their attention on strengthening the bounds by partially imposing path elementarity. Two main frameworks can be found in the literature to deal with the issue of partially addressing the elementarity constraints in an efficient manner: decremental state-space relaxation (DSSR, [41, 98]); and cycle elimination [1, 2, 84, 92].

In DSSR, the elementarity constraints are imposed gradually. Starting from a pure SPPRC, the path with the least reduced cost is checked for elementarity . If it is, an optimal solution has been found; otherwise, a cycle is detected and the elementarity requirements are strengthened for the subsequent iterations. Cycle elimination techniques, in turn, are closely related to the notion of route relaxations. In the seminal article of Desrochers et al. [84], the authors presented the first successful application of column generation for VRPs. They considered the vehicle routing problem with times windows (VRPTW, [79]) and solved by branch-and-price. The authors recognized the difficulty of the underlying pricing subproblem and described for the first time the SPPRC without cycles of length 2 (*2-cyc*-SPPRC). Unlike the ESPPRC, this problem can be solved in pseudo-polynomial time. Later, Irnich and Villeneuve [92] studied the *k-cyc*-SPPRC, for values of k that range between 3 and 5, which provides bounds that are much stronger than those obtained by the *2-cyc*-SPPRC, yet, in longer computing times. Baldacci et al. [2] introduced the *ng*-route relaxation (*ng*-SPPRC), which relies on a memory mechanism to allow/disallow revisiting nodes that have been visited recently within a route. Thus, a feasible path may still contain cycles, but those cycles should be long and have a limited negative impact on the resulting bounds. Some hybrid strategies have been proposed by Martinelli et al. [97], who successfully achieved improved results by adapting DSSR to tackle the *ng*-SPPRC, and Contardo et al. [89], who showed that elementary paths can be obtained by imposing partial elementarity in an escalated way within a column-generation-based solver for routing problems.

More recently, Bulhões et al. [1] proposed the *arc-ng*-SPPRC, an extension of the classical *ng*-SPPRC, in which the neighborhoods are associated with arcs instead of nodes. The authors proposed a novel set-based dominance mechanism. Arc-based neighborhoods allow a better management of the undesirable cycles. However, when embedded into a DSSR framework, they usually induce many more iterations compared to a traditional node-based *ng*-SPPRC.

Desaulniers et al. [107] introduced the concept of *selective* pricing when the elementarity constraints are dropped at least partially. A pricing algorithm (associated with a column generation pricing subproblem) is said to be *selective* if it can discard non-elementary paths that may be Pareto-optimal but guarantees finding at least one path (possibly with cycles) of negative reduced cost when there exists at least one elementary path with a negative reduced cost. Such an algorithm can stop when it proves that no such elementary paths exist, even if there are still some non-elementary paths with a negative reduced cost. A selective pricing strategy can be used to derive valid dual bounds, and those bounds have the potential of being stronger than those that would be achieved using a non-selective pricing strategy. The authors illustrate the selective pricing strategy on the *ng*-SPPRC, but the algorithm is quite demanding and requires the addition of several data structures and sub-procedures, whose overhead is not always beneficial in the long run.

In this article, we extend the set-based dominance criterion of Bulhões et al. [1] for the *arc-ng*-SPPRC by making it selective. Unlike the original approach of using additional data structures to keep track of the non-dominated labels [107], the proposed mechanism requires minimal modifications of the labeling algorithm and is therefore easy to implement. As a result, the proposed selective pricing strategy proves to be faster in practice than the original (non-selective) *arc-ng*-SPPRC when embedded within a column generation algorithm for solving benchmark instances of the VRPTW.

The remainder of this article is organized as follows: Section 5.2 presents some of the main route relaxations proposed in the literature, with an emphasis on the *arc-ng*-SPPRC. In Section 5.3, we describe the selective *arc-ng*-SPPRC, provide some theoretical proofs of its correctness, and discuss how this new framework extends the traditional *arc-ng*-SPPRC. In Section 5.4, we report the results of our computational experiments to show the impact of considering the new framework when tackling some hard VRPTW instances. Finally, some conclusions are drawn in Section 5.5.

5.2 Route relaxations

As discussed in the previous section, the natural way of modeling pricing subproblems arising from most VRP applications is by considering an ESPPRC. However, due to the high computational complexity of this problem, route relaxations have been preferred to design efficient column-generation-based algorithms. In this section, we discuss some of the main route relaxations proposed in the literature, namely: the SPPRC, which is obtained when all elementarity requirements are relaxed; and the *ng*-SPPRC and *arc-ng*-SPPRC, which tend to avoid the formation of cycles containing nodes that might be located close to each

other. To simplify the exposition, we consider that the pricing subproblem is defined over a digraph $G = (N, A)$, where $N = \{0, \ldots, n, n+1\}$ is its set of nodes and $A \subseteq N \times N$ its set of arcs. Nodes 0 (source) and $n+1$ (sink) represent the origin and the destination of a path, respectively, whereas $N^+ = N \setminus \{0, n+1\}$ is the set of customer nodes. The following discussion can be easily extended to other networks.

5.2.1 The SPPRC

The SPPRC can be formally described as follows. Consider network G and let $\mathcal{R}$ be a set of resources, which are used to model some route feasibility rules for the problem at hand, such as vehicle capacity, time windows, or precedence, among others [81]. The objective of the SPPRC is to find a least-cost path, not necessarily elementary, from node 0 to node $n+1$, in such a way that the path respects the following resource constraints.

With each node $v \in N$ and resource $r \in \mathcal{R}$, define resource windows $[e_v^r, l_v^r]$, with $0 \leq e_v^r \leq l_v^r$, representing the allowed limits on the consumption of each resource r by a path ending at node v. Specifically, it is assumed that $l_0^r = 0$ and $l_{n+1}^r = H^r$, where H^r represents the maximum consumption allowed for resource r when arriving at the sink.

With each arc $(u, v) \in A$, we associate a modified routing cost $\bar{c}_{uv} \in \mathbb{R}$ that depends on the RMP dual values such that the sum of the modified costs of the arcs composing a path between nodes 0 and $n+1$ is equal to the reduced cost of the corresponding route variable. For example, if we denote by α_i, $i \in N^+$, the dual variables associated with constraints (5.2), then the modified cost of arc $(u, v) \in A$ is given by:

$$\bar{c}_{uv} = \begin{cases} c_{uv} - \alpha_u & \text{if } u \in N^+ \\ c_{uv} & \text{otherwise (i.e., } u = 0), \end{cases} \tag{5.4}$$

where c_{uv} is the routing cost along arc (u, v). With each arc (u, v), we also associate a vector of resource consumptions $(\gamma_{uv}^r)_{r \in \mathcal{R}} \in \mathbb{N}_+^{|\mathcal{R}|}$ (e.g., travel time, load delivered, etc.). In many classical SPPRC variants, it is assumed that, if a path ending at node v uses less than e_v^r for some resource $r \in \mathcal{R}$, it is still possible to consume the extra amount of resource, so as to respect the lower bound e_v^r. In other words, when a path is extended along an arc (u, v), assuming the resource consumptions at node u given by ξ_u^r, $\forall r \in \mathcal{R}$, they are updated at node v with the resource extension functions $\xi_v^r \leftarrow \max\{e_v^r, \xi_u^r + \gamma_{uv}^r\}$, $\forall r \in \mathcal{R}$. The extension is deemed *resource-feasible* (i.e., it satisfies the resource constraints) if $e_v^r \leq \xi_v^r \leq l_v^r$, $\forall r \in \mathcal{R}$, otherwise it is deemed infeasible. Note that, depending on the problem at hand, the resources may vary along the paths according to more complex resource extension functions [82]. For the sake of simplicity and without loss of generality, we limit our discussion to the above

resource extension functions.

5.2.2 The *ng*-SPPRC of Baldacci et al. [2]

Currently, the state-of-the-art –and widely adopted– framework to eliminate cycles is the *ng*-route relaxation (*ng*-SPPRC) proposed by Baldacci et al. [2]. The *ng*-SPPRC differs from the *2-cyc*-SPPRC and *k-cyc*-SPPRC because the notion of length of a cycle does not depend on the number of arcs (or edges) traversed along the cycle, but rather on the notion of proximity between the nodes that belong to it. In the *ng*-SPPRC, cycles are forbidden whenever they contain customers that are close to each other. For every customer $v \in N^+$, one considers a *ng*-set (neighborhood) $\mathcal{N}_v \subseteq N^+$. Because the goal is to prevent short cycles, it is common to define the *ng*-set $\mathcal{N}_v$ as containing the Δ geographically closest customers to v, and the vertex v itself, where $\Delta > 0$ is a predefined parameter. Let $P = \{v_1, \ldots, v_h = v, \ldots, v_{h'} = v, \ldots, v_\ell\}$ be a path generated while solving the *ng*-SPPRC. The cycle $H = \{v_h = v, \ldots, v_{h'} = v\}$ is feasible if and only if $v \notin \cap_{i=h+1}^{h'-1} \mathcal{N}_{v_i}$, i.e., v does not belong to the *ng*-set of at least one intermediate node in the cycle. The theoretical complexity of the *ng*-SPPRC is also pseudo-polynomial if $|\mathcal{N}_v| \leq \Delta + 1$ for every $v \in N^+$. In this case, a labeling algorithm can solve the *ng*-SPPRC in pseudo-polynomial time with a multiplying constant that depends on $2^{2\Delta}$. The *ng*-SPPRC normally provides much tighter bounds than those provided by the *k-cyc*-SPPRC in comparable CPU times. In fact, experiments carried out by Poggi and Uchoa [94] using some hard capacitated VRP instances have shown that *ng*-sets with $\Delta = 8$ yield bounds similar to those obtained by the *5-cyc*-SPPRC, but spending a time comparable to the *4-cyc*-SPPRC. For this reason, the *ng*-SPPRC has become the route relaxation of preference ever since.

5.2.3 The *arc-ng*-SPPRC of Bulhões et al. [1]

In a recent article by Bulhões et al. [1], the *ng*-SPPRC is extended to consider neighborhoods associated with arcs instead of nodes. We call this problem the *ng*-SPPRC with arc-based neighborhoods, and denote it *arc-ng*-SPPRC. The *arc-ng*-SPPRC can be defined as follows. With every arc $a = (u, v) \in A$, we associate a *ng*-set $\mathcal{N}_a \subseteq N^+$ that corresponds to the set of customer nodes that can be remembered by arc a. The *ng*-sets $\mathcal{N}_a$ may be defined according to any suitable criterion. Most successful implementations use some notion of distance to favor neighborhoods containing nodes that are close to each other. An example is to consider the set of nodes which are close to both extremities of the arcs, i.e., $\mathcal{N}_a \subseteq \mathcal{N}_u \cap \mathcal{N}_v$, with $\mathcal{N}_u$ and $\mathcal{N}_v$ defined as for the *ng*-SPPRC. Let $P = (a_0, a_1, \ldots, a_p)$ be a partial path, composed of arcs $a_0 = (0, v_1)$, $a_1 = (v_1, v_2)$, $\ldots$, $a_p = (v_p, v_{p+1})$ and $\pi(v_p)$ be the memory of this path

upon arrival at node v_p. An extension using arc $a = (v_p, v_{p+1})$ is deemed *arc-ng-feasible* only if $v_{p+1} \notin \pi(v_p)$. The memory upon arrival to v_{p+1} is updated according to the formula $\pi(v_{p+1}) \leftarrow (\pi(v_p) \cap \mathcal{N}_a) \cup \{v_{p+1}\}$. If $\mathcal{N}_a = \mathcal{N}_u \cap \mathcal{N}_v$, for every arc $a = (u, v) \in A$, the *arc-ng*-SPPRC coincides with the classical *ng*-SPPRC, and, therefore, it is more general.

By considering the notions presented in Section 5.2.1, a path $P' = (v_1 = 0, \ldots, v_{p+1} = n+1)$ is said to be feasible only if every prefix $P'_i = (v_1, \ldots, v_i)$, $i \in \{2, \ldots, p+1\}$ is both resource- and *ng*-feasible. The cost of P' is given by $c(P') = \sum_{1 \leq q \leq p} c_{v_q, v_{q+1}}$. The *arc-ng*-SPPRC consists of finding a feasible path P' of minimum total cost.

Not only this mechanism allows a better management of the undesired cycles, but also yields a sharper dominance rule. Indeed, while the traditional dominance rule for the *ng*-SPPRC allows only pairwise comparisons, Bulhões et al. [1] introduce a novel set-based dominance criterion that includes the former as a particular case.

Labeling algorithm

A general description of a labeling algorithm similar to the one adopted by Bulhões et al. [1] to solve the *arc-ng*-SPPRC is as follows. A label L encodes a partial path $(v_1 = 0, \ldots, v_p)$ from the source until reaching a node v_p through the following data structures: a terminal node $v(L) = v_p$; resource consumptions $\xi^r(L)$, $r \in \mathcal{R}$; a (reduced) cost $\bar{c}(L)$; a memory $\Pi(L)$; and a pointer $pred(L)$, which indicates the predecessor of label L'.

An extension to a node w over an arc $a = (v_p, w)$ yields a label L' with the associated data structures computed as follows:

$$v(L') \leftarrow w \tag{5.5}$$

$$\xi^r(L') \leftarrow \max\{e^r_w, \xi^r(L) + \gamma^r_a\} \qquad \forall r \in \mathcal{R} \tag{5.6}$$

$$\bar{c}(L') \leftarrow \bar{c}(L) + \bar{c}_{v_p w} \tag{5.7}$$

$$\Pi(L') \leftarrow (\Pi(L) \cap \mathcal{N}_a) \cup \{w\} \tag{5.8}$$

$$pred(L') \leftarrow L. \tag{5.9}$$

Label L' is deemed feasible if and only if it is both resource- and *ng*-feasible. For the sake of conciseness, we denote from now on an extension of label L to a node w by $L \oplus w$.

A generic representation of a labeling algorithm analogous to the one considered by Bulhões et al. [1] is outlined in Algorithm 1. Although they have implemented a bi-directional labeling algorithm [98], we present a mono-directional (forward) labeling algorithm for the sake of simplicity and because it illustrates the main parts of their algorithm. Similarly to what

is done in most of the recent implementations (e.g., [45, 46]), Bulhões et al. [1] adopt label buckets to store non-dominated labels. This data structure allows a more efficient application of dominance rules. For each node $v \in N$ and possible consumption ξ^{r^*} of a given resource $r^* \in \mathcal{R}$, one defines a bucket $\mathcal{B}(v, \xi^{r^*})$ that stores all the non-dominated labels associated with node v and having consumption ξ^{r^*} for resource r^*. It is initialized as empty and enlarged dynamically. The choice for the resource defining the buckets may vary according to the problem at hand (e.g., load or time). In Algorithm 1, the set $\mathcal{V}$ keeps the unextended labels waiting in a queue. This set is kept sorted at all times in lexicographic order with regard to the consumption of resource r^*, i.e., the labels with lowest consumptions of r^* are positioned before those with larger consumptions [40]. As mentioned before, the maximum consumption allowed for resource r^* is H^{r^*}. The set $\mathcal{A}(L) = \{a = (u, v) \in A : u = v(L), v \in N \setminus \Pi(L)$ and $L \oplus v$ is resource-feasible$\}$ contains the arcs that can be used to extend L. For two labels L and L', $L \prec L'$ means that L dominates L' (according to the rules defined in Sections 5.2.3 and 5.3). Let L_0 be the label encoding an empty path of cost zero, with zero resource consumptions, empty memory and such that $pred(L_0) = nil$. Algorithm 1 returns the label L_{best} representing a path with lowest cost. This labeling algorithm extends non-dominated labels to create new labels (line 8), coupled with a dominance rule to discard non-promising labels (lines 13 and 15). The dominance procedure prevents a combinatorial explosion and, therefore, keeps the computational burden within reasonable limits.

Multiple partial label dominance

Dominance in Bulhões et al. [1] is not only performed pairwise. The authors use a sharper *multiple partial label dominance rule* similar to that introduced by Irnich and Villeneuve [92] for the *k-cyc*-SPPRC. To formalize this idea in the context of the *arc-ng*-SPPRC, we use the notion of *weak* and *strong* dominance described in Irnich and Villeneuve [92]. Let L and L' be two feasible labels. Label L is said to *weakly dominate* L' if all the following conditions hold:

$$v(L) = v(L'), \tag{5.10}$$

$$\xi^r(L) \leq \xi^r(L'), \qquad \forall r \in \mathcal{R} \tag{5.11}$$

$$\bar{c}(L) \leq \bar{c}(L'). \tag{5.12}$$

Conditions (5.10)–(5.12), however, do not ensure a proper dominance of L' by L. In fact, depending on the label memories $\Pi(L)$ and $\Pi(L')$, it may still be possible for L' to be extended to a node w that would not be a *ng*-feasible extension of L. This would happen for every

Algorithm 1: Labeling algorithm

$\mathcal{V} \leftarrow \{L_0\}$, $L_{best} \leftarrow nil$, $\bar{c}_{min} \leftarrow +\infty$,
$\mathcal{B}(v, \xi^{r^*}) \leftarrow \varnothing$, $\forall v \in N, \forall \xi^{r^*} \in \{0, 1, \ldots, H^{r^*}\}$
while $\mathcal{V} \neq \varnothing$ **do**
  Let L be the first label in $\mathcal{V}$
  Set $\mathcal{V} \leftarrow \mathcal{V} \setminus \{L\}$
  Set $\mathcal{B}(v(L), \xi^{r^*}(L)) \leftarrow \mathcal{B}(v(L), \xi^{r^*}(L)) \cup \{L\}$
  for all $a \in \mathcal{A}(L)$ **do**
    Let L' be the label resulting from extending L along arc a using (5.5)–(5.9)
    if $v(L') = n + 1$ **then**
      if $\bar{c}(L') < \bar{c}_{min}$ **then**
        Set $L_{best} \leftarrow L', \bar{c}_{min} \leftarrow \bar{c}(L')$
      end if
    else if there exists no label $L'' \in \mathcal{B}(v(L'), \xi^{r^*}(L'))$ such that $L'' \prec L'$ **then**
      Set $\mathcal{B}(v(L'), \xi^{r^*}(L')) \leftarrow \mathcal{B}(v(L'), \xi^{r^*}(L')) \cup \{L'\}$
      Remove from $\mathcal{V}$ all labels L'' such that $L' \prec L''$
    end if
  end for
end while
return L_{best}

node $w \in \Pi(L) \setminus \Pi(L')$. The additional condition

$$\Pi(L) \subseteq \Pi(L') \tag{5.13}$$

ensures the correctness of the pairwise dominance.

Bulhões et al. [1] realized that condition (5.13) may be unnecessarily too restrictive. In fact, if a label L weakly dominates another label L', one may restrict the extensions of L' only to the nodes $w \in \Pi(L) \setminus \Pi(L') \cup \{j \in N \setminus \Pi(L') \,|\, \mathcal{N}_{(v(L),j)} \cap (\Pi(L) \setminus \Pi(L')) \neq \varnothing\}$, i.e., the nodes where L' can be directly extended but not L or the nodes that remember at least one such node. This brings out the following set-based dominance criterion. Let $\mathcal{L}$ be a collection of labels $(L_i)_{i \in \mathcal{L}}$. It is said to *strongly dominate* a label $L' \notin \mathcal{L}$ if the two following conditions hold:

i. L_i weakly dominates L' for all $i \in \mathcal{L}$,

ii. for every *ng*-feasible extension of L' to a node w, there exists $i \in \mathcal{L}$ such that L_i can also be extended to w and $\Pi(L_i \oplus w) \subseteq \Pi(L' \oplus w)$.

Conditions (i.) and (ii.) combined assure that any path obtained by feasibly extending label L' would be dominated. Hence, L' can be discarded. Moreover, the authors also demonstrate

that, whenever a weakly dominating label L_i is identified, one can immediately restrict feasible extensions of L', by discarding those that would be dominated by subpath $L_i \oplus w$ for all $w \in N \setminus \Pi(L_i)$ such that $L_i \oplus w$ is resource-feasible. As a result, not only full dominance reduces the number of labels, but also partial dominance contributes to avoiding certain extensions. On the other hand, their computational experiments showed that it may be computationally expensive to store the set of dominated extensions for each label. Consequently, as a compromise, instead of representing this set explicitly, they rely on an implicit representation which requires to only extend the memory $\Pi(L')$ of a dominated label L' along a dominated extension, thus, avoiding the extension of the label cost and resource components.

5.3 Selective *arc-ng*-SPPRC

In this section, we redefine strong dominance. For the sake of readability, we present the mechanism in two steps. First of all, we show that the pairwise comparison criterion (5.13) can be replaced by:

$$\Pi(L) \subseteq \Pi(pred(L')) \cup \{v(L')\}, \tag{5.14}$$

Proposition 1 below formalizes the correctness of this criterion. Its proof relies on the following lemma.

Lemma 1. *A labeling algorithm using conditions* (5.10)–(5.12), (5.14) *to establish pairwise dominance produces, for every label E representing an elementary partial path P, at least one non-dominated label L such that $v(L) = v(E)$, $\xi^r(L) \leq \xi^r(E)$, $\forall r \in R$, $\bar{c}(L) \leq \bar{c}(E)$, and $\Pi(L) \subseteq V(E)$, where $V(E)$ denotes the set of customer nodes visited by P.*

Proof. The proof works by induction on the size p of $P = \{v_1 = 0, \ldots, v_p\}$. If $p = 1$, then E is non-dominated and $L = E$. If $p \geq 2$, let us assume that E is obtained by extending another label E' over an arc $(v_p, v_{p+1}) \in A$, with $v_p = v(E')$ and $v_{p+1} \notin V(E')$. Using the induction hypothesis, the labeling algorithm generates a non-dominated label L' such that $v(L') = v(E')$, $\xi^r(L') \leq \xi^r(E')$, $\forall r \in R$, $c(L') \leq c(E')$, $\Pi(L') \subseteq V(E')$. Denote by L'' the label produced by extending L' along arc (v_p, v_{p+1}). It is easy to see that L'' is a feasible extension of L', and that $v(L'') = v(E)$, $c(L'') \leq c(E)$ and $\xi^r(L'') \leq \xi^r(E)$, $\forall r \in R$, hold. In addition, $\Pi(L'') \subseteq \Pi(L') \cup \{v_{p+1}\} \subseteq V(E') \cup \{v_{p+1}\} = V(E)$. If L'' is not dominated, then $L = L''$. Otherwise, if L'' is dominated by a non-dominated label L''', then L''' satisfies $v(L''') = v(E)$, $c(L''') \leq c(L'') \leq c(E)$, $\xi^r(L''') \leq \xi^r(L'') \leq \xi^r(E)$, $\forall r \in R$, and

$\Pi(L''') \subseteq \Pi(pred(L'')) \cup \{v_{p+1}\} = \Pi(L') \cup \{v_{p+1}\} \subseteq V(E') \cup \{v_{p+1}\} = V(E)$. In this case, $L = L'''$. □

From this lemma, we deduce the following result.

Proposition 1. *If there exists an elementary path P from 0 to $n+1$ with a negative reduced cost, then a labeling algorithm based on the dominance rule* (5.10)–(5.12) *and* (5.14) *finds at least one (possibly non-elementary) path P' from 0 to $n+1$ with a negative reduced cost.*

Proof. Assume that there exists such a path P which is represented by a label E. If E is not dominated, then $P' = P$. Otherwise, according to Lemma 1, there exists a non-dominated label L such that $v(L) = v(E)$ and $\bar{c}(L) \leq \bar{c}(E)$. This label represents path P'. □

To illustrate the strength of this new dominance rule, let us consider the following example. Let L_1 and L_2 be two labels such that $v(L_1) = v(L_2) = 1$, $\bar{c}(L_1) \leq \bar{c}(L_2)$, $\xi^r(L_1) \leq \xi^r(L_2)$, $\forall r \in R$, and let us assume that $\Pi(L_1) = \{1, 2, 3\}$ and $\Pi(L_2) = \{1, 2, 4\}$. Although L_1 weakly dominates L_2, the latter cannot be discarded according to (5.10)–(5.13) because $\Pi(L_1) \nsubseteq \Pi(L_2)$ (Figure 5.1). Now, let us assume that label L_2 was extended from label $pred(L_2)$ with $v(pred(L_2)) = 2$, over the arc $a = (2, 1)$, and such that $\mathcal{N}_a = \{1, 2, 4\}$ and $\Pi(pred(L_2)) = \{2, 3, 4\}$. We now have that $\Pi(L_1) = \{1, 2, 3\} \subseteq \{1, 2, 3, 4\} = \Pi(pred(L_2)) \cup \{1\}$, and label L_2 can be safely discarded according to condition (5.14) (Figure 5.2).

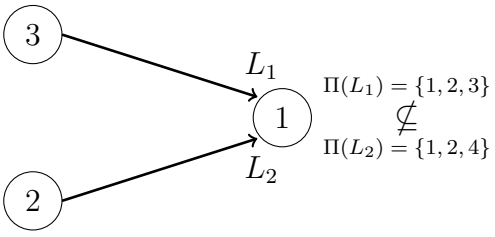


Figure 5.1 Standard dominance

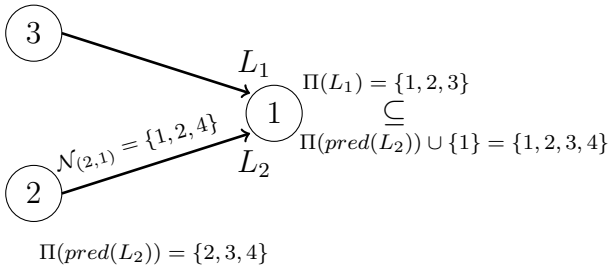


Figure 5.2 Selective dominance

Observe that, in this example, it might be feasible to extend label L_2 to node 3 and that this extension might subsequently yield non-dominated ng-paths between 0 and $n + 1$, possibly an optimal one. Nevertheless, when label L_2 is dominated according to conditions (5.10)–(5.12) and (5.14), this extension is not possible and the algorithm remains valid because the discarded ng-paths are not elementary (they visit node 3 more than once). The pricing is, thus, selective.

The second step in the presentation of our selective set-based dominance rule is stated in a corollary of the pairwise dominance condition, and extends the strong dominance criterion given by conditions (i.)-(ii.).

Corollary 1. *For a given label $L^\star$, an extension to a node $w \in N^+ \setminus \Pi(L^\star)$ can be safely omitted if there exists a label L such that all the following conditions hold:*

$$v(L) = v(L^\star \oplus w) = w, \tag{5.15}$$

$$\bar{c}(L) \leq \bar{c}(L^\star \oplus w), \tag{5.16}$$

$$\xi^r(L) \leq \xi^r(L^\star \oplus w), \qquad \forall r \in R \tag{5.17}$$

$$\Pi(L) \subseteq \Pi(L^\star) \cup \{w\}. \tag{5.18}$$

In fact, one can observe that (5.15)–(5.18) correspond to (5.10)–(5.12) and (5.14) when setting $L' = L^\star \oplus w$, showing that, if L exists and $L^\star$ was extended along arc $(v(L^\star), w)$, the resulting label $L' = L^\star \oplus w$ would be dominated by L.

Similarly to the idea proposed by Bulhões et al. [1], the dominance rule given by conditions (5.15)–(5.18) intends to consider some available information to identify dominated label extensions. Yet, the dominance rule described in Corollary 1 is stronger than the one proposed by Bulhões et al. [1] because it uses the selection mechanism given by condition (5.18), which is stronger than condition (5.13) (with $L' = L^\star \oplus w$). Another difference between the two algorithms is that our set-based dominance rule does not assume L and $L^\star \oplus w$ to be extended from the same node (see Figure 5.3), unlike that of Bulhões et al. [1] (see Figure 5.4). In these cases, label $L^\star \oplus w$ can be identified as dominated by L without extending the cost and resource components of $L^\star$ in the algorithm of Bulhões et al. [1] and without extending the memory of $L^\star$ in our selective algorithm. Therefore, if the cardinality size of vectors Π is much larger than $|\mathcal{R}|$ (the number of resources), the selective dominance rule proposed in our algorithm can yield an additional speedup.

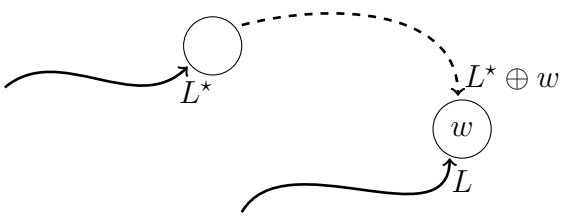


Figure 5.3 Selective set-based comparison

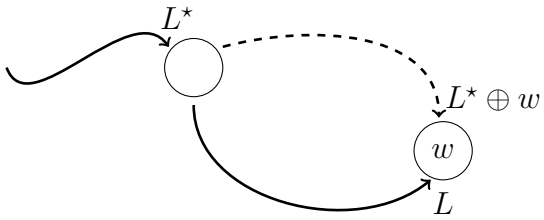


Figure 5.4 Bulhões et al.'s mechanism

One way to reinforce dominance relations (5.14) and (5.18) is by adding to the set Π of the dominating label, the unreachable nodes, i.e., the nodes to which extending the path would not be resource-feasible (equivalently, it is possible to do the same for relation (5.13)). Assuming that the resource consumptions γ_{ij}^r, $(i,j) \in A$ satisfy the triangle inequality for each resource $r \in R$, the set of unreachable nodes of a label L is defined as $\mathcal{U}(L) = \{w \in N^+ |\ \exists r \in R \text{ such that } \xi^r(L) + \gamma_{v(L),w}^r > l_w^r\}$. Given that L and $L^\star \oplus w$ cannot be feasibly extended to any node in $\mathcal{U}(L)$ and $\mathcal{U}(L^\star \oplus w)$, respectively, these sets can be included in the right-hand side of their respective dominance conditions (5.14) and (5.18), which can be rewritten as:

$$\Pi(L) \subseteq \Pi(pred(L')) \cup \{v(L')\} \cup \mathcal{U}(L') \tag{5.19}$$

$$\Pi(L) \subseteq \Pi(L^\star) \cup \{w\} \cup \mathcal{U}(L^\star \oplus w). \tag{5.20}$$

To avoid computing the unreachable set of labels $L^* \oplus w$ before testing dominance condition (5.20), the following slightly weaker condition can be used:

$$\Pi(L) \subseteq \Pi(L^\star) \cup \mathcal{U}(L^\star) \cup \{w\}. \tag{5.21}$$

5.4 Computational experiments

To assess the effectiveness of the proposed selective dominance rules, we performed a series of computational experiments on well-known benchmark instances of the VRPTW which can be defined as follows. Consider the network $G = (N, A)$ described in Section 5.2.1, where N is the set of nodes, A is the set of arcs, and N^+ is the set of customer nodes. Each customer $i \in N^+$ is associated with a demand q_i, a service time s_i, and a time window $[e_i, l_i]$ within which the service must start. Moreover, we set $q_0 = s_0 = e_0 = q_{n+1} = s_{n+1} = e_{n+1} = 0$ and $l_0 = l_{n+1} = H$, where H is the planning horizon duration. Each arc $(i,j) \in A$ is associated with a routing cost c_{ij} and a traveling time t_{ij}. An unlimited fleet of homogeneous vehicles with capacity Q is available at a single depot. The VRPTW consists of finding feasible routes visiting all customers $i \in N^+$ exactly once such that the total routing cost is minimized. A route is deemed feasible if: it is elementary; all visited customers are served within their time windows; the total demand of these customers does not exceed the vehicle capacity Q. Furthermore, the cost of a route is computed as the sum of the costs of the arcs traversed by the route.

For the VRPTW, the pricing subproblem involves a set R of two resources, namely, time and load. At a node $i \in N$, the resource windows $[e_i^r, l_i^r]$, $r \in R$, are defined by their time window

$[e_i, l_i]$ and by the load window $[0, Q]$. Along an arc $(i, j) \in A$, the resource consumptions γ_{ij}^r, $r \in R$, are given by $t_{ij} + s_i$ for the time resource and by q_j for the load resource.

In this section, we provide some details on how we conducted our computational experiments and report the results obtained. The performance of the selective *arc-ng*-SPPRC pricing is compared against the default *arc-ng*-SPPRC pricing. Both of these settings are implemented over the same column-and-cut generation framework, which is described in Section 5.4.1. In Section 5.4.2, we provide details on the experiments design. Finally, in Section 5.4.3, we report and discuss the computational results obtained.

5.4.1 Column-and-cut-generation framework

In our experiments, we consider a column-and-cut-generation framework very similar to the one developed by Pecin et al. [46] for the VRPTW. The algorithm includes several refinements that enhance its performance. The labeling algorithm used to solve the pricing subproblem relies on a bidirectional DSSR procedure (without completion bounds [45]), having the time as the critical resource [98]. The maximum size of *ng*-sets in the algorithm vary according to the instance being solved as discussed below. Furthermore, three fast heuristic labeling algorithms are always executed before any execution of the exact pricing algorithm. The first heuristic keeps only the least-cost label associated with a given time arriving at each customer node. The second and third heuristics rely on a reduced version of the network G, where around 7 and 12 (for the second and third, respectively) arcs entering and leaving each customer node are kept (see [44] for details on how these arcs are selected). At each column generation iteration, these three heuristic labeling algorithms are called in sequence until one of them finds negative reduced cost columns. If they all fail, the cut separation routines (see below) are invoked. When cuts are found, they are added to the RMP and column generation is re-started. Otherwise, the exact pricing algorithm is called. If it finds negative reduced cost columns, the process starts over again. Otherwise, the algorithm stops.

Because we wanted to make the implementation of our column generation algorithm as simple as possible, we decided not to consider the dynamic mechanism described by Bulhões et al. to increase and reduce the size of the *ng*-sets when needed. We judged that this procedure would make the implementation of the algorithm very cumbersome. Hence, we decided to rely on Martinelli et al. [97]'s framework to handle the size of the *ng*-sets. In our implementation, *ng*-sets are initially empty and are updated throughout the solution process. Two strategies were considered to update the arc *ng*-sets: i) `only-cycle`, where only *ng*-sets associated with arcs appearing in an infeasible cycle are increased; and ii) `all-arcs`, where all the arcs induced by the customer nodes in an infeasible cycle have their *ng*-sets increased. The strategy

`only-cycle` is the same considered by Bulhões et al. [1]. As pointed out by the authors, and also as we have observed in our preliminary experiments, the `only-cycle` strategy produces a higher number of DSSR iterations. However, the computational complexity of each such iteration remains lower (this is more notorious towards the end).

The baseline algorithm also includes the separation of several cuts, namely: rounded capacity cuts (RCCs, [76]), limited-memory rank-1 cuts Pecin et al. [45, 46], and elementary cuts [46]. Except for the RCCs, all the other cuts are non-robust, i.e., their dual variables cannot be considered directly in the modified arc costs when solving the pricing subproblem. The explicit handling of these dual variables increases the difficulty of solving it. Consequently, the solution process is divided in two phases, called the *robust* and *non-robust* phases. In the robust phase, the search for violated cuts is restricted to the RCCs. This phase ends with a call to the exact pricing algorithm that yielded no new columns and the application of variable arc fixing [130]. The non-robust phase then starts by searching for non-robust violated cuts. If some are found, the solution process is re-started. This phase also ends with a final unsuccessful call to the exact pricing algorithm, ensuring that the value of the current RMP optimal solution provides a valid lower bound.

5.4.2 Experiments design

Our tests were performed on two datasets: 1) the 14 hardest 100-customer VRPTW instances from Solomon [214]; and 2) the 200-customer instances of Gehring and Homberger [215] that are reported to be solved in less than 5 hours by Sadykov et al. [216]. These instances are called hereafter the S and the GH instances. Both datasets contain instances whose node locations are chosen at random, clustered, and mixed (random and clustered). This is denoted in the S and GH instances, respectively, as: S-C, S-R, and S-RC; and GH-C1, GH-C2, GH-R1, GH-R2, GH-RC1 and GH-RC2. For the GH instances, specifically, instances in sets GH-C1, GH-R1, and GH-RC1 have tight time windows and vehicle capacity, and typically admit optimal solutions with short vehicle routes. On the other hand, sets GH-C2, GH-R2, and GH-RC2 have large time windows and loose vehicle capacity and, typically, admit optimal solutions with fewer but longer vehicle routes.

Similarly to what is done in Pecin et al. [46], for all S instances except S-R208, and all GH-C1, GH-RC1, and GH-R1 instances, the cardinality of *ng*-sets $\mathcal{N}_i$, $i \in N^+$ are limited to 10. For the other instances, where allowing more cycles can be very harmful to the lower bound, we consider $|\mathcal{N}_i| = 20$. These node *ng*-sets are used to generate the arc *ng*-sets $\mathcal{N}_a$, $a = (u, v) \in A$, by using the formula $\mathcal{N}_a \leftarrow \mathcal{N}_u \cap \mathcal{N}_v$. Furthermore, for all instances except the GH-C1, GH-RC1, and GH-R1 instances, we observed that the capacity constraint is not

binding. Therefore, to alleviate the solution of the pricing subproblem, we did not consider this constraint, which is later enforced using RCCs whenever needed.

Given that our goal is not to solve to optimality the considered instances but rather to illustrate the impact of applying the selective *arc-ng*-SPPRC pricing, we focus only on the root node results. The algorithm described in Section 5.4.1 was coded in C++, and IBM ILOG CPLEX Optimizer 12.8 was used as the LP solver. The experiments were carried out on an Intel Xeon ES-2637 3.5GHz with 128GB RAM, running Linux Oracle Server 7.6. A time limit of 24h was set to solve all instances.

In our experiments, we compare the performance of the default *arc-ng*-SPPRC pricing against three versions of the selective *arc-ng*-SPPRC pricing, which differ by the dominance rule employed. All settings considered in our study are detailed below:

1. `Default`: The baseline algorithm that considers relations (5.10)–(5.13) as dominance rule. Relation (5.13) is reinforced by including the set $\mathcal{U}(L')$ of unreachable nodes in its right-hand side (which becomes $\Pi(L') \cup \mathcal{U}(L')$).
2. `SetBased`: Considers relations (5.10)–(5.12) and the set-based relation (5.21) as dominance rule.
3. `Pairwise`: Considers relations (5.10)–(5.12) and the pairwise relation (5.19) as dominance rule.
4. `SetPair`: Considers relations (5.10)–(5.12), both pairwise (5.19) and set-based (5.21) relations as dominance rule.

Additionally, each pricing setting is executed twice, each time considering a different DSSR strategy, i.e., `only-cycle` or `all-arcs`. Hence, in total, we test eight distinct configurations.

Notice that the `Default` setting applies a pairwise comparison on sets Π. We do not consider in our analysis the set-based dominance rule of the default *arc-ng*-SPPRC pricing developed by Bulhões et al. [1] because it would have required major changes to our implementation. Because our set-based dominance rule encompasses that of Bulhões et al., it is expected to yield better results.

For each instance, we retrieve the following information: lower bound (**lb**), number of column generation iterations performed by each algorithm (**#iters**), CPU time in seconds (**T(s)**), average number of DSSR iterations per column generation iteration performed by each algorithm (**#DSSR**), and average number of labels generated per iteration (**#labels**). This information is collected before (robust phase) and after (non-robust phase) the addition of non-robust cuts. Yet, because the main indicators for evaluating the efficiency of the selective algorithm are computational time and average number of labels generated per iteration, in

Section 5.4.3, we focus our analysis on these two measures. Detailed results for all indicators are presented in Appendix A.

5.4.3 Computational results

In this section, we discuss the results obtained during our experiments. For the sake of conciseness, we only report summary results (see Appendix A for complete results). In Tables 5.1–5.4, we provide the average and the median for **T(s)** and **#labels** per instance class. In each table, we compare the performance of settings `Default`, `SetBased`, `Pairwise`, and `SetPair`. We analyze separately the performance of the algorithms with respect to the DSSR strategy employed (`only-cycle` and `all-arcs`). We provide distinct tables for the algorithms before and after separating non-robust cuts. Finally, line **#Best** shows the number of times that each setting provides the best results for each indicator and line **#OPD** displays the number of times that the corresponding setting outperforms the `Default` setting. Notice that, in the case where both algorithms produce equal results for a given instance, this instance is counted for both algorithms. As a consequence, the sum of the **#Best** values can exceed the number of instances.

In Tables 5.1 and 5.2, we present the summary results produced by the algorithms before the separation of non-robust cuts. These results show a clear advantage of the selective settings over the default algorithm in terms of running time and the number of labels produced. This statement is supported by the **#OPD** values. Settings `SetPair` and `SetBased` are the ones yielding the best gains in terms of running time for `only-cycle` and `all-arcs`, respectively. Regarding the number of labels, for both DSSR strategies, the setting `Pairwise` is the most effective at reducing the number of non-dominated labels kept for both DSSR strategies. Moreover, as one can see from detailed results in Appendix A, during the robust phase of the algorithm, the selective algorithms allow better bounds to be achieved. This observation is more notable when solving GH instances using the `only-cycle` DSSR strategy. The selective strategies also tend to require less DSSR iterations.

When comparing the performance of the selective algorithms employing different DSSR strategies, the results confirm what has already been hinted by Bulhões et al. [1], i.e., the algorithms employing `only-cycle` produce a larger number of DSSR iterations. Also, they tend to keep more non-dominated labels. On the other hand, these algorithms seem to require *ng*-sets of smaller sizes.

Tables 5.3 and 5.4 show the results obtained during the non-robust phase of the algorithms. In the presence of non-robust cuts, the selective algorithms can become more time-consuming. Nevertheless, they are still faster than the default algorithm for the majority of the instances.

Table 5.1 Aggregated results using the `only-cycle` DSSR strategy before adding non-robust cuts

Classes		**T(s)**				**#labels**			
		`Default`	`SetBased`	`Pairwise`	`SetPair`	`Default`	`SetBased`	`Pairwise`	`SetPair`
S-C	Average	14,830	8,723	10,232	9,247	44,428	38,831	39,232	43,842
	Median	14,830	8,723	10,232	9,247	44,428	38,831	39,232	43,842
S-R	Average	1,581	1,254	1,168	1,230	76,292	56,785	57,885	61,668
	Median	641	551	531	515	38,749	32,613	33,084	37,482
S-RC	Average	7,015	4,733	4,517	4,894	502,004	343,281	319,508	328,526
	Median	3,071	2,473	2,425	2,476	632,136	463,053	425,550	443,300
GH-C1	Average	604	525	541	494	319,713	258,680	246,711	261,399
	Median	566	503	492	479	296,760	236,845	237,015	245,659
GH-C2	Average	6,136	5,386	4,763	4,826	21,571	20,257	17,651	18,199
	Median	6,342	6,322	5,605	5,688	15,499	13,644	14,472	12,457
GH-R1	Average	175	142	175	147	209,212	187,153	196,498	183,037
	Median	153	134	157	135	190,129	170,515	191,588	179,043
GH-R2	Average	4,789	4,342	4,399	4,250	13,845	11,459	11,717	12,536
	Median	4,217	3,836	3,639	3,938	11,431	9,902	10,221	9,247
GH-RC1	Average	201	167	192	174	438,176	369,259	390,951	379,880
	Median	144	128	137	129	253,216	218,390	228,745	214,626
GH-RC2	Average	6,689	5,757	5,887	5,816	46,391	36,427	40,806	41,672
	Median	6,621	5,665	5,793	5,739	45,338	36,552	36,654	36,959
#Best		2/45	11/45	15/45	17/45	3/45	19/45	20/45	9/45
#OPD		–	42/45	38/45	42/45	–	38/45	42/45	40/45

Table 5.2 Aggregated results using the `all-arcs` DSSR strategy before adding non-robust cuts

Classes		**T(s)**				**#labels**			
		`Default`	`SetBased`	`Pairwise`	`SetPair`	`Default`	`SetBased`	`Pairwise`	`SetPair`
S-C	Average	5,143	11,198	4,897	8,214	36,339	29,880	30,321	29,100
	Median	5,143	11,198	4,897	8,214	36,339	29,880	30,321	29,100
S-R	Average	815	749	747	773	46,854	41,961	43,092	42,001
	Median	390	365	348	375	24,038	25,711	21,949	24,779
S-RC	Average	1,812	1,526	1,496	1,452	163,293	135,035	136,881	121,865
	Median	904	786	826	814	218,110	163,647	178,990	163,170
GH-C1	Average	378	323	326	329	170,063	164,153	148,659	160,393
	Median	400	317	342	329	147,946	127,657	115,431	126,360
GH-C2	Average	3,156	3,068	3,030	2,891	13,658	14,850	12,871	13,136
	Median	3,564	3,412	3,405	3,144	11,771	13,276	12,090	11,894
GH-R1	Average	139	119	136	121	133,901	128,970	118,771	122,280
	Median	132	123	128	121	117,367	124,032	122,804	106,912
GH-R2	Average	3,738	3,436	3,508	3,467	9,810	10,231	8,693	8,666
	Median	3,616	3,471	3,339	3,352	7,647	8,488	8,044	7,994
GH-RC1	Average	131	112	128	116	225,358	217,014	213,540	222,278
	Median	116	102	117	108	170,003	164,044	154,841	171,926
GH-RC2	Average	3,959	3,777	3,580	3,653	25,770	25,056	23,105	25,354
	Median	3,411	3,372	3,314	3,289	25,309	23,846	23,324	22,316
#Best		4/45	16/45	13/45	12/45	11/45	11/45	18/45	12/45
#OPD		–	36/45	34/45	38/45	–	23/45	37/45	32/45

The better performance of the selective algorithms is confirmed when we analyze the performance of the algorithms pairwise in Appendix A. The somewhat high average times associated with some selective settings are due to the poor performance of the selective algorithms when solving particular instances. However, when analyzing a less biased indicator such as the median, the performance of the algorithms tend to be more similar. Sometimes, the selective algorithms can even yield better results, as it is the case for the setting `SetBased` when solving instances from classes GH-R2 and GH-RC2 in Table 5.3. Regarding the number of labels, the selective algorithms remain more effective in reducing the number of generated labels in each algorithm.

Table 5.3 Aggregated results using the `only-cycle` DSSR strategy after adding non-robust cuts

Classes		**T(s)**				**#labels**			
		`Default`	`SetBased`	`Pairwise`	`SetPair`	`Default`	`SetBased`	`Pairwise`	`SetPair`
S-C	Average	14,831	8,723	10,232	9,248	44,428	38,831	39,232	43,842
	Median	14,831	8,723	10,232	9,248	44,428	38,831	39,232	43,842
S-R	Average	24,607	28,713	23,606	25,376	161,918	161,095	151,546	134,202
	Median	3,034	2,219	2,682	2,374	30,501	29,865	30,571	27,684
S-RC	Average	8,342	6,231	5,926	6,442	265,520	188,351	175,184	182,177
	Median	4,626	4,107	3,940	4,172	347,683	249,233	233,377	244,941
GH-C1	Average	31,590	10,321	14,917	60,826	211,779	186,186	173,657	181,226
	Median	1,090	967	1,006	911	209,024	188,646	168,723	179,803
GH-C2	Average	8,575	7,606	7,172	7,114	14,852	13,912	12,893	12,495
	Median	7,449	7,748	6,624	7,322	8,996	9,145	7,490	7,256
GH-R1	Average	6,769	4,550	5,270	8,425	162,425	155,355	154,977	155,259
	Median	2,041	2,014	2,377	2,497	86,138	81,608	79,223	82,495
GH-R2	Average	7,356	7,563	7,343	6,994	5,939	5,014	5,208	5,153
	Median	5,808	5,294	5,232	5,487	5,578	4,926	4,638	4,622
GH-RC1	Average	39,091	37,398	43,179	50,820	292,527	297,061	302,914	285,138
	Median	15,333	19,823	17,230	33,158	114,444	113,465	115,899	113,536
RC2	Average	19,399	20,420	19,846	23,343	15,501	13,658	13,924	12,968
	Median	13,046	11,836	9,761	11,356	14,820	13,225	12,231	11,468
#Best		6/45	18/45	14/45	7/45	4/45	12/45	14/45	18/45
#OPD		–	33/45	32/45	23/45	–	34/45	31/45	39/45

Finally, we present time profiles Dolan and More [217] to compare the performance of the algorithms, which are generated as follows. Let $\mathcal{A}$ be a set of algorithms, namely, $\mathcal{A} = \{$`Default`, `SetBased`, `Pairwise`, `SetPair`$\}$ in our case. Let I be a set of instances, namely, the 45 instances considered in our experiments. For each algorithm $A \in \mathcal{A}$ and each instance $i \in I$, denote by T_A^i the time spent by A to solve instance i and let $T_{\text{Best}}^i = \min_{A \in \mathcal{A}}\{T_A^i\}$ be the best time achieved by an algorithm in $\mathcal{A}$ to solve this instance. For an algorithm $A \in \mathcal{A}$, a time profile is a function $\rho_A(\tau)$ of a ratio $\tau \geq 1$ that is equal to the percentage of instances such that $\frac{T_A^i}{T_{\text{Best}}^i} \leq \tau$, i.e.,

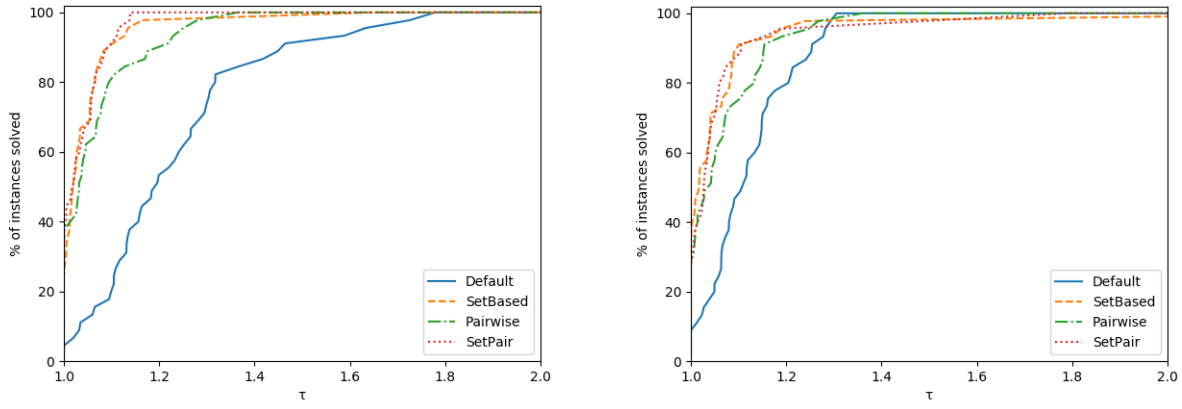
$$\rho_A(\tau) = 100\,\frac{|\{i \in I \mid \frac{T_A^i}{T_{\text{Best}}^i} \leq \tau\}|}{|I|}. \tag{5.22}$$

In particular, $\rho_A(1)$ represents the percentage of instances for which algorithm A was the

Table 5.4 Aggregated results using the `all-arcs` DSSR strategy after adding non-robust cuts

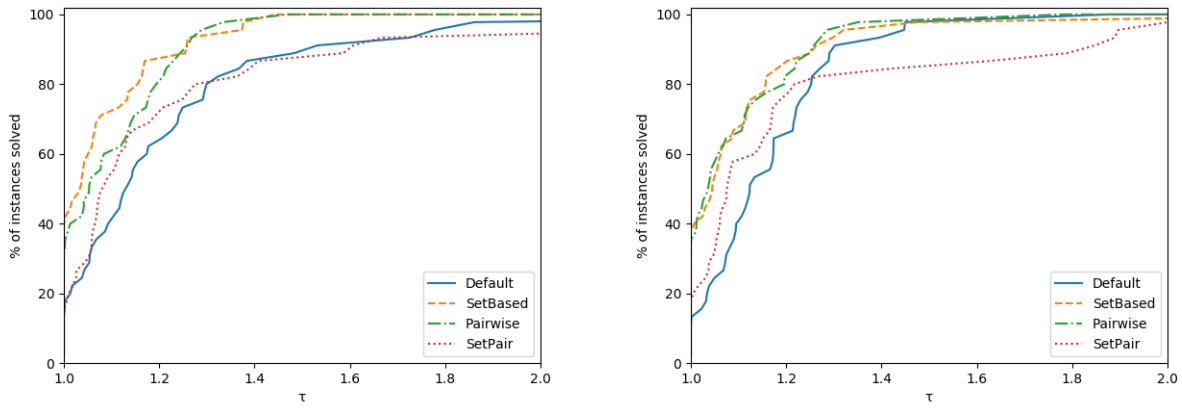
Classes		T(s)				#labels			
		`Default`	`SetBased`	`Pairwise`	`SetPair`	`Default`	`SetBased`	`Pairwise`	`SetPair`
S-C	Average	5,144	11,199	4,897	8,215	36,339	29,880	30,321	29,100
	Median	5,144	11,199	4,897	8,215	36,339	29,880	30,321	29,100
S-R	Average	24,899	22,678	23,555	24,346	116,572	113,544	120,109	103,872
	Median	1,784	1,631	2,035	1,739	21,936	27,285	25,382	28,997
S-RC	Average	2,847	2,554	2,515	2,613	93,557	78,568	77,082	73,720
	Median	1,961	1,746	1,766	1,852	128,015	95,283	96,170	90,537
GH-C1	Average	8,893	11,382	11,834	24,217	23,675	23,930	23,447	23,065
	Median	670	573	576	572	25,871	26,420	24,454	25,218
GH-C2	Average	5,730	4,790	4,498	4,643	4,444	4,497	4,072	4,135
	Median	4,633	4,390	4,494	4,234	4,033	4,423	3,986	3,850
GH-R1	Average	4,447	3,942	4,402	6,188	51,827	52,685	51,153	52,042
	Median	2,140	2,074	1,940	1,923	29,849	31,203	29,589	29,301
GH-R2	Average	6,032	5,883	5,855	6,247	2,782	2,903	2,767	2,742
	Median	4,994	4,842	4,653	4,747	2,213	2,287	2,287	2,233
GH-RC1	Average	23,394	18,850	27,210	27,301	55,442	58,633	58,564	55,439
	Median	16,704	14,473	13,318	25,329	39,128	40,898	38,055	40,736
GH-RC2	Average	12,091	11,194	10,633	10,593	7,473	8,120	6,662	7,342
	Median	7,410	7,946	7,370	7,212	7,422	7,932	6,863	7,360
#Best		5/45	17/45	15/45	8/45	9/45	15/45	11/45	13/45
#OPD		–	37/45	30/45	29/45	–	25/45	33/45	30/45

fastest. Figures 5.5 and 5.6 display the time profiles of the four algorithms for the robust and the non-robust phases, respectively. From these profiles, one can see that the selective algorithms yield considerable speed-ups when compared to the `Default` algorithm. The acceleration is more notable when comparing algorithms employing the `only-cycle` DSSR strategy. Additionally, we remark that setting `SetPair` is the fastest during the robust phase, but becomes time consuming once the separation of non-robust cuts starts.



Using `only-cycle` DSSR Using `all-arcs` DSSR

Figure 5.5 Time profiles for all instances before adding non-robust cuts



Using `only-cycle` DSSR Using `all-arcs` DSSR

Figure 5.6 Time profiles for all instances after adding non-robust cuts

5.5 Concluding remarks

In this paper, we have presented two new selective dominance rules for the *arc-ng*-SPPRC, namely, one that applies pairwise label comparison and one that extends the set-based rule of [1]. The latter rule is stronger than the one proposed by Bulhões et al. [1] as it considers the selective mechanism expressed by condition (5.18), which increases the chances that a label L dominates a label $L' = L^\star \oplus w$. Furthermore, an additional speedup may be observed if the average memory size is much larger than the number of cost and resource components in a label. Our computational experiments on VRPTW benchmark instances showed that, in general, the new mechanism allows a reduction in terms of the number of treated labels and, consequently, of the computational time. Note also that one of the main interests for using this mechanism is that it does not rely on complex data structures like the ones described in [1] (Section 4.2) and [107] (Section 3.2) to work satisfactorily.

An extension of our idea would be to investigate the impact of generalizing relation (5.14) to consider other predecessors further down in the path. Theoretically, this generalization would strengthen even more the dominance rule. However, it seems that an elaborated data structure would be required to address the overhead incurred by the new relations.

Acknowledgments

This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada [Grant 2017-05683]. This support is gratefully acknowledged.

CHAPTER 6 STABILIZED COLUMN GENERATION VIA AGGREGATED ROWS SEPARATION

6.1 Introduction

In the last years, CG has been the leading technique to cope with large-scale problems arising from several contexts, namely: vehicle scheduling and routing problems [8, 9], crew scheduling problems [218], cutting and packing problems [10], computer vision problems [12], among others. Due to the underlying structure of the problems above, Dantzig-Wolfe (DW) decomposition [23] allows modeling them as set-partitioning (covering / packing) formulations, in which their variables often encode combinatorial objects as paths, sets, or permutations [15].

Without loss of generality, by applying DW decomposition, a generic set-covering formulation for the problems listed above can be expressed as follows:

$$\min \sum_{j \in \Omega} c_j \theta_j \tag{6.1}$$

$$\text{subject to:} \sum_{j \in \Omega} \mathbf{a_j} \theta_j \geq \mathbf{b} \tag{6.2}$$

$$\theta_j \in \mathbb{Z}^+ \qquad \forall j \in \Omega, \tag{6.3}$$

where Ω is a finite set of indices, scalar c_j, and vectors $\mathbf{a_j}$ and $\mathbf{b}$ are associated with the problem structure, and θ_j are decision variables. Notice that, in (6.2), the sign $\geq$ might be replaced by $=$ or $\leq$ for the set-partitioning and set-packing, respectively. Formulations defined as (6.1)–(6.3) are likely to present an exponential number of variables, which makes it prohibitive to handle them all at once. Hence, CG arises as the suitable technique to overcome this inconvenient.

Despite its efficiency in dealing with problems containing a large number of variables, CG may suffer from convergence issues due, mainly, to the degenerate structure of the RMP and the instability associated with its dual variables [15, 50]. As discussed in Chapter 2 (Section 2.3), several techniques have been proposed to tackle degeneracy and the instability of the dual variables. Some attempts have been made in trying to overcome or even exploit the degeneracy associated with CG algorithms [49, 52, 54]. However, the implementation of these methods may not be straightforward, as they may require changes in the CG process [49], or yield harder pricing subproblems (PSs) [52]. Moreover, these techniques are limited to solving set partitioning problems.

Regarding dual oscillation, prior knowledge about the domain of the dual variables may be used to derive valid inequalities for restraining the dual feasible space [66–68]. A restricted dual space helps the convergence of CG algorithms because it may reduce the number of dual optimal solutions potentially yielding attractive columns to be added to the RMP. Moreover, it may break the degenerate structure of the RMP, as fewer dual optimal solutions may be associated with the degenerate extreme points [66]. Typically, this practice is problem-dependent and relies a lot on problem-specific knowledge, which limits its use to the problem at hand or problems with a very similar structure [12, 66–70]. More recent studies, however, have started to present more general approaches in this direction [71, 72].

Alternatively, it is possible to modify the implementation of a standard CG algorithm and consider center-based methods [58, 60, 63] or interior-point methods [64, 65] to alleviate the impact of bad quality dual solutions on the CG process and accelerate the algorithm convergence. These methods, however, may present some drawbacks such as making the pricing process harder [63, 64], increasing the size of the RMP [60], and requiring several parameters to be tuned [60].

In this chapter, we present a new stabilization framework that relies on the dynamic separation of aggregated rows for the CG RMP. The aggregation of the constraints relies on *neighborhoods* that are defined in terms of similarities between the items to be covered in the RMP. Unlike other problems based on the aggregation of constraints in the RMP [49, 66], our method is very generic. It can be used to solve problems formulated as set-partitioning, covering, or packing formulations. As a matter of fact, the applications considered in our experiments arise from different contexts and differ considerably in terms of objective function and PS. Furthermore, no changes in the structure of the problem are required, which makes the implementation of the method very straightforward. Our computational experiments show that the new stabilization scheme allows gains in terms of the number of CG iterations performed and total running time when compared to a standard column generation algorithm.

The remainder of this chapter is structured as follows: In Section 6.2, we present our stabilization framework and discuss how it relates to other stabilization methods. In Section 6.3, we present our computational experiments. For the sake of clarity, each one of the applications that we solve is presented in separate subsections. Each one of the sections contains a short definition of the problem being solved, a description of the instances considered in the experiments, and the numerical results obtained. Finally, some conclusions and insights are drawn in Section 6.4.

6.2 The dynamic aggregated-rows separation method

In this section, we provide a general description of the stabilization method based on a dynamic separation of aggregated rows (`dyn`–SAR). For the sake of simplicity, we describe the `dyn`-SAR algorithm using a set-covering formulation. However, as it will become clear throughout this section, extending the method to address set-packing or set-partitioning problems is straightforward. To make this section self-contained, we re-visit the formulation (6.1)–(6.3), and we take the opportunity to define some concepts and terms that will be useful when describing the method. The `dyn`-SAR algorithm implicitly relies on an extended version of the RMP, which contains as many variables as the original RMP, and an exponential number of constraints. The new formulation is presented in Section 6.2.1 and the new method is described in Section 6.2.2.

6.2.1 Problem description

In this section, we provide a formal description of the set-covering problem and present the RMP arising from this problem. Additionally, we derive the new extended RMP over which our method implicitly depends.

Set-covering problem

Let N be a set of items to be covered. Let Ω be the set of all feasible patterns containing items $i \in N$. A binary parameter a_j^i specifies whether an item i belongs to a pattern j. For each pattern $j \in \Omega$, a binary variable θ_j takes value 1 if a pattern j is chosen to be in the solution or 0, otherwise. Finally, c_j corresponds to the cost incurred from selecting a pattern j in the solution. The set-covering problem can be modeled as follows:

$$\min \sum_{j \in \Omega} c_j \theta_j \tag{6.4}$$

$$\text{suject to:} \sum_{j \in \Omega} a_j^i \theta_j \geq 1 \qquad \forall i \in N \tag{6.5}$$

$$\theta_j \in \{0, 1\} \qquad \forall j \in \Omega. \tag{6.6}$$

The objective function (6.4) aims at minimizing the total cost incurred from covering all the items. Constraints (6.5) impose that every item will be covered by at least one pattern. Finally, constraints (6.6) define the domain of the variables.

Restricted master problem of column generation

As stated in Section 6.1, CG is the suitable technique to tackle extended formulations like (6.4)–(6.6). From (6.4)–(6.6), the RMP of the CG algorithm is obtained by considering only a subset $\Omega' \subseteq \Omega$ of patterns, and by dropping the integrality requirements. The remaining parameters assume the same meaning as described in the previous section.

$$\min \sum_{j \in \Omega'} c_j \theta_j \tag{6.7}$$

$$\text{suject to:} \sum_{j \in \Omega'} a_j^i \theta_j \geq 1 \qquad \forall i \in N \tag{6.8}$$

$$\theta_j \geq 0 \qquad \forall j \in \Omega'. \tag{6.9}$$

If we consider the complete set Ω instead of just the set Ω', the master problem (MP) of the CG is defined.

Extended restricted master problem

We introduce the extended RMP (ERMP), which contains a constraint for each subset $\varnothing \subsetneq S \subseteq N$. In practice, constraints (6.11) in ERMP are obtained by simply summing up constraints (6.8) from RMP corresponding to each item $i \in S$. Therefore, the coefficient a_j^S associated with aggregated constraints are computed as $a_j^S = \sum_{i \in S} a_j^i$, and indicates how many times items in S are covered by a pattern j. The ERMP is expressed as follows:

$$\min \sum_{j \in \Omega'} c_j \theta_j \tag{6.10}$$

$$\text{suject to:} \sum_{j \in \Omega'} a_j^S \theta_j \geq |S| \qquad \forall S \subseteq N, |S| > 0 \tag{6.11}$$

$$\theta_j \geq 0 \qquad \forall j \in \Omega', \tag{6.12}$$

The remaining parameters and variables have the same meaning as described previously in this section. Similarly to what is done for the RMP, if we replace set Ω by set Ω' in the formulation, the extended MP (EMP) is obtained. Notice that all constraints in (6.8) are also in (6.11). Hence, all constraints for $|S| \geq 2$ are redundant with the ones for $|S| = 1$. Since columns in RMP and ERMP are the same, formulations (6.7)–(6.9) and (6.10)–(6.12) are equivalent.

Because the ERMP is obtained directly from (6.7)–(6.9), it also has an exponential number of variables. At the same time, it contains an exponentially large number of constraints. As a

consequence, employing a standard CG method, in which all the constraints of the RMP are known in advance, and handled at once, may not be doable because of the formulation size. To overcome this issue, we develop a new method that not only generates variables dynamically but also separates constraints when needed. Notice that our method does not characterize a typical column-and-constraint generation algorithm as the ones embedded in branch-price-and-cut methods. The constraints generated throughout the method are necessary to ensure the feasibility of the problem instead of reinforcing the formulation.

6.2.2 The `dyn`-SAR method

This section is divided into two parts. First, we provide some theoretical foundations to support the use of the `dyn`-SAR method. Second, we describe `dyn`-SAR algorithmically.

Foundations

The `dyn`-SAR is an iterative method that uses CG to solve to optimality a sequence of relaxations $\mathcal{R}^1, \mathcal{R}^2, \ldots, \mathcal{R}^k$ arising from EMP, where k is the number of problems solved. $\mathcal{R}^1, \mathcal{R}^2, \ldots, \mathcal{R}^k$ are defined when new constraints of type (6.11) are sequentially incorporated into the formulation throughout the method. Even if singleton sets S (sets with a single element) are among the sets, as it will be described later in this section, the method starts with constraints corresponding to much larger sets S. Despite being applied to a formulation containing aggregated constraints (obtained by summing up constraints of type (6.8)), `dyn`-SAR relies on the original RMP to identify violated constraints to be added to the relaxations of ERMP. When a relaxation $\mathcal{R}$ is solved, one verifies if the solution found is also feasible for the original RMP, i.e., if all the items $i \in N$ are covered. Otherwise, aggregated constraints involving non-covered items are generated and incorporated into the current problem. These relaxations are such that $\underline{z}(\mathcal{R}^1) \leq \underline{z}(\mathcal{R}^2) \leq \ldots \leq \underline{z}(\mathcal{R}^k)$, where $\underline{z}$ indicates the lower bound attained once a CG algorithm has been applied to solve the problem to optimality.

Contrarily to other methods relying on a modified RMP, our method does not require the problem to have any specific structure [49, 67], which allows it to be applied to a variety of problems. Moreover, it does not rely on a partition of the RMP constraints, which may entail more complicated PSs [52]. Despite relying on a simultaneous generation of variables and constraints, our method is different from the ones described in [55] and [56], where only partial dual information is available. In our case, due to the reformulation (6.10)–(6.12), all the aggregated constraints are (implicitly) defined. Therefore, the dual information coming from constraints (6.11) is complete and may be easily translated into the dual variables associated with the constraints (6.8) in the original RMP, without the need of being estimated [55] or

projected [56]. Therefore, the same PS arising from (6.7)–(6.9) may also be considered when tackling (6.10)–(6.12). Finally, some similarities between our method and the use of surrogate relaxations [219, 220] may be found. However, in the literature, surrogate constraints are typically employed to provide an approximation of the original problem.

For each subset S, let $\mathscr{C}(S)$ denote a constraint of type (6.11). Explicitly, we denote:

$$\mathscr{C}(S) \qquad \sum_{j \in \Omega'} a_j^S \theta_j \geq |S|, \tag{6.13}$$

where, as described in Section 6.2.1, a_j^S is an integer indicating the number of times the items in S are covered by the pattern j. Let γ_S be the dual variable associated with a constraint (6.13), and let $\pi_i,\ \forall i \in N$, be the dual variables associated with constraints (6.8). A relation between the two variables γ and π is given by:

$$\pi_i = \sum_{S \subseteq N} b_i^S \gamma_S \qquad \forall i \in N, \tag{6.14}$$

where b_i^S is a binary parameter indicating if item i belongs to set S. Note that while relation (6.14) is valid way to compute the dual vector π, it might not be the only valid assignment of dual variables.

Therefore, when solving the PS, the reduced cost of a variable $j \in \Omega$ may be computed as follows:

$$\bar{c}_j = c_j - \sum_{i \in N} a_j^i \pi_i, \tag{6.15}$$

The stabilization effect induced by `dyn`-SAR appears at two levels. First, the idea of considering only a few aggregated constraints when starting the method helps to alleviate the degeneracy. Because coefficients in constraints (6.11) are associated with the sets S rather than with individual items, the number of non-zeros in the columns tends to be reduced. Yet, working with the ERMP does not necessarily imply that a more compact version of the RMP will be generated as in [49]. As the separation of the constraints happens, it is not guaranteed that the size of the working basis will remain smaller than that of the original RMP throughout the whole process. In fact, we have observed that the number of constraints in the ERMP when the method finishes is consistently larger than $|N|$. Second, the relationship between the dual variables $\pi_i,\ i \in N$, considered in the pricing algorithm is much stronger than the one obtained via relation (6.14). When they are obtained directly from formulation (6.8), dual variables are more susceptible to move freely around the dual space. In the case of

relation (6.14), dual variables $\pi_i,\ i \in N$, are inter-related and better distributed, and hence extreme dual values are less likely to appear.

6.2.3 Description of the method

The `dyn`-SAR method is very flexible and its validity does not depend on any specific structure of the problem at hand. Yet, the method exploits the underlying structure of the problem to guide the construction of sets S in an attempt to speed-up the convergence of the method. In this context, we introduce the notion of *neighborhood* of a given item i, which we denote $\mathcal{H}(i)$. When defining the neighborhoods of an item $i \in N$, we regroup items that share some similarities among them. The measure of similarity varies according to the problem being solved. For instance, it could be the geographic location, the weight, or any other attribute associated with items $i \in N$.

A general representation of `dyn`-SAR is given in Algorithm 2. The `dyn`-SAR algorithm starts by solving a relaxation $\mathcal{R}^1$ that consists of the objective function (6.10) and constraints $\mathscr{C}(S_h),\ h = \{1, \ldots, \ell\}$, where $\bigcup_{h=1}^{\ell} S_h = N$ and $\bigcap_{h=1}^{\ell} S_h = \varnothing$ (Line 1). This grouping is defined in terms of some criterion arising naturally from the problem structure. In Section 6.3, we explain how this partition is determined for each one of the problems solved in our experiments. Once $\mathcal{R}^1$ is solved, one verifies if the computed solution $\mathcal{X}^1$ is feasible for the associated RMP, otherwise one proceeds with the separation of new constraints $\mathscr{C}(S)$.

The process of constructing sets S is simple and straightforward. Initially, one starts by collecting all items in the current solution that are not fully covered (set-covering) by running the procedure `checkViolation`$(\mathcal{X}^1)$, i.e., we build set $U = \{i \in N : \texttt{viol}(\mathscr{C}(\{i\})) > 0\}$ (Line 5). Notice that, if a set-partitioning problem is solved, it would be necessary to look for items that are not fully covered or items that are over-covered. For a set-packing problem, in turn, violated items are those which are over-covered. Procedure `checkViolation`$(\mathcal{X})$ checks if any of the items in N is not fully covered. If yes, additional constraints may be generated to be added to the ERMP. The complexity of `checkViolation`$(\mathcal{X})$ is polynomial and given by: $O(|\Omega'_{>0}| \times |N|)$, where $\Omega'_{>0}$ is the set of basic variables and is such that $|\Omega'_{>0}| \leq |N|$.

Procedure `buildViolatedSets`(U) builds violated sets S by regrouping items in U. The generated sets are stored in a pool $\mathcal{V}$, which contains violated sets having the potential to be added to the RMP. Initially, the items in U are sorted in non-increasing order according to their violation. Then, for each $i \in U$, one creates a set S_i containing i and at most the `maxNbItemsPerSet` most violated items $u \in \mathcal{H}_i \cap U$. The parameter `maxNbItemsPerSet` imposes the maximum cardinality of sets S. When all potential sets S_i have been built, it can happen that $|\mathcal{V}|$ is larger than a given parameter `maxNbConstrsToAdd`. When this happens,

one calls the procedure `selectViolatedSets`($\mathcal{V}$) to select `maxNbConstrsToAdd` violated sets $S \subset \mathcal{V}$ by running a max-dispersion heuristic (Line 11). When only the most violated sets from $\mathcal{V}$ are considered, i.e., when the dispersion heuristic is not executed, it is very likely that the same items will be covered several times in the built sets. This would leave several items uncovered, which would require additional calls to the separation procedure. The newly generated constraints are then added to $\mathcal{R}^1$, defining a new relaxation $\mathcal{R}^2$ (Line 13). The process continues iteratively until a given stopping criterion is satisfied. Typically, the algorithm stops when all items in N are covered. However, one can also consider other criteria such as: if the lower bound is not improved by at least a certain value after a given number of iterations; if the number of constraints reaches a given threshold; if a given time limit is exceeded; etc. When some of the alternative criteria are considered, the method provides an approximate solution.

Algorithm 2: `dyn`-SAR

1: Initialize $\mathcal{R}^1$ with objective function (6.10) and constraints $\mathscr{C}(S_h)$, $h = \{1, \ldots, \ell\}$, where $\bigcup_{h=1}^{\ell} S_h = N$ and $\bigcap_{h=1}^{\ell} S_h = \varnothing$.
2: $k \leftarrow 1$
3: **while** Stopping criterion not satisfied **do**
4: Solve $\mathcal{R}^k$ using CG to obtain solution $\mathcal{X}^k$
5: $U \leftarrow$ `checkViolation`($\mathcal{X}^k$)
6: **if** $U = \varnothing$ **then**
7: **break**
8: **end if**
9: $\mathcal{V} \leftarrow$ `buildViolatedSets`(U)
10: **if** $|\mathcal{V}| >$ `maxNbConstrsToAdd` **then**
11: $\mathcal{V} \leftarrow$ `selectViolatedSets`($\mathcal{V}$)
12: **end if**
13: $\mathcal{R}^{k+1} = \mathcal{R}^k \cup \bigcup_{S \in \mathcal{V}} \mathscr{C}(S)$
14: $k \leftarrow k + 1$
15: **end while**

As already mentioned, one of the main advantages of using `dyn`-SAR to tackle extended formulations is its simplicity. No significant changes in terms of implementation on a standard CG algorithm are required. Furthermore, because the `dyn`-SAR method consists of solving a sequence of relaxations for the ERMP, and in turn for the RMP, it is possible to abort the execution of the method whenever the optimal solution for a relaxation $\mathcal{R}^1, \mathcal{R}^2, \ldots, \mathcal{R}^h$, $h = 1, \ldots, k$, is found. The value associated with this solution provides a valid lower bound for the original RMP. This aspect may be particularly useful when embedding `dyn`-SAR into

a branch-price-and-cut framework.

6.3 Computational experiments

In this section, we present the numerical experiments carried out on instances of the VRPTW, the multi-person pose estimation problem (MPPE), and the bin packing problem with conflicts (BPPC). Given that our goal is to illustrate the impact of applying the new stabilization framework, we focus only on the root node results. Each one of the following subsections contains: a brief definition of the problem being solved; a brief description of the algorithms implemented in our experiments; a detailing of the benchmark dataset considered; and a discussion of the numerical results obtained.

6.3.1 Vehicle routing problem with time windows

Because the VRPTW has already been largely discussed in this document, for the sake of conciseness, we refer to other chapters in this thesis for more details on the problem. The VRPTW is formally defined in Section 4.1.1. Here, however, the set of customers to be covered is denoted N. When applying CG, we model the VRPTW as described in Section 5.4. Here, however, we do not consider the PS as being an *arc-ng*-SPPRC, but rather a *ng*-SPPRC (Sections 4.3.1 and 5.2.2). For the experiments, we consider instances from Gehring and Homberger [215]'s benchmark dataset with 200 and 400 customers, which are described in Section 5.4.2.

Algorithm

The performance of the `dyn`-SAR method is compared against a standard CG (`std`-CG) algorithm with no explicit stabilization technique. The CG master problem of the latter, however, is formulated as a set-covering problem that restrains the feasible dual space [21]. For both algorithms, the RMP is initialized with singleton routes. Both algorithms `dyn`-SAR and `std`-CG employ the same pricing algorithm. In our experiments, valid inequalities are not employed to reinforce the RMP.

The PS is solved by means of a bidirectional labeling algorithm [98], having the time as the critical resource. Moreover, decremental state-space relaxation (DSSR, [41,42]) and *ng*-route relaxation, with neighborhoods formed by the ten closest customers [2], are employed to handle elementarity constraints. To accelerate the CG process, three heuristics are always executed before any execution of the exact pricing: the first consisting of a tabu search, and the second and the third are labeling algorithms solved over reduced networks containing at

most, respectively, around 5 and 10 arcs entering and exiting each customer node. All these heuristics are described in [44]. These heuristics are applied sequentially before any call to the exact labeling algorithm.

With respect to `dyn`-SAR, the method starts by sorting the customers in nondecreasing order according to their closeness to the depot. The customers are then separated into non-overlaping sets S, which are built, sequentially, by selecting the κ, the 2κ , the 4κ, ..., closest customers to the depot, every time increasing the value of κ by two. This procedure continues until all the customers in the instances are selected. Each set induces a constraint $\mathscr{C}(S)$ to be added to the RMP. The parameter κ is set to 20.

For each customer $i \in N$, we build neighborhoods containing the $\eta = |N|/4$ closest customers to i according to a *biased distance* $\hat{d}_{ij} = (d_{ij})^{\beta}$, where $\beta = \max\{1, \frac{d_{0i}}{d_{0j}}\}$. This correction in the value of the distance aims at favoring customers located geographically far from the depot to be covered earlier in the algorithm. During the separation of the constraints $\mathscr{C}(S)$, the cardinality of sets S is limited to the average size of the current basic routes. Additionally, the maximum number of constraints added to the RMP at each iteration of the `dyn`-SAR method is limited to 5% of $|N|$. When performing the separation of the constraints, we only apply the dispersion method to the $25\%|N|$ most violated sets S. The `dyn`-SAR algorithm is performed until all customers are fully covered, and no more columns with negative reduced cost can be found.

Experiments

Both algorithms (`std`-CG and `dyn`-SAR) were implemented using GENCOL library version 4.5, having IBM CPLEX Optimizer version 12.6 as the LP solver. The experiments were carried out on a machine Intel i7-8700 @ 3.20 GHz with 64 GB of RAM. A the limit of 24h was set to solve each instance. In our tests, instances R2_4_4, R2_4_8, RC2_4_4, and RC2_4_10 could not be solved within the time limit imposed.

For each one of the algorithms, we retrieve the number of column generation iterations (**#iters**) and the CPU time (**T(s)**). We present average values for each instance class. Additionally, we compute the `ratio` of **#iters** and **T(s)** for `std`-CG relative to `dyn`-SAR. We report average, minimum and maximum values.

Tables 6.1 and 6.2 summarize our computational experiments for the `std`-CG and the `dyn`-SAR. As a general observation, `dyn`-SAR helps in reducing the number of column generation iterations. For all classes of instances except RC1 with 200 customers, on average, `dyn`-SAR was capable of decreasing the number of iterations by at least 20%. Regarding the

computational time, dyn-SAR yields considerable speedups when solving instances with long routes (C2, R2, and RC2). This observation is somehow expected, given that these instances are associated with long routes. Long routes are associated with dense columns, which is one of the sources of degeneracy [49]. Our method, in turn, is effective in tackling problems with such structures.

When we analyze relative values by comparing the CPU times, it may seem that the gains obtained for groups C2, R2, and RC2 are canceled by the losses in groups C1, R1, and RC1. However, we point out that the speedups yielded by dyn-SAR in absolute values are very significant. As described in Section 5.4.2, while routes associated with instances C1, R1, and RC1 tend to be short, i.e., on average 10 customers per route, routes for instances C2, R2, and RC2 tend to be long, i.e., around 20 customers per route. These structures impact on the complexity of solving instances in each one of these groups. The fact that the routes in group 2 instances are long makes the number of variables with zero value in the primal bases higher, which is an indication of a highly degenerate problems. Because the dyn-SAR algorithm is suitable to couple with degeneracy, the good performance of the method is expected. On the other hand, for solving instances that do not exhibit as much degeneracy, the overhead of applying dyn-SAR may not pay off. In the cases where dyn-SAR is slower than std-CG, the slowdown factor is never greater than 5 min. On the other hand, dyn-SAR may accelerate the execution of the algorithm by a couple of hours when tackling hard instances.

Table 6.1 Summary results for instances with 200 customers

Group	std-CG		dyn-SAR		Ratio **#iters.**			Ratio **T(s)**		
	#iters.	**T(s)**	**#iters**	**T(s)**	avg.	min	max	avg.	min	max
C1	302.0	30.2	162.5	30.3	2.05	1.41	2.91	1.49	0.65	3.35
C2	1476.0	557.5	401.9	179.6	3.86	2.64	4.89	4.22	2.07	5.92
R1	332.1	69.2	284.7	104.4	1.19	0.83	1.67	0.70	0.53	1.01
R2	2220.5	4078.1	814.3	2277.5	2.72	1.41	6.78	2.26	1.11	3.65
RC1	346.2	68.3	349.9	97.7	0.99	0.65	1.48	0.68	0.47	0.84
RC2	2656.2	6177.6	1055.5	5173.7	2.49	1.77	3.98	1.56	0.98	2.89

6.3.2 Multi-person pose estimation

MPPE is the problem of identifying each unique person in an image and annotating their body parts. The MPPE has several applications in computer vision, including self-driving vehicles, rehabilitation, and military uses. This task is very complicated, given that people's appearance may notably change due to their clothes, body position, or other images in the background of the picture. Additionally, some parts of their body may be only partially

Table 6.2 Summary results for instances with 400 customers

Group	std-CG		dyn-SAR		Ratio #iters.			Ratio T(s)		
	#iters.	T(s)	#iters	T(s)	avg.	min	max	avg.	min	max
C1	722.7	175.1	300.8	205.0	2.42	2.15	2.61	0.98	0.68	1.54
C2	3324.9	5044.3	760.6	2157.2	4.51	3.64	5.65	2.61	1.88	3.36
R1	760.3	447.2	486.4	636.1	1.59	0.85	2.24	0.68	0.42	0.94
R2	6046.0	18461.0	1171.6	8208.6	5.10	1.86	10.41	4.16	1.44	6.32
RC1	754.7	380.1	528.0	514.6	1.43	0.98	1.97	0.71	0.44	0.85
RC2	8402.4	27916.8	1438.9	21855.2	5.87	3.31	8.98	2.55	0.83	6.16

visible due to occlusion. In the MPPE, the set of items corresponds to body part detections that are generated by deep neural networks [221, 222]. The problem then aims at identifying which detections are associated with a given person, where each person is modeled according to a tree structure [223–225]. These detections are associated with fourteen human body parts, namely: head, neck, left/right shoulders, elbows, wrists, hips, knees, and ankles.

Let N be the set of detections. For each detection $i \in N$, a parameter R_i indicates to which body part a given detection is associated. Thus, we can define the set N^r that corresponds to the set of detections associated with a body part r. Let Ω be the potential set of persons, which is defined as being the power set of N. It is important to mention that a person may be associated with more than one detection of any given body part. Furthermore, a person may not include any detection of a given body part due to occlusion. When modeling the problem, a binary parameter a_j^i indicates if a detection $i \in N$ is assigned to a person $j \in \Omega$.

The cost structure of the MPPE is defined as follows. For each detection $i \in N$, we define a cost φ_i^1 that associates each detection to a person. Moreover, for each pair of detections $i_1, i_2 \in N$, a cost φ_{i_1,i_2}^2 indicates if both detections should be associated with the same person. Positive/negative values discourage/encourage the presence of i_1 and i_2 jointly in the same person. Because each person is modeled as being a tree, costs φ_{i_1,i_2}^2 are non-zero if $R_{i_1} = R_{i_2}$ or if R_{i_1} is adjacent to R_{i_2}. A cost φ^0 measures a Bayesian belief that a given number of persons will be in the image. Positive/negative values of φ^0 discourage/encourage the presence of more persons in the packing. The cost of a pose (person) is then computed as follows:

$$c_j = \varphi^0 + \sum_{i \in N} \varphi_i^1 a_j^i + \sum_{i \in N} a_j^i \varphi_i^1 \sum_{\substack{i_1 \in N \\ i_2 \in N}} a_j^{i_1} a_j^{i_2} \varphi_{i_1,i_2} \tag{6.16}$$

To solve the MPPE using CG, one models the problem as being a minimum weight set packing (MWSP) problem. A MWSP can be represented by (6.4)–(6.6), except that in constraints

(6.5) the sign $\geq$ must be replaced by $\leq$. In this formulation, each constraint is associated with a detection $i \in N$. When solving the PS for the MPPE, one starts by identifying neck detections from N. The pricing problem then consists of finding least-cost poses (person) rooted at each one of these neck detections. The reduced cost of a pose is computed as $\bar{c}_j = c_j - \sum_{i \in N} a_j^i \pi_i$, where $\pi_i \in N$ are dual variables coming from the RMP. For more details on how to model and solve computer vision problems using CG, and specifically the MPPE, the interested reader is referred to [12, 13].

Algorithm

With a view of evaluating the performance of `dyn`-SAR in solving the MPPE, we consider two methods: a standard implementation of a CG algorithm (`std`-CG) and a CG method that employs the *varying dual optimal inequalities* of Yarkony et al. [12] (`DOI`) as stabilization method. For more details on the latter, the reader is referred to Section 2.3. To the best of our knowledge, the latter is one of the state-of-the-art stabilization strategies for CG applied to the MPPE. For all three settings – `std`-CG, `DOI`, and `dyn`-SAR – the dynamic programming pricing subproblems are solved using the nested Benders decomposition algorithm of Wang et al. [13].

More specifically for the `dyn`-SAR method, the algorithm starts by solving a relaxation of (6.10)– (6.12) containing a single constraint (6.11) defined for $S = N$. When applying `dyn`-SAR for the MPPE, we do not consider specific neighborhoods. Rather than relying on static attributes of the detections, we look at the basic solution obtained once CG has been solved to optimality. Every time a problem $\mathcal{R}$ is solved, for each active pose (basic variable), we create a constraint for all the members (detections) of that pose that are over-included. No limits are imposed on the number of items per constraint (`maxNbItemsPerSet` $= \infty$), nor on the number of constraints added to the RMP at each round of separation (`maxNbConstrsToAdd` $= \infty$). As a consequence, the procedure `maxDispersionHeuristic` is not executed. The `dyn`-SAR algorithm stops when any of the detections is over-included, and no more columns with negative reduced cost can be found to be added to the ERMP.

Instances

To evaluate the performance of our method when solving MPPE instances, we consider the MPII-multi-person dataset [226]. This dataset is composed of 418 instances, each one containing approximately 10,000 detections each. The cost terms and the problem structure are similar to the ones in [222], but they are modified to increase the convergence of the algorithm. Details about this preprocessing step are given in [12].

Experiments

Experiments for the MPPE were carried out on a machine Intel i7-6850K @ 3.60 GHz. Algorithms described in the previous section were coded in Matlab 2017 and employed the built-in Matlab solver based on interior points as the LP solver. For each instance and setting, we collect the total CPU time (**T(s)**) and the number of CG iterations (**#iters**). Additionally, we report relative values of **T(s)** and **#iters** for `std`-CG relative to `dyn`-SAR, and for `dyn`-SAR relative to `DOI`. The former allows us to identify speedups achieved by `dyn`-SAR in comparison with `std`-CG, whereas the latter enables us to assess `dyn`-SAR performance when compared to a CG algorithm tailored with dual optimal inequalities.

In our analysis, we decided to only consider instances for which the solution process imposed some challenges to the methods. For this reason, we only report results for the 30 instances for which `std-CG` took the longest to solve. Considering easy instances might affect negatively the quality of the conclusions drawn.

The results in Table 6.3 show that `dyn`-SAR can yield considerable speedups when compared to `std`-CG. On average, `dyn`-SAR is 16 times faster than the `std`-CG and can reduce the number of iterations by half. When compared to the method based on the use of tailored DOIs, `dyn`-SAR is, on average, 3 times slower and requires 5 times more iterations. This worse performance is expected given that `DOI` is an approach tailored to solve the MPPE, where `dyn`-SAR is a generic method and can be applied to solve different problems.

Table 6.3 Results for the Multi-Person Pose Estimation

Instance	std-CG		DOI		dyn-SAR		std-CG / dyn-SAR		dyn-SAR / DOI	
	T(s)	**#iters**	**T(s)**	**#iters**	**T(s)**	**#iters**	**T(s)**	**#iters**	**T(s)**	**#iters**
0051	43.6	90	3.3	10	10.1	58	4.3	1.6	3.1	5.8
0118	49.4	120	8.0	26	20.9	123	2.4	1.0	2.6	4.7
0551	52.1	70	8.6	14	18.4	53	2.8	1.3	2.1	3.8
0590	101.8	67	20.8	25	51.8	72	2.0	0.9	2.5	2.9
0807	43.8	71	6.4	19	24.0	92	1.8	0.8	3.7	4.8
0842	151.1	162	6.6	20	15.1	72	10.0	2.3	2.3	3.6
0858	74.3	78	12.2	16	40.2	68	1.8	1.1	3.3	4.3
0924	1833.6	394	6.2	17	18.4	65	99.8	6.1	3.0	3.8
1018	56.0	77	4.3	11	9.8	45	5.7	1.7	2.3	4.1
1027	75.5	149	6.0	17	15.1	87	5.0	1.7	2.5	5.1
1064	189.3	180	3.8	13	8.9	69	21.3	2.6	2.3	5.3
1341	792.6	317	5.3	14	14.3	72	55.4	4.4	2.7	5.1
1366	430.7	206	5.3	16	11.8	58	36.5	3.6	2.2	3.6
1547	43.5	50	15.3	17	39.0	88	1.1	0.6	2.6	5.2
1742	66.7	80	5.9	12	30.2	79	2.2	1.0	5.1	6.6
2290	131.0	99	8.5	23	44.2	124	3.0	0.8	5.2	5.4
2296	72.6	78	3.4	12	8.2	42	8.8	1.9	2.5	3.5
2311	98.0	82	4.9	9	20.0	72	4.9	1.1	4.1	8.0
2531	266.3	316	4.7	18	14.7	96	18.1	3.3	3.1	5.3
2606	63.3	117	2.8	10	5.1	31	12.4	3.8	1.8	3.1
3003	250.2	206	5.7	24	28.9	136	8.7	1.5	5.1	5.7
3079	120.9	219	4.6	18	5.9	46	20.3	4.8	1.3	2.6
3156	64.1	92	3.3	10	13.2	61	4.9	1.5	4.0	6.1
3322	399.0	319	2.0	12	4.9	50	81.8	6.4	2.5	4.2
3324	60.8	161	2.4	16	4.1	60	14.8	2.7	1.7	3.8
3328	52.3	111	3.7	19	6.3	63	8.3	1.8	1.7	3.3
3398	211.2	240	2.7	8	10.9	59	19.5	4.1	4.0	7.4
3466	47.9	72	4.9	11	14.5	72	3.3	1.0	3.0	6.5
3735	546.8	322	9.4	16	45.9	124	11.9	2.6	4.9	7.8
3774	87.9	137	3.1	10	8.1	44	10.9	3.1	2.6	4.4
Average	215.9	156.1	6.1	15.4	18.8	72.7	16.1	2.4	3.0	4.9

6.3.3 Bin Packing Problem with Conflicts

The BPPC can be defined as follows. Let $N = \{1, \ldots, n\}$ be a set of n items. With each item $i \in N$ it is associated a weight w_i and a conflict set $C_i \subset N$, that contains all items in conflict with i. Moreover, let B be an infinite set of bins, each one having capacity W. In the BPPC, there is a *conflict graph* $G = (N, E)$, where an edge $(i, j) \in E$ exists if items i and j are in conflict, i.e., $i \in C_j$ and $j \in C_i$. From graph G, it is possible to obtain an *extended conflict graph* $G' = (N, E')$, where the set of edges is defined as $E' = E \cup \{\{i, j\} : i, j \in N \text{ and } w_i + w_j \geq W\}$. The BPPC consists in finding an assignment of the items to the bins such that: each item $i \in N$ is assigned to exactly one bin; the number of bins used is minimized; the total weight of the items in any given bin does not exceed W; and no pair items in a bin are in conflict.

CG algorithms designed to solve the BPPC may consider a set partitioning or covering formulation [19, 68, 227, 228]. From formulation (6.7)–(6.9): Ω represents the set of all assignment patterns (i.e., subsets of items assigned to a single bin); the binary parameter a_j^i takes value 1 if item $i \in N$ belongs to pattern $j \in \Omega$, or 0, otherwise; and the cost of a pattern is such that $c_j = 1$. The PS from the BPPC is modeled as a knapsack problem with conflicts (KPC). This problem is typically solved in the literature using a MIP solver [227, 228], but some more efficient approaches have been recently devised [19]. The reduced cost of each variable (pattern) is computed with expression (6.15).

Algorithm

To evaluate the performance of the `dyn`-SAR method, we compare our results with those obtained by a standard CG algorithm (`std`-CG). The MP considered in the latter is modeled using a set-covering formulation. When applying `dyn`-SAR to the BPPC, the EMP is derived from a set-covering formulation as well. In both methods, the KPC defining the PS is solved with the depth-first-search BB algorithm proposed by Sadykov and Vanderbeck [19]. This method consists of a recursive approach that combines a classic BB framework, with the solution of the continuous relaxation of 0-1 knapsack problems to derive bounds, and an enumeration algorithm used to solve maximum clique problems. In our implementation, only a single best-reduced cost column is added at each iteration.

Because we wanted to evaluate the impact of applying `dyn`-SAR when solving the BPPC, we tried to keep the algorithm as simple as possible. For this reason, we do not implement the prepossessing step described in [227], nor we consider any heuristic pricing algorithm to solve the KPC. As a matter of fact, Sadykov and Vanderbeck [19] observed that in their implemen-

tation, using non-exact pricing tended to slow down the convergence of the CG procedure. Furthermore, the BB algorithm has proven to scale well for the instances considered, which does not justify the use of heuristic pricing methods.

Regarding the `dyn`-SAR algorithm, the initial relaxation solved in the method is defined by partitioning the items as follows. Initially, we sort the items in non-increasing order according to their weights. Then, we build sets S that contain, respectively, the first 2κ items, then the following 4κ items, and so on, until all the items have been added to some set S. In our implementation, $\kappa = 1$. During this first step, conflicts among items are not taken into account.

The neighborhoods considered in method `buildViolatedSets` in the Algorithm 2 are generated as follows. For each item $i \in N$, neighborhoods are such that: $\mathcal{H}(i) = \{\forall j \in N : i \neq j,\ |w_i - w_j| \leq W/10$ and $|C_i \,\Delta\, C_j| \leq 3|N|/10\}$. That is, neighborhoods contain items that are similar in weight, and for which the cardinality of the symmetric difference between the respective conflict sets is smaller than 30% of the size of the instance. We impose $|\mathcal{H}(i)| \leq |N|/10$. Finally, in Algorithm 2 for the BPPC, `maxNbItemsPerSet` is set to the average number of items in the bins associated with the basic variables in the previous iteration of the CG procedure, and `maxNbConstrsToAdd` is limited to $|N|/10$. The algorithm is executed when all items in N have been fully covered and when no more columns with negative reduced cost can be found to be added to the ERMP.

Instances

We tested the performance of our method on instances with arbitrary conflict graphs generated according to the procedure described by Sadykov and Vanderbeck [19]. The choice for instances with this type of graph is based on the conclusions drawn by Sadykov and Vanderbeck, who state that instances with arbitrary conflict graphs are significantly harder to solve than instances with interval conflict graphs [227, 229].

The instances that we generate are based on the traditional dataset proposed by Gendreau et al. [229]. There are eight classes of instances. The first four, denoted as `u`, have bins with capacity 150, and items with integer weights that are obtained randomly from the interval $[20, 100]$. The number of items n in each class of instances are, 120, 250, 500, and 1,000, respectively. The remaining instances have a *triplet* structure, i.e., each bin contains exactly three items. These instances are denoted `t` and have bins with capacity $W = 1,000$. The weight of items is generated as follows. At first, integer weights for two items $i, j \in N$ are randomly generated from $[20, 100]$. Then, the weight of a third item k is such that $w_k = W - w_i - w_j$. The four classes of instances with a triplet structure have, respectively,

$60, 120, 249, 501$ items. For each instance class – `u120`, `u250`, `u500`, `u1000`, `t60`, `t120`, `t249`, `t501` – we generate 10 instances. For each instance, nine different instances are produced, each one associated with one density in $D = \{0.1, 0.2, \ldots, 0.9\}$, resulting in 90 instances per class. For each density $d \in D$, the procedure to generate conflict graphs consists in generating pairs (i, j), such that $i, j \in N$ and $i < j$, without repetition, until $d\,\frac{n(n-1)}{2}$ pairs have been generated.

Experiments

In our experiments, both algorithms (`std`-CG and `dyn`-SAR) were coded in C++, and IBM CPLEX Optimizer version 12.8 was employed as the LP solver. The tests were carried out on a machine Intel Xeon E5-2637 v2 @ 3.50 GHz with 128 GB of RAM. A time limit of 10h was set to solve each instance.

Tables 6.4 and 6.5 present average results for instances `t` and `u`, respectively. Each line in these tables provides average values obtained over all instances in the class. To avoid noise in our analysis, we only report results for instances for which `std`-CG required more than 15s. For each instance, we collect the CPU time (**T(s)**) and the number of column generation iterations (**#iters**). Additionally, for each instance class, we compute the `ratio` for **#iters** and **T(s)** of `std`-CG relative to `dyn`-SAR. We report average, minimum and maximum values.

Numerical results show that, for some groups of instances, `dyn`-SAR can yield speedups superior to an order of magnitude. This observation is more notable when addressing triplet instances. These types of instances are among the hardest BPPC instances in the literature [227, 230]. Regarding the number of iterations, one may notice that `dyn`-SAR requires consistently fewer iterations, but the reductions only neared 5%. This shows that the `dyn`-SAR method tends to require, on average, less CPU time per iteration. Two aspects may explain the better performance yielded by `dyn`-SAR. First, even if the ERMP may end with more constraints than the original RMP, the number of constraints at the beginning of the algorithm is considerably small. Hence, the overall CPU time required to solve the ERMP is smaller. The second aspect to be taken into account concerns the distribution of dual variables. It appears during our computational experiments that `dyn`-SAR produces dual variables that yield easier-to-solve PSs. For the instances with weights uniformly distributed, the superiority of `dyn`-SAR was less drastic, though still very significant.

Table 6.4 Summary results – Bin Packing Problem with Conflicts – Triplet instances

Group	std-CG		dyn-SAR		ratio #iters.			ratio T(s)		
	#iters.	T(s)	#iters	T(s)	avg.	min	max	avg.	min	max
ta249_1	853	333.4	786	34.0	1.09	1.04	1.15	9.87	8.40	11.10
ta249_2	833	179.4	791	23.7	1.05	0.98	1.16	7.66	6.35	8.54
ta249_3	859	95.7	800	15.7	1.07	1.03	1.13	6.20	5.02	7.36
ta249_4	870	46.3	814	13.0	1.07	1.02	1.10	3.59	3.13	4.04
ta249_5	892	20.9	843	11.5	1.06	1.01	1.10	1.84	1.59	2.34
ta501_1	1637	11,409.3	1552	554.6	1.05	1.01	1.07	20.93	16.76	24.50
ta501_2	1619	6,268.4	1563	339.1	1.04	1.02	1.06	19.07	13.48	22.98
ta501_3	1619	3,169.5	1561	168.9	1.04	0.99	1.06	19.02	15.74	22.29
ta501_4	1644	1,423.1	1575	93.9	1.04	1.00	1.08	15.73	12.09	20.70
ta501_5	1684	573.6	1599	53.2	1.05	1.03	1.08	11.04	8.60	13.85
ta501_6	1731	209.8	1623	38.3	1.07	1.04	1.12	5.90	2.84	7.48
ta501_7	1636	68.0	1552	22.8	1.06	1.03	1.12	3.39	2.08	6.23
ta501_8	1595	28.4	1491	19.6	1.07	1.03	1.11	1.59	1.22	2.32

Table 6.5 Summary results – Bin Packing Problem with Conflicts – Uniform instances

Group	std-CG		dyn-SAR		ratio #iters.			ratio T(s)		
	#iters.	T(s)	#iters	T(s)	avg.	min	max	avg.	min	max
ua500_1	1430	145.7	1385	36.6	1.07	1.03	1.11	3.24	1.19	8.78
ua500_2	1436	32.7	1357	20.2	1.06	1.00	1.09	1.69	0.88	2.40
ua500_3	1436	17.8	1355	13.7	1.06	1.02	1.10	1.35	1.10	1.72
ua1000_1	2822	11,693.2	2616	1,441.6	1.08	1.05	1.10	13.13	1.89	29.07
ua1000_2	2817	2,219.2	2598	266.3	1.08	1.04	1.11	9.47	3.00	19.98
ua1000_3	2792	322.5	2589	100.1	1.08	1.05	1.11	3.32	1.28	7.30
ua1000_4	2786	101.9	2574	68.9	1.08	1.06	1.11	1.48	1.16	1.75
ua1000_5	2804	74.5	2575	55.6	1.09	1.06	1.12	1.34	1.24	1.59
ua1000_6	2806	61.9	2589	51.3	1.08	1.06	1.12	1.21	1.11	1.37
ua1000_7	2830	57.0	2612	48.1	1.08	1.07	1.12	1.19	1.12	1.28
ua1000_8	2943	51.2	2717	48.1	1.08	1.05	1.14	1.07	0.97	1.16
ua1000_9	3196	51.0	2900	52.6	1.10	1.07	1.12	0.97	0.89	1.04

6.4 Conclusions

In this chapter, we have proposed a new generic stabilization framework, called `dyn`-SAR, to tackle highly degenerate CG problems. Contrarily to other approaches in the literature, our method has the advantage of being simple and easy to understand and implement. Moreover, `dyn`-SAR has shown to be a general method. This observation is confirmed by the distinct structures of the problems solved in our experiments. Computational tests have shown that `dyn`-SAR can yield considerable gains in terms of running time and the number of iterations performed by the algorithm. As future work, we intend to design a procedure to remove constraints that are no longer useful in the algorithm. Moreover, we plan to investigate the `dyn`-SAR method from a theoretical point of view. This will help us to understand the theoretical foundations behind `dyn`-SAR, which may be useful in designing more general stabilization methods.

CHAPTER 7 GENERAL DISCUSSION

The methodological survey on BPCs for VRPs represents an important tool for CG practitioners. While it can serve the traditional purpose of a survey – which is to provide a general overview of the literature to allow the identification of potential research avenues –, it is not limited to the latter. The motivation behind our study was to provide a tutorial/guide for researchers who intend to work on the development of CG-based algorithms for VRPs. According to Feillet [21], studies with this nature are important because, in spite of being popular, BPC methods are typically hard to understand and reproduce due to 1) their complexity and the large literature in the field; 2) the variety of perspectives from which the method can be explored; 3) the lack of comprehensible descriptions of the method. In this context, our survey aimed at filling the gap in the literature regarding the existence of works that explain with details the main components of BPC algorithms. To the best of our knowledge, in the recent literature, there are no research works similar to ours. The studies by Poggi and Uchoa [94] and Baldacci et al. [78] present general reviews on the development of exact algorithms to classic VRP variants like the CVRP and the VRPTW. Our work is broader in the sense that it highlights and discusses the main modeling and methodological strategies proposed over the years to design BPC algorithms for different VRPs. In our study, we not only describe each one of the strategies but also, when convenient, provide further information regarding implementation aspects. Therefore, the survey can be useful for either people who want to learn more about the development of BPC algorithms in general, or for experienced researchers who may be interested in more technical details.

From our survey and also from the literature review presented in Chapter 2, we could identify the two remaining aspects that motivated this thesis. First, despite the intense work in the area in the last years, there is still a need for new mechanisms that allow designing more efficient algorithms to solve VRPs. Second, there exists a lack of generic stabilization techniques, i.e., methods that can be applied to problems with different structures. Regarding the former aspect, our selective pricing algorithm devised to solve (E)SPPRCs defining pricing subproblems in vehicle routing applications corresponds to one more attempt in this direction. When compared to the algorithm for the *arc-ng*-SPPRC developed by Bulhões et al., the dominance relations yielded by the selective pricing are stronger and more general. Moreover, to have its potential fully exploited, Bulhões et al.'s strategy requires careful implementation and the use of a cumbersome data structure. In turn, contrary to the selective pricing devised by Desaulniers et al. [107], the new selective method is easier to implement. Whereas the latter can be applied simply by modifying the dominance rule, the former requires the addition

of several data structures and sub-procedures, which may add some overhead to the method.

Finally, with respect to the development of more general stabilization techniques, our `dyn`-SAR algorithm represents a good alternative. It does not require the matrix of coefficients to have any particular structure [66–68], and can be applied to solve problems modeled as set-partitioning, covering or packing problems [49, 52]. Despite having constraints generated dynamically, due to the extended formulation in which the method is implicitly based, all the dual information is available at all times. For this reason, no changes at the pricing problem level are required [52], nor the dual variables must be estimated [55] or projected [56]. The `dyn`-SAR method is easy to understand because it does not require complex concepts [63], and its implementation is straightforward. Indeed, it is possible to employ directly a pricing algorithm that is used in a standard CG method.

CHAPTER 8 CONCLUSION AND RECOMMENDATIONS

In this thesis, we discussed and proposed techniques to enhance CG algorithms applied to COPs. In Section 8.1 we summarize the works derived from this thesis. In Section 8.2, we discuss the main limitations for each one of the ideas proposed and suggest potential research avenues to be exploited.

8.1 Summary of Works

Motivated by the lack of studies concerning the existence of works with comprehensive explanations about BPCs algorithms, we wrote a methodological survey that is presented in Chapter 4. In this study, we highlight and discuss the main algorithmic and modeling contributions made over the years in the context of BPCs for VRPs. Our survey is divided into two main parts. In the first part, we present ideas that are applicable to different VRP variants. We discuss topics related to pricing algorithms, cutting separation, branching strategies, and stabilization. The second part is more problem-oriented. We describe how attributes such as heterogeneous fleet, multi-depots, soft time windows, split deliveries, time dependency, pickup and deliveries, uncertainty, and environmental aspects are handled when designing BPC algorithms. We believe that this survey paper will help researchers to identify potential improvements to be made on BPC algorithms for VRPs, hence, helping to advance the research in the domain.

In Chapter 5, we propose a selective pricing algorithm that is based on a new dominance criterion for the *arc-ng*-SPPRC. This dominance rule extends the set-based dominance rule proposed by Bulhões et al. [1], making it more general and stronger. One of the main advantages of using the new strategy is that it can be easily implemented, once it does not rely on complex data structures or sub-procedures [1, 107]. Computational results performed over VRPTW instances show that the proposed mechanism helps in reducing the number of non-dominated labels kept by the labeling algorithm and, as a consequence, the CPU time.

Finally, in Chapter 6, we developed a stabilization method to tackle COPs with degenerate structures. The new stabilization framework, called `dyn`-SAR, is based on the dynamic separation of aggregated constraints, which are obtained by summing up constraints from the RMP. The stabilization effect induced by the `dyn`-SAR method comes from the fact that it yields stronger interactions between the dual variables, which is not observed when solving explicitly a set-partitioning (covering/packing) formulation. The interest in using the

dyn-SAR algorithm is twofold. First, implementing dyn-SAR is straightforward. Second, dyn-SAR is capable of solving problems with different structures. The latter aspect is illustrated in our experiments, where we solve instances from problems differing considerably in terms of objective function and pricing subproblem, namely: the VRPTW, the MPPE, and the BPPC. Numerical results show a clear advantage of the dyn-SAR algorithm over a standard CG method in reducing both the number of iterations and the CPU time.

8.2 Limitations and future research

In our survey, we focused on presenting ideas developed in the context of BPC algorithms applied to problems that are related to the CVRP. As a consequence, numerous traditional routing problems were left out, namely: vehicle scheduling problem, inventory routing, ship routing, location routing, among others. Given that CG has become the leading technique for solving different classes of COPs, tailored strategies have certainly been developed to exploit the specific structure of these problems. For this reason, we believe that methodological studies regarding BPC methods for different routing problems or even other COPs would become a powerful research tool.

The selective pricing algorithm that we describe in Chapter 5 employs a dominance relation that relies on the set of forbidden extensions associated with the predecessor of a label to decide if keeping this label is beneficial or not. This strategy has the potential of yielding sharper dominance rules and avoiding unpromising extensions. In our method, we only look at the immediate predecessor of a label. An extension of our idea would be to investigate the impact of considering the set of forbidden extensions of other predecessors further down in the path. On the one hand, this generalization has the potential of strengthening even more dominance rules. On the other hand, a more elaborate data structure would be required to cope with the overhead incurred by the new relations. The proof that we provide holds only for the case where the immediate predecessor is considered. In this context, some future research may be done in two directions: 1) extend our proof to encompass the more general case, and 2) devise tailored data structures to accommodate the additional computational burden.

Regarding dyn-SAR, despite its efficiency in improving the convergence of CG algorithms applied to degenerate problems, we are not entirely aware of the theoretical properties guiding the method. All the intuition that we have developed so far was based on empirical evidence. For some problems such as the classical bin packing problem [10], dyn-SAR can be seen as an adapted (dynamic) version of the aggregation method applied to implement static dual inequalities [67]. However, for more complex problems such as the ones that we

considered in our analysis, this equivalency is not clear. Yet, in our method, when deciding which constraints from the RMP to aggregate, we tend to favor those which may have similar (not necessarily equal) dual variables. An analogous requirement is also sought when identifying dual inequalities to restrain the dual solution space. In light of this, we believe that investigating more in-depth the theoretical foundations behind `dyn`-SAR can help to gain some insights that may be useful when designing more general stabilization methods. A potential research direction that we point out concerns the study of dual inequalities for VRPs. The number of works in this field is scarce, not to say in-existent. To the best of our knowledge, the only paper in the literature proposing dual inequalities for VRPs is the one by Gschwind et al. [70]. However, the stabilization effect in this application comes from the special relationship between the demands of the customers in the problem. Thus, the method is very tailored to the problem being solved.

REFERENCES

[1] T. Bulhões, R. Sadykov, and E. Uchoa, "A branch-and-price algorithm for the minimum latency problem," *Computers & Operations Research*, vol. 93, pp. 66–78, 2018.

[2] R. Baldacci, A. Mingozzi, and R. Roberti, "New route relaxation and pricing strategies for the vehicle routing problem," *Operations Research*, vol. 59, no. 5, pp. 1269–1283, 2011.

[3] P. Toth and D. Vigo, Eds., *Vehicle routing: problems, methods and applications*. MOS/SIAM Series on Optimization, 2014.

[4] European Space Agency, "How many stars are there in the universe?" Accessed: January, 16, 2020. [Online]. Available: https://www.esa.int/Science_Exploration/Space_Science/Herschel/How_many_stars_are_there_in_the_Universe

[5] V. Maniezzo, T. Stützle, and S. Voß, Eds., *Matheuristics - Hybridizing Metaheuristics and Mathematical Programming*, ser. Annals of Information Systems. Springer, 2010, vol. 10.

[6] M. Gendreau and J.-Y. Potvin, Eds., *Handbook of Metaheuristics*, 3rd ed., ser. International Series in Operations Research & Management Science. Springer International Publishing, 2019.

[7] G. Desaulniers, J. Desrosiers, and M. M. Solomon, Eds., *Column Generation*, 1st ed. Springer US, 2005.

[8] G. Desaulniers, J. Desrosiers, I. Ioachim, M. M. Solomon, F. Soumis, and D. Villeneuve, "A unified framework for deterministic time constrained vehicle routing and crew scheduling problems," in *Fleet Management and Logistics*, T. G. Crainic and G. Laporte, Eds. Boston, MA: Springer US, 1998, ch. 3, pp. 57–93.

[9] L. Costa, C. Contardo, and G. Desaulniers, "Exact branch-price-and-cut algorithms for vehicle routing," *Transportation Science*, vol. 53, no. 4, pp. 946–985, 2019.

[10] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," *European Journal of Operational Research*, vol. 255, no. 1, pp. 1–20, 2016.

[11] T. Bulhões, R. Sadykov, A. Subramanian, and E. Uchoa, "On the exact solution of a large class of parallel machine scheduling problems," *Journal of Scheduling*, 2020, forthcoming.

[12] J. Yarkony, Y. Adulyasak, M. Singh, and G. Desaulniers, "Data association via set packing for computer vision applications," Montréal, Research report G-2019-42, Les Cahiers du GERAD, HEC Montréal, 2019.

[13] S. Wang, A. Ihler, K. Kording, and J. Yarkony, "Accelerating dynamic programs via nested benders decomposition with application to multi-person pose estimation," in *Proc. 15th European Conference on Computer Vision*, Munich, Germany, 2018, pp. 652–666.

[14] C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance, "Branch-and-price: Column generation for solving huge integer programs," *Operations Research*, vol. 46, no. 3, pp. 316–329, 1998.

[15] M. E. Lübbecke and J. Desrosiers, "Selected topics in column generation," *Operations Research*, vol. 53, no. 6, pp. 1007–1023, 2005.

[16] G. B. Dantzig and J. H. Ramser, "The truck dispatching problem," *Management Science*, vol. 6, no. 1, pp. 80–91, 1959.

[17] T. Vidal, G. Laporte, and P. Matl, "A concise guide to existing and emerging vehicle routing problem variants," *European Journal of Operational Research*, pp. 1–42, 2019, forthcoming.

[18] K. Jansen, "An Approximation Scheme for Bin Packing with Conflicts," *Journal of Combinatorial Optimization*, vol. 3, no. 4, pp. 363–377, 1999.

[19] R. Sadykov and F. Vanderbeck, "Bin packing with conflicts: A generic branch-and-price algorithm," *INFORMS Journal on Computing*, vol. 25, no. 2, pp. 244–255, 2013.

[20] J. Desrosiers, F. Soumis, and M. Desrochers, "Routing with time windows by column generation," *Networks*, vol. 14, no. 4, pp. 545–565, 1984.

[21] D. Feillet, "A tutorial on column generation and branch-and-price for vehicle routing problems," *4OR*, vol. 8, no. 4, pp. 407–424, 2010.

[22] B. Korte and J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 6th ed., ser. Algorithms and Combinatorics. Bonn, Germany: Springer-Verlag Berlin Heidelberg, 2018, vol. 21.

[23] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs," *Operations Research*, vol. 8, no. 1, pp. 101–111, 1960.

[24] F. Vanderbeck and L. A. Wolsey, *Reformulation and Decomposition of Integer Programs.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. 13, pp. 431–502.

[25] F. Margot, *Symmetry in Integer Linear Programming.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ch. 17, pp. 647–686.

[26] L. R. Ford and D. R. Fulkerson, "A suggested computation for maximal multi-commodity network flows," *Management Science*, vol. 50, no. 12_supplement, pp. 1778–1780, 2004.

[27] P. C. Gilmore and R. E. Gomory, "Linear programming approach cutting stock," *Operations Research*, vol. 9, no. 6, pp. 849–859, 1961.

[28] I. Elhallaoui, A. Metrane, G. Desaulniers, and F. Soumis, "An improved primal simplex algorithm for degenerate linear programs," *INFORMS Journal on Computing*, vol. 23, no. 4, pp. 569–577, 2011.

[29] A. Zaghrouti, F. Soumis, and I. El Hallaoui, "Integral simplex using decomposition for the set partitioning problem," *Operations Research*, vol. 62, no. 2, pp. 435–449, 2014.

[30] L. A. Wolsey, *Integer Programming.* New York, NY, USA: Wiley-Interscience, 1998.

[31] G. L. Nemhauser and L. A. Wolsey, *Integer and combinatorial optimization.* New York, NY, USA: Wiley-Interscience, 1988.

[32] G. L. Nemhauser and S. Park, "A polyhedral approach to edge coloring," *Operations Research Letters*, vol. 10, no. 6, pp. 315–322, 1991.

[33] M. Poggi de Aragão and E. Uchoa, "Integer program reformulation for robust branch-and-cut-and-price algorithms," in *Proceedings of the Conference Mathematical Programming in Rio: A Conference in Honour of Nelson Maculan*, 2003, pp. 56–61.

[34] E. Uchoa, R. Fukasawa, J. Lysgaard, A. Pessoa, M. P. De Aragão, and D. Andrade, "Robust branch-cut-and-price for the Capacitated Minimum Spanning Tree problem over a large extended formulation," *Mathematical Programming*, vol. 112, no. 2, pp. 443–472, 2008.

[35] A. Pessoa, E. Uchoa, M. P. de Aragão, and R. Rodrigues, "Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems," *Mathematical Programming Computation*, vol. 2, no. 3, pp. 259–290, 2010.

[36] G. Desaulniers, J. Desrosiers, and S. Spoorendonk, "Cutting planes for branch-and-price algorithms," *Networks*, vol. 58, no. 4, pp. 301–310, 2011.

[37] L. Caccetta and A. Kulanoot, "Computational aspects of hard knapsack problems," in *Proceedings of the Third World Congress of Nonlinear Analysts*, vol. 47, no. 8, 2001, pp. 5547–5558.

[38] M. Dror, "Note on the complexity of the shortest path models for column generation in VRPTW," *Operations Research*, vol. 42, no. 5, pp. 977–978, 1994.

[39] D. Pisinger, "A minimal algorithm for the 0-1 knapsack problem," *Operations Research*, vol. 45, no. 5, pp. 758–767, 1997.

[40] M. Desrochers and F. Soumis, "A generalized permanent labeling algorithm for the shortest path problem with time windows," *INFOR*, vol. 26, pp. 191–212, 1988.

[41] N. Boland, J. Dethridge, and I. Dumitrescu, "Accelerated label setting algorithms for the elementary resource constrained shortest path problem," *Operations Research Letters*, vol. 34, no. 1, pp. 58–68, 2006.

[42] G. Righini and M. Salani, "New dynamic programming algorithms for the resource constrained elementary shortest path problem," *Networks*, vol. 51, no. 3, pp. 155–170, 2008.

[43] R. Fukasawa, H. Longo, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, and R. F. Werneck, "Robust branch-and-cut-and-price for the capacitated vehicle routing problem," *Mathematical Programming*, vol. 106, no. 3, pp. 491–511, 2006.

[44] G. Desaulniers, F. Lessard, and A. Hadjar, "Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows," *Transportation Science*, vol. 42, no. 3, pp. 387–404, 2008.

[45] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa, "Improved branch-cut-and-price for capacitated vehicle routing," *Mathematical Programming Computation*, vol. 9, no. 1, pp. 61–100, 2017.

[46] D. Pecin, C. Contardo, G. Desaulniers, and E. Uchoa, "New enhancements for the exact solution of the vehicle routing problem with time windows," *INFORMS Journal on Computing*, vol. 29, no. 3, pp. 489–502, 2017.

[47] A. Pessoa, R. Sadykov, and E. Uchoa, "Enhanced branch-cut-and-price algorithm for heterogeneous fleet vehicle routing problems," *European Journal of Operational Research*, vol. 270, no. 2, pp. 530–543, 2018.

[48] F. Quesnel, G. Desaulniers, and F. Soumis, "Improving air crew rostering by considering crew preferences in the crew pairing problem," *Transportation Science*, pp. 1–18, 2019, forthcoming.

[49] I. Elhallaoui, D. Villeneuve, F. Soumis, and G. Desaulniers, "Dynamic aggregation of set-partitioning constraints in column generation," *Operations Research*, vol. 53, no. 4, pp. 632–645, 2005.

[50] F. Vanderbeck, "On Dantzig-Wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm," *Operations Research*, vol. 48, no. 1, pp. 111–128, 2000.

[51] I. Elhallaoui, G. Desaulniers, A. Metrane, and F. Soumis, "Bi-dynamic constraint aggregation and subproblem reduction," *Computers & Operations Research*, vol. 35, no. 5, pp. 1713–1724, 2008.

[52] J. Desrosiers, J. B. Gauthier, and M. E. Lübbecke, "Row-reduced column generation for degenerate master problems," *European Journal of Operational Research*, vol. 236, no. 2, pp. 453–460, 2014.

[53] A. Tahir, G. Desaulniers, and I. El Hallaoui, "Integral column generation for the set partitioning problem," *EURO Journal on Transportation and Logistics*, vol. 8, no. 5, pp. 713–744, 2019.

[54] J. B. Gauthier, J. Desrosiers, and M. E. Lübbecke, "Tools for primal degenerate linear programs: IPS, DCA, and PE," *EURO Journal on Transportation and Logistics*, vol. 5, no. 2, pp. 161–204, 2016.

[55] İ. Muter, Ş. İ. Birbil, and K. Bülbül, "Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows," *Mathematical Programming*, vol. 142, no. 1, pp. 47–82, 2013.

[56] R. Sadykov and F. Vanderbeck, "Column generation for extended formulations," *EURO Journal on Computational Optimization*, vol. 1, pp. 81–115, 2013.

[57] R. E. Marsten, W. W. Hogan, and J. W. Blankenship, "The boxstep method for large-scale optimization," *Operations Research*, vol. 23, no. 3, pp. 389–405, 1975.

[58] O. du Merle, D. Villeneuve, J. Desrosiers, and P. Hansen, "Stabilized column generation," *Discrete Mathematics*, vol. 194, no. 1-3, pp. 229–237, 1999.

[59] A. Oukil, H. Ben Amor, J. Desrosiers, and H. El Gueddari, "Stabilized column generation for highly degenerate multiple-depot vehicle scheduling problems," *Computers & Operations Research*, vol. 34, no. 3, pp. 817–834, 2007.

[60] H. M. T. Ben Amor, J. Desrosiers, and A. Frangioni, "On the choice of explicit stabilizing terms in column generation," *Discrete Applied Mathematics*, vol. 157, no. 6, pp. 1167–1184, 2009.

[61] P. Wentges, "Weighted Dantzig-Wolfe decomposition for linear mixed-integer programming," *International Transactions in Operational Research*, vol. 4, no. 2, pp. 151–162, 1997.

[62] P. J. Neame, "Nonsmooth dual methods in integer programming," Ph.D. dissertation, University of Melbourne, Department of Mathematics and Statistics, 1999.

[63] A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck, "Automation and combination of linear-programming based stabilization techniques in column generation," *INFORMS Journal on Computing*, vol. 30, no. 2, pp. 339–360, 2018.

[64] L. M. Rousseau, M. Gendreau, and D. Feillet, "Interior point stabilization for column generation," *Operations Research Letters*, vol. 35, no. 5, pp. 660–668, 2007.

[65] J. Gondzio, P. González-Brevis, and P. Munari, "New developments in the primal-dual column generation technique," *European Journal of Operational Research*, vol. 224, no. 1, pp. 41–51, 2013.

[66] J. M. Valério de Carvalho, "Using extra dual cuts to accelerate column generation," *INFORMS Journal on Computing*, vol. 17, no. 2, pp. 175–182, 2005.

[67] H. Ben Amor, J. Desrosiers, and J. M. Valério de Carvalho, "Dual-optimal inequalities for stabilized column generation," *Operations Research*, vol. 54, no. 3, pp. 454–463, 2006.

[68] T. Gschwind and S. Irnich, "Dual inequalities for stabilized column generation revisited," *INFORMS Journal on Computing*, vol. 28, no. 1, pp. 175–194, 2016.

[69] K. Heßler, T. Gschwind, and S. Irnich, "Stabilized branch-and-price algorithms for vector packing problems," *European Journal of Operational Research*, vol. 271, no. 2, pp. 401–419, 2018.

[70] T. Gschwind, N. Bianchessi, and S. Irnich, "Stabilized branch-price-and-cut for the commodity-constrained split delivery vehicle routing problem," *European Journal of Operational Research*, vol. 278, no. 1, pp. 91–104, 2019.

[71] V. S. Lokhande, S. Wang, M. Singh, and J. Yarkony, "Accelerating column generation via flexible dual optimal inequalities with application to entity resolution," Research report, 2019. [Online]. Available: http://arxiv.org/abs/1909.05460

[72] N. Haghani, C. Contardo, and J. Yarkony, "Smooth and flexible dual optimal inequalities," Research report, 2020. [Online]. Available: http://arxiv.org/abs/2001.02267

[73] B. Eksioglu, A. Volkan, and A. Reisman, "The vehicle routing problem : A taxonomic review," *Computers & Industrial Engineering*, vol. 57, no. 4, pp. 1472–1483, 2009.

[74] S. Irnich, P. Toth, and D. Vigo, "The family of vehicle routing problems," in *Vehicle Routing: Problems, Methods and Applications*, 2nd ed., P. Toth and D. Vigo, Eds. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014, ch. 1, pp. 1–33.

[75] K. Braekers, K. Ramaekers, and I. V. Nieuwenhuyse, "The vehicle routing problem: State of the art classification and review," *Computers & Industrial Engineering*, vol. 99, pp. 300–313, 2016.

[76] G. Laporte, Y. Nobert, and M. Desrochers, "Optimal routing under capacity and distance restrictions," *Operations Research*, vol. 33, no. 5, pp. 1050–1073, 1985.

[77] J. Lysgaard, A. N. Letchford, and R. W. Eglese, "A new branch-and-cut algorithm for the capacitated vehicle routing problem," *Mathematical Programming*, vol. 100, no. 2, pp. 423–445, 2004.

[78] R. Baldacci, A. Mingozzi, and R. Roberti, "Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints," *European Journal of Operational Research*, vol. 218, no. 1, pp. 1–6, 2012.

[79] G. Desaulniers, O. B. G. Madsen, and S. Ropke, "The vehicle routing problem with time windows," in *Vehicle Routing: Problems, Methods and Applications*, 2nd ed., P. Toth and D. Vigo, Eds. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014, ch. 5, pp. 119–159.

[80] M. L. Balinski and R. E. Quandt, "On an integer program for a delivery problem," *Operations Research*, vol. 12, no. 2, pp. 300–304, 1964.

[81] S. Irnich and G. Desaulniers, "Shortest path problems with resource constraints," in *Column Generation*, 1st ed., G. Desaulniers, J. Desrosiers, and M. M. Solomon, Eds. Boston, MA: Springer US, 2005, ch. 2, pp. 33–65.

[82] S. Irnich, "Resource extension functions: Properties, inversion and generalization to segments," *OR Spectrum*, vol. 30, no. 1, pp. 113–148, 2008.

[83] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin, *Network flows: Theory, algorithms, and applications*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1993.

[84] M. Desrochers, J. Desrosiers, and M. Solomon, "A new optimization algorithm for the vehicle routing problem with time windows," *Operations Research*, vol. 40, no. 2, pp. 342–354, 1992.

[85] D. Villeneuve and G. Desaulniers, "The shortest path problem with forbidden paths," *European Journal of Operational Research*, vol. 53, no. 4, pp. 97–107, 2005.

[86] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen, "An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems," *Networks*, vol. 44, no. 3, pp. 216–229, 2004.

[87] J. E. Beasley and N. Christofides, "An algorithm for the resource constrained shortest path problem," *Networks*, vol. 19, no. 4, pp. 379–394, 1989.

[88] A. Chabrier, "Vehicle routing problem with elementary shortest path based column generation," *Computers & Operations Research*, vol. 33, no. 10, pp. 2972–2990, 2006.

[89] C. Contardo, G. Desaulniers, and F. Lessard, "Reaching the elementary lower bound in the vehicle routing problem with time windows," *Networks*, vol. 65, no. 1, pp. 88–99, 2015.

[90] N. Christofides, A. Mingozzi, and P. Toth, "Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations," *Mathematical Programming*, vol. 20, no. 1, pp. 255–282, 1981.

[91] D. Houck, J. Picard, M. Queyranne, and R. Vemuganti, "The travelling salesman problem as a constrained shortest path problem: theory and computational experience," *OPSEARCH*, vol. 17, pp. 93–109, 1980.

[92] S. Irnich and D. Villeneuve, "The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$," *INFORMS Journal on Computing*, vol. 18, no. 3, pp. 391–406, 2006.

[93] C. Contardo and R. Martinelli, "A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints," *Discrete Optimization*, vol. 12, pp. 129–146, 2014.

[94] M. Poggi and E. Uchoa, "New exact algorithms for the capacitated vehicle routing problem," in *Vehicle Routing: Problems, Methods and Applications*, 2nd ed., P. Toth and D. Vigo, Eds. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014, ch. 3, pp. 59–86.

[95] L. Lozano, D. Duque, and A. L. Medaglia, "An exact algorithm for the elementary shortest path problem with resource constraints," *Transportation Science*, vol. 50, no. 1, pp. 348–357, 2016.

[96] N. Christofides, A. Mingozzi, and P. Toth, "State-space relaxation procedures for the computation of bounds to routing problems," *Networks*, vol. 11, pp. 145–164, 1981.

[97] R. Martinelli, D. Pecin, and M. Poggi, "Efficient elementary and restricted non-elementary route pricing," *European Journal of Operational Research*, vol. 239, no. 1, pp. 102–111, 2014.

[98] G. Righini and M. Salani, "Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints," *Discrete Optimization*, vol. 3, no. 3, pp. 255–273, 2006.

[99] R. Baldacci, N. Christofides, and A. Mingozzi, "An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts," *Mathematical Programming*, vol. 115, no. 2, pp. 351–385, 2008.

[100] C. Tilk, A.-K. Rothenbächer, T. Gschwind, and S. Irnich, "Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster," *European Journal of Operational Research*, vol. 261, no. 2, pp. 530–539, 2017.

[101] M. E. Lübbecke, "Dual variable based fathoming in dynamic programs for column generation," *European Journal of Operational Research*, vol. 162, no. 1, pp. 122–125, 2005.

[102] C. Contardo, J.-F. Cordeau, and B. Gendron, "An exact algorithm based on cut-and-column generation for the capacitated location-routing problem," *INFORMS Journal on Computing*, vol. 26, no. 1, pp. 88–102, 2014.

[103] C. Archetti, M. Bouchard, and G. Desaulniers, "Enhanced branch and price and cut for vehicle routing with split deliveries and time windows," *Transportation Science*, vol. 45, no. 3, pp. 285–298, 2011.

[104] L.-M. Rousseau, M. Gendreau, and G. Pesant, "Solving VRPTWs with constraint programming based column generation," *Annals of Operations Research*, vol. 130, no. 1–4, pp. 199–216, 2004.

[105] D. Feillet, M. Gendreau, and L.-M. Rousseau, "New refinements for the solution of vehicle routing problems with branch and price," *INFOR: Information Systems and Operational Research*, vol. 45, no. 4, pp. 239–256, 2008.

[106] L. Lozano and A. L. Medaglia, "On an exact method for the constrained shortest path problem," *Computers & Operations Research*, vol. 40, no. 1, pp. 378–384, 2013.

[107] G. Desaulniers, D. Pecin, and C. Contardo, "Selective pricing in branch-price-and-cut algorithms for vehicle routing," *EURO Journal on Transportation and Logistics*, vol. 8, no. 2, pp. 147–168, 2019.

[108] M. A. Boschetti, V. Maniezzo, and F. Strappaveccia, "Route relaxations on GPU for vehicle routing problems," *European Journal of Operational Research*, vol. 258, no. 2, pp. 456–466, 2017.

[109] A. N. Letchford and J.-J. Salazar-González, "Projection results for vehicle routing," *Mathematical Programming*, vol. 105, no. 2, pp. 251–274, 2006.

[110] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale travelling-salesman problem," *Journal of the Operations Research Society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[111] D. Naddef and G. Rinaldi, "Branch-and-cut algorithms for the capacitated vrp," in *The vehicle routing problem*, 1st ed., P. Toth and D. Vigo, Eds. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2002, ch. 3, pp. 53–84.

[112] N. Kohl, J. Desrosiers, O. B. G. Madsen, M. M. Solomon, and F. Soumis, "2-path cuts for the vehicle routing problem with time windows," *Transportation Science*, vol. 33, pp. 101–116, 1999.

[113] A. Pessoa, M. Poggi de Aragão, and E. Uchoa, "Robust branch-cut-and-price algorithms for vehicle routing problems," in *The Vehicle Routing Problem: Latest Advances and New Challenges*, B. Golden, , S. Raghavan, , and E. Wasil, Eds. Boston, MA: Springer US, 2008, pp. 297–325.

[114] ——, "A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem," *Networks*, vol. 54, no. 4, pp. 167–177, 2009.

[115] M. Jepsen, B. Petersen, S. Spoorendonk, and D. Pisinger, "Subset-row inequalities applied to the vehicle routing problem with time windows," *Operations Research*, vol. 56, no. 2, pp. 497–511, 2008.

[116] D. G. Pecin, "Exact algorithms for the capacitated vehicle routing problem," Ph.D. dissertation, PUC-Rio, 2014.

[117] D. Pecin, A. Pessoa, M. Poggi, E. Uchoa, and H. Santos, "Limited memory rank-1 cuts for vehicle routing problems," *Operations Research Letters*, vol. 45, no. 3, pp. 206–209, 2017.

[118] B. Petersen, D. Pisinger, and S. 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*, B. Golden, S. Raghavan, and E. Wasil, Eds. Boston, MA: Springer US, 2008, pp. 397–419.

[119] E. Balas, "Some valid inequalities for the set partitioning problem," *Annals of Discrete Mathematics*, vol. 1, pp. 13–47, 1977.

[120] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi, "An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation," *Operations Research*, vol. 52, no. 5, pp. 723–738, 2004.

[121] J. Lysgaard, "CVRPSEP: A package of separation routines for the capacitated vehicle routing problem," Department of Management Science and Logistics, Aarhus School of Business, Tech. Rep., 2003.

[122] E. Balas and M. W. Padberg, "Set partitioning: A survey," *SIAM Review*, vol. 18, no. 4, pp. 710–760, 1976.

[123] S. Spoorendonk and G. Desaulniers, "Clique inequalities applied to the vehicle routing problem with time windows branch-cut-and-price," *INFOR Journal*, vol. 48, no. 1, pp. 53–67, 2010.

[124] P. Augerat, J. Belenguer, E. Benavent, A. Coberan, D. Naddef, and G. Rinaldi, "Computational results with a branch-and-cut code for the capacitated vehicle routing problem," Istituto di Analisi dei Sistemi ed Informatica, Tech. Rep., 1998. [Online]. Available: http://www.iasi.cnr.it/reports{_}html/R495

[125] S. Gélinas, M. Desrochers, J. Desrosiers, and M. M. Solomon, "A new branching strategy for time constrained routing problems with application to backhauling," *Annals of Operations Research*, vol. 61, no. 1, pp. 91–109, 1995.

[126] M. Dell'Amico, G. Righini, and M. Salani, "A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection," *Transportation Science*, vol. 40, no. 2, pp. 235–247, 2006.

[127] C. H. Christiansen and J. Lysgaard, "A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands," *Operations Research Letters*, vol. 35, no. 6, pp. 773–781, 2007.

[128] S. Irnich, "A new branch-and-price algorithm for the traveling tournament problem," *European Journal of Operational Research*, vol. 204, no. 2, pp. 218–228, 2010.

[129] A. Hadjar, O. Marcotte, and F. Soumis, "A branch-and-cut algorithm for the multiple depot vehicle scheduling problem," *Operations Research*, vol. 54, no. 1, pp. 130–149, 2006.

[130] S. Irnich, G. Desaulniers, J. Desrosiers, and A. Hadjar, "Path-reduced costs for eliminating arcs in routing and scheduling," *INFORMS Journal on Computing*, vol. 22, no. 2, pp. 297–313, 2010.

[131] W. E. Walker, "A method for obtaining the optimal dual solution to a linear program using the Dantzig-Wolfe decomposition," *Operations Research*, vol. 17, no. 2, pp. 368–370, 1969.

[132] E. Choi and D. W. Tcha, "A column generation approach to the heterogeneous fleet vehicle routing problem," *Computers & Operations Research*, vol. 34, no. 7, pp. 2080–2095, 2007.

[133] R. Baldacci and A. Mingozzi, "A unified exact method for solving different classes of vehicle routing problems," *Mathematical Programming*, vol. 120, no. 2, pp. 347–380, 2009.

[134] A. Bettinelli, A. Ceselli, and G. Righini, "A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows," *Transportation Research Part C: Emerging Technologies*, vol. 19, no. 5, pp. 723–740, 2011.

[135] S. E. Butt and D. M. Ryan, "An optimal solution procedure for the multiple tour maximum collection problem using column generation," *Computers & Operations Research*, vol. 26, no. 4, pp. 427–441, 1999.

[136] S. Boussier, D. Feillet, and M. Gendreau, "An exact algorithm for team orienteering problems," *4OR*, vol. 5, no. 3, pp. 211–230, 2007.

[137] M. Keshtkaran, K. Ziarati, A. Bettinelli, and D. Vigo, "Enhanced exact solution methods for the team orienteering problem," *International Journal of Production Research*, vol. 54, no. 2, pp. 591–601, 2015.

[138] C. Archetti, D. Feillet, A. Hertz, and M. G. Speranza, "The capacitated team orienteering and profitable tour problems," *Journal of the Operational Research Society*, vol. 60, no. 6, pp. 831–842, 2009.

[139] C. Archetti, N. Bianchessi, and M. G. Speranza, "Optimal solutions for routing problems with profits," *Discrete Applied Mathematics*, vol. 161, no. 4-5, pp. 547–557, 2013.

[140] ——, "The capacitated team orienteering problem with incomplete service," *Optimization Letters*, vol. 7, pp. 1405–1417, 2013.

[141] ——, "The split delivery capacitated team orienteering problem," *Networks*, vol. 63, no. 1, pp. 16–33, 2014.

[142] T. Bulhões, M. H. Hà, R. Martinelli, and T. Vidal, "The vehicle routing problem with service level constraints," *European Journal of Operational Research*, vol. 265, no. 2, pp. 544–558, 2018.

[143] N. Azi, M. Gendreau, and J. Y. Potvin, "An exact algorithm for a vehicle routing problem with time windows and multiple use of vehicles," *European Journal of Operational Research*, vol. 202, no. 3, pp. 756–763, 2010.

[144] C. Archetti, N. Bianchessi, M. G. Speranza, and A. Hertz, "Incomplete service and split deliveries in a routing problem with profits," *Networks*, vol. 63, no. 2, pp. 135–145, 2014.

[145] S. N. Parragh, J. P. Sousa, and B. Almada-Lobo, "The dial-a-ride problem with split requests and profits," *Transportation Science*, vol. 49, no. 2, pp. 311–334, 2015.

[146] Z. Luo, M. Liu, and A. Lim, "A two-phase branch-and-price-and-cut for a dial-a-ride problem in patient transportation," *Transportation Science*, vol. Forthcoming, pp. 1–18, 2018.

[147] G. Gutiérrez-Jarpa, G. Desaulniers, G. Laporte, and V. Marianov, "A branch-and-price algorithm for the vehicle routing problem with deliveries, selective pickups and time windows," *European Journal of Operational Research*, vol. 206, no. 2, pp. 341–349, 2010.

[148] F. Liberatore, G. Righini, and M. Salani, "A column generation algorithm for the vehicle routing problem with soft time windows," *4OR*, vol. 9, no. 1, pp. 49–82, 2011.

[149] A. Bettinelli, A. Ceselli, and G. Righini, "A branch-and-price algorithm for the multi-depot heterogeneous-fleet pickup and delivery problem with soft time windows," *Mathematical Programming Computation*, vol. 6, no. 2, pp. 171–197, 2014.

[150] K. S. Abdallah and J. Jang, "An exact solution for vehicle routing problems with semi-hard resource constraints," *Computers & Industrial Engineering*, vol. 76, pp. 366–377, 2014.

[151] D. Taş, M. Gendreau, N. Dellaert, T. Van Woensel, and A. G. de Kok, "Vehicle routing with soft time windows and stochastic travel times: A column generation and branch-and-price solution approach," *European Journal of Operational Research*, vol. 236, no. 3, pp. 789–799, 2014.

[152] A. Mingozzi, R. Roberti, and P. Toth, "An exact algorithm for the multitrip vehicle routing problem," *INFORMS Journal on Computing*, vol. 25, no. 2, pp. 193–207, 2013.

[153] N. Azi, M. Gendreau, and J. Y. Potvin, "An exact algorithm for a single-vehicle routing problem with time windows and multiple routes," *European Journal of Operational Research*, vol. 178, no. 3, pp. 755–766, 2007.

[154] F. Hernandez, D. Feillet, R. Giroudeau, and O. Naud, "A new exact algorithm to solve the multi-trip vehicle routing problem with time windows and limited duration," *4OR*, vol. 12, no. 3, pp. 235–259, 2014.

[155] ——, "Branch-and-price algorithms for the solution of the multi-trip vehicle routing problem with time windows," *European Journal of Operational Research*, vol. 249, no. 2, pp. 551–559, 2016.

[156] I. Muter, J.-F. Cordeau, and G. Laporte, "A branch-and-price algorithm for the multi-depot vehicle routing problem with interdepot routes," *Transportation Science*, vol. 48, no. 3, pp. 425–441, 2014.

[157] C. Archetti and M. G. Speranza, "Vehicle routing problems with split deliveries," *International Transactions in Operational Research*, vol. 19, no. 1-2, pp. 3–22, 2012.

[158] M. Gendreau, P. Dejax, D. Feillet, and C. Gueguen, "Vehicle routing with time windows and split deliveries," Laboratoire Informatique d'Avignon, Avignon, France., Tech. Rep. 2006–851, 2006.

[159] D. Feillet, P. Dejax, and M. Gendreau, "The profitable arc tour problem: Solution with a branch-and-price algorithm," *Transportation Science*, vol. 39, no. 4, pp. 539–552, 2005.

[160] G. Desaulniers, "Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows," *Operations Research*, vol. 58, no. 1, pp. 179–192, 2010.

[161] M. Salani and I. Vacca, "Branch and price for the vehicle routing problem with discrete split deliveries and time windows," *European Journal of Operational Research*, vol. 213, no. 3, pp. 470–477, 2011.

[162] Z. Luo, H. Qin, W. Zhu, and A. Lim, "Branch and price and cut for the split-delivery vehicle routing problem with time windows and linear weight-related cost," *Transportation Science*, vol. 51, no. 2, pp. 668–687, 2017.

[163] C. Archetti, N. Bianchessi, and M. G. Speranza, "A column generation approach for the split delivery vehicle routing problem," *Networks*, vol. 58, no. 4, pp. 241–254, 2011.

[164] L. Moreno, M. Poggi de Aragão, and E. Uchoa, "Improved lower bounds for the split delivery vehicle routing problem," *Operations Research Letters*, vol. 38, no. 4, pp. 302–306, 2010.

[165] C. Archetti, N. Bianchessi, and M. G. Speranza, "A branch-price-and-cut algorithm for the commodity constrained split delivery vehicle routing problem," *Computers & Operations Research*, vol. 64, pp. 1–10, 2015.

[166] D. M. Ryan and B. A. Foster, "An integer programming approach to scheduling," in *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, A. Wren, Ed. North-Holland, 1981, pp. 269–280.

[167] S. Ichoua, M. Gendreau, and J.-Y. Potvin, "Vehicle dispatching with time-dependent travel times," *European Journal of Operational Research*, vol. 144, no. 2, pp. 379–396, 2003.

[168] S. Dabia, S. Ropke, T. Van Woensel, and T. de Kok, "Branch and price for the time-dependent vehicle routing problem with time windows," *Transportation Science*, vol. 47, no. 3, pp. 380–396, 2013.

[169] I. Kara, B. Kara, and K. Yetis, "Cumulative vehicle routing problems," in *Vehicle Routing Problem*, T. Caric and H. Gold, Eds. Vienna: I-Tech Education and Publishing KG, 2008, ch. 6, pp. 85–98.

[170] J. Lysgaard and S. Wøhlk, "A branch-and-cut-and-price algorithm for the cumulative capacitated vehicle routing problem," *European Journal of Operational Research*, vol. 236, no. 3, pp. 800–810, 2014.

[171] R. Fukasawa, Q. He, and Y. Song, "A branch-cut-and-price algorithm for the energy minimization vehicle routing problem," *Transportation Science*, vol. 50, no. 1, pp. 23–34, 2016.

[172] C. Lin, K. Choy, G. Ho, S. Chung, and H. Lam, "Survey of green vehicle routing problem: Past and future trends," *Expert Systems with Applications*, vol. 41, no. 4, Part 1, pp. 1118–1138, 2014.

[173] S. Pelletier, O. Jabali, and G. Laporte, "50th anniversary invited article - goods distribution with electric vehicles: Review and research perspectives," *Transportation Science*, vol. 50, no. 1, pp. 3–22, 2016.

[174] T. Bektaş and G. Laporte, "The pollution-routing problem," *Transportation Research Part B: Methodological*, vol. 45, no. 8, pp. 1232–1250, 2011, supply chain disruption and risk management.

[175] S. Dabia, E. Demir, and T. Van Woensel, "An exact approach for a variant of the pollution-routing problem," *Transportation Science*, vol. 51, no. 2, pp. 607–628, 2017.

[176] G. Desaulniers, F. Errico, S. Irnich, and M. Schneider, "Exact algorithms for electric vehicle-routing problems with time windows," *Operations Research*, vol. 64, no. 6, pp. 1388–1405, 2016.

[177] J. Andelmin and E. Bartolini, "An exact algorithm for the green vehicle routing problem," *Transportation Science*, vol. 51, no. 4, pp. 1288–1303, 2017.

[178] M. Gendreau, O. Jabali, and W. Rei, "Stochastic vehicle routing problems," in *Vehicle Routing: Problems, Methods and Applications*, 2nd ed., P. Toth and D. Vigo, Eds. Philadelphia, PA: Society for Industrial and Applied Mathematics, 2014, ch. 8, pp. 213–239.

[179] J. Oyola, H. Arntzen, and D. L. Woodruff, "The stochastic vehicle routing problem, a literature review, part I: Models," *EURO Journal on Transportation and Logistics*, vol. Forthcoming, pp. 1–29, 2016.

[180] M. Gendreau, G. Laporte, and R. Séguin, "Stochastic vehicle routing," *European Journal of Operational Research*, vol. 7, no. 1992, pp. 3–12, 1996.

[181] T. Dinh, R. Fukasawa, and J. Luedtke, "Exact algorithms for the chance-constrained vehicle routing problem," *Mathematical Programming*, vol. Forthcoming, pp. 1–34, 2017.

[182] C. Gauvin, G. Desaulniers, and M. Gendreau, "A branch-cut-and-price algorithm for the vehicle routing problem with stochastic demands," *Computers & Operations Research*, vol. 50, pp. 141–153, 2014.

[183] M. Noorizadegan and B. Chen, "Vehicle routing with probabilistic capacity constraints," *European Journal of Operational Research*, vol. 270, no. 2, pp. 544–555, 2018.

[184] F. Errico, G. Desaulniers, G. Gendreau, W. Rei, and L.-M. Roussau, "The vehicle routing problem with hard time windows and stochastic service times," *EURO Journal on Transportation and Logistics*, vol. Forthcoming, pp. 1–29, 2016.

[185] F. Errico, G. Desaulniers, M. Gendreau, W. Rei, and L. M. Rousseau, "A priori optimization with recourse for the vehicle routing problem with hard time windows and stochastic service times," *European Journal of Operational Research*, vol. 249, no. 1, pp. 55–66, 2016.

[186] A. Ben-Tal, L. E. Ghaoui, and A. Nemirovski, *Robust Optimization*. Princeton University Press, 2009.

[187] D. Bertsimas, D. Pachamanova, and M. Sim, "Robust linear optimization under general norms," *Operations Research Letters*, vol. 32, no. 6, pp. 510–516, 2004.

[188] C. Lee, K. Lee, and S. Park, "Robust vehicle routing problem with deadlines and travel time/demand uncertainty," *Journal of the Operational Research Society*, vol. 63, pp. 1294–1306, 2012.

[189] S. Souyris, C. E. Cortés, F. Ordóñez, and A. Weintraub, "A robust optimization approach to dispatching technicians under stochastic service times," *Optimization Letters*, vol. 7, no. 7, pp. 1549–1568, 2013.

[190] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte, "Static pickup and delivery problems: A classification scheme and survey," *TOP*, vol. 15, no. 1, pp. 1–31, 2007.

[191] T. Gschwind, S. Irnich, A.-K. Rothenbächer, and C. Tilk, "Bidirectional labeling in column-generation algorithms for pickup-and-delivery problems," *European Journal of Operational Research*, vol. 266, no. 1, pp. 521–530, 2018.

[192] Y. Dumas, J. Desrosiers, and F. Soumis, "The pickup and delivery problem with time windows," *European Journal of Operational Research*, vol. 54, no. 1, pp. 7–22, 1991.

[193] M. Savelsbergh and M. Sol, "Drive: Dynamic routing of independent vehicles," *Operations Research*, vol. 46, no. 4, pp. 474–490, 1998.

[194] S. Ropke and J.-F. Cordeau, "Branch and cut and price for the pickup and delivery problem with time windows," *Transportation Science*, vol. 43, no. 3, pp. 267–286, 2009.

[195] R. Baldacci, E. Bartolini, and A. Mingozzi, "An exact algorithm for the pickup and delivery problem with time windows," *Operations Research*, vol. 59, no. 2, pp. 414–426, 2011.

[196] J.-F. Cordeau and G. Laporte, "The dial-a-ride problem: models and algorithms," *Annals of Operations Research*, vol. 153, no. 1, pp. 29–46, Sep 2007.

[197] S. Ropke, "Heuristic and exact algorithms for vehicle routing problems," Ph.D. dissertation, University of Copenhagen, 2005. [Online]. Available: http://www.diku.dk/{~}sropke/Papers/PHDThesis.pdf

[198] T. Gschwind and S. Irnich, "Effective handling of dynamic time windows and its application to solving the dial-a-ride problem," *Transportation Science*, vol. 49, no. 2, pp. 335–354, 2015.

[199] S. Ropke, J.-F. Cordeau, and G. Laporte, "Models and branch-and-cut algorithms for pickup and delivery problems with time windows," *Networks*, vol. 49, no. 4, pp. 258–272, 2007.

[200] J.-F. Cordeau, "A branch-and-cut algorithm for the dial-a-ride problem," *Operations Research*, vol. 54, no. 3, pp. 573–586, 2006.

[201] T. Gschwind, "A comparison of column-generation approaches to the synchronized pickup and delivery problem," *European Journal of Operational Research*, vol. 247, no. 1, pp. 60–71, 2015.

[202] M. Drexl, "Synchronization in vehicle routing - a survey of vrps with multiple synchronization constraints," *Transportation Science*, vol. 46, no. 3, pp. 297–316, 2012.

[203] Y. Qu and J. F. Bard, "A branch-and-price-and-cut algorithm for heterogeneous pickup and delivery problems with configurable vehicle capacity," *Transportation Science*, vol. 49, no. 2, pp. 254–270, 2015.

[204] R. Masson, S. Ropke, F. Lehuédé, and O. Péton, "A branch-and-cut-and-price approach for the pickup and delivery problem with shuttle routes," *European Journal of Operational Research*, vol. 236, no. 3, pp. 849–862, 2013.

[205] V. Ghilas, J.-F. Cordeau, E. Demir, and T. V. Woensel, "Branch-and-price for the pickup and delivery problem with time windows and scheduled lines," *Transportation Science*, vol. 52, no. 5, pp. 1191–1210, 2018.

[206] M. Cherkesly, G. Desaulniers, and G. Laporte, "Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading," *Transportation Science*, vol. 49, no. 4, pp. 752–766, 2015.

[207] M. Cherkesly, G. Desaulniers, S. Irnich, and G. Laporte, "Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and multiple stacks," *European Journal of Operational Research*, vol. 250, pp. 782–793, 2016.

[208] J.-F. Côté, C. Archetti, M. G. Speranza, M. Gendreau, and J.-Y. Potvin, "A branch-and-cut algorithm for the pickup and delivery traveling salesman problem with multiple stacks," *Networks*, vol. 60, no. 4, pp. 212–226, 2012.

[209] M. Iori and J. Riera-Ledesma, "Exact algorithms for the double vehicle routing problem with multiple stacks," *Computers & Operations Research*, vol. 63, pp. 83–101, 2015.

[210] M. Veenstra, M. Cherkesly, G. Desaulniers, and G. Laporte, "The pickup and delivery problem with time windows and handling operations," *Computers & Operations Research*, vol. 77, pp. 127–140, 2017.

[211] E. Angelelli and R. Mansini, "The vehicle routing problem with time windows and simultaneous pick-up and delivery," in *Quantitative Approaches to Distribution Logistics*

and Supply Chain Management, A. Klose, M. G. Speranza, and L. N. Van Wassenhove, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2002, pp. 249–267.

[212] A. Subramanian, E. Uchoa, A. A. Pessoa, and L. S. Ochi, "Branch-cut-and-price for the vehicle routing problem with simultaneous pickup and delivery," *Optimization Letters*, vol. 7, no. 7, pp. 1569–1581, 2013.

[213] L. Costa, C. Contardo, G. Desaulniers, and D. Pecin, "Selective arc-ng pricing for vehicle routing," GERAD, Montréal, Research report G-2020-07, Les Cahiers du GERAD, HEC Montréal, 2020.

[214] M. M. Solomon, "Algorithms for the vehicle routing and scheduling problems with time window constraints," *Operations Research*, vol. 35, no. 2, pp. 254–265, 1987.

[215] H. Gehring and J. Homberger, "A parallel two-phase metaheuristic for routing problems with time windows," *Asia-Pacific Journal of Operational Research*, vol. 18, no. 1, p. 35, 2001.

[216] R. Sadykov, E. Uchoa, and A. A. Pessoa, "A bucket graph based labeling algorithm with application to vehicle routing," Universidade Federal Fluminense, Research Report Cadernos do LOGIS 2017/7, 2017.

[217] E. D. Dolan and J. J. More, "Benchmarking optimization software with performance profiles," *Mathematical Programming A*, vol. 91, pp. 201–213, 2002.

[218] M. Deveci and N. Ç. Demirel, "A survey of the literature on airline crew scheduling," *Engineering Applications of Artificial Intelligence*, vol. 74, pp. 54–69, 2018.

[219] F. Glover, "Surrogate constraints," *Operations Research*, vol. 16, no. 4, pp. 741–749, 1968.

[220] ——, "Tutorial on surrogate constraint approaches for optimization in graphs," *Journal of Heuristics*, vol. 9, no. 3, pp. 175–227, 2003.

[221] L. Pishchulin, E. Insafutdinov, S. Tang, B. Andres, M. Andriluka, P. V. Gehler, and B. Schiele, "Deepcut: Joint subset partition and labeling for multi person pose estimation," in *Proc. 22nd Conference on Computer Vision and Pattern Recognition*, Las Vegas, Nevada, 2016, pp. 4929–4937.

[222] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele, "Deepercut: A deeper, stronger, and faster multi-person pose estimation model," in *Proc. 14th*

European Conference on Computer Vision, Amsterdam, The Netherlands, 2016, pp. 34–50.

[223] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *Proc. 30th Conference on Computer Vision and Pattern Recognition*, Anchorage, Alaska, 2008, pp. 1–8.

[224] Y. Yang and D. Ramanan, "Articulated pose estimation with flexible mixtures-of-parts," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE, 2011, pp. 1385–1392.

[225] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[226] M. Andriluka, L. Pishchulin, P. Gehler, and S. B, "2d human pose estimation: New benchmark and state of the art analysis," in *Proc. 27th Conference on Computer Vision and Pattern Recognition*, Columbus, Ohio, 2014, pp. 3686–3693.

[227] A. E. Fernandes Muritiba, M. Iori, E. Malaguti, and P. Toth, "Algorithms for the bin packing problem with conflicts," *INFORMS Journal on Computing*, vol. 22, no. 3, pp. 401–415, 2010.

[228] S. Elhedhli, L. Li, M. Gzara, and J. Naoum-Sawaya, "A branch-and-price algorithm for the bin packing problem with conflicts," *INFORMS Journal on Computing*, vol. 23, no. 3, pp. 404–415, 2011.

[229] M. Gendreau, G. Laporte, and F. Semet, "Heuristics and lower bounds for the bin packing problem with conflicts," *Computers & Operations Research*, vol. 31, no. 3, pp. 347–358, 2004.

[230] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," *Journal of Heuristics*, vol. 2, pp. 5–30, 1996.

APPENDIX A DETAILED RESULTS – SELECTIVE ARC-NG PRICING FOR VEHICLE ROUTING

In this section, we report detailed results for all instances solved by settings `Default`, `SetBased`, `Pairwise`, and `SetPair`. We provide pairwise comparisons between the `Default` algorithm and each of the selective settings. Sections A and A present results for S and GH instances, respectively. For each instance, we report: the lower bound (**lb**) achieved, the number of column generation iterations performed by each algorithm (**#iters**), the CPU time in seconds (**T(s)**), the average number of DSSR iterations per column generation iteration performed by each algorithm (**#DSSR**), and the average number of labels generated per iteration (**#labels**). When comparing the performance of two algorithms, we highlight in boldface the best results for each indicator. Line **#Best** shows the total number of instances for which each algorithm was capable of achieving the best values. Notice that, in the case where both algorithms produce equal results for a given instance, this instance is counted for both algorithms.

Solomon's instances

Table A.1 Detailed results for comparing `Default` vs `SetBased` settings using `only-cycle` DSSR before adding non-robust cuts

Instance	Default					SetBased				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	5,887.0	4,454	800	**2.1**	13,516	5,887.0	**2,841**	**517**	2.3	**12,527**
C204	5,881.0	25,207	1,239	**2.4**	75,340	5,881.0	**14,604**	**780**	2.7	**65,135**
R202	10,164.8	290	**130**	4.2	10,223	10,164.8	**268**	136	**3.6**	**8,169**
R203	8,597.0	785	169	5.1	31,121	8,597.0	**615**	**131**	5.1	**27,872**
R204	7,201.6	2,434	175	6.9	121,553	7,201.6	**1,704**	**171**	**5.3**	**78,889**
R206	8,600.1	582	**103**	7.6	38,749	8,600.1	**551**	113	**6.0**	**30,732**
R207	7,805.0	1,152	152	**5.9**	54,037	7,805.0	**881**	**117**	6.1	**52,027**
R208	6,910.9	7,539	**234**	8.7	291,927	6,910.9	**6,059**	250	**7.0**	**204,117**
R209	8,371.8	340	**81**	7.2	29,284	8,371.8	**303**	86	**5.8**	**22,617**
R210	8,813.2	463	93	8.5	34,247	8,813.2	**398**	**82**	**8.0**	**32,613**
R211	7,311.0	641	115	7.8	75,489	7,311.0	**503**	115	**6.2**	**54,027**
RC204	7,764.9	17,166	269	11.9	789,806	7,764.9	**11,096**	**256**	**10.8**	**463,053**
RC207	9,415.8	808	112	**14.3**	84,070	9,415.8	**630**	**94**	14.8	**73,374**
RC208	7,636.4	3,071	148	22.0	632,136	7,636.4	**2,472**	**139**	**21.9**	**493,415**
#Best	14/14	0/14	5/14	5/14	0/14	14/14	14/14	10/14	10/14	14/14

Table A.2 Detailed results for comparing `Default` vs `SetBased` settings using `only-cycle` DSSR after adding non-robust cuts

Instance	Default					SetBased				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	5,887.0	4,454	800	**2.1**	13,516	5,887.0	**2,842**	**517**	2.3	**12,527**
C204	5,881.0	25,207	1,239	**2.4**	75,340	5,881.0	**14,605**	**780**	2.7	**65,135**
R202	**10,296.0**	796	736	1.7	**3,788**	10,288.2	**615**	**647**	1.6	3,826
R203	8,708.0	1,239	725	2.1	11,040	8,708.0	**998**	**554**	2.1	**10,953**
R204	7,313.0	5,352	**638**	2.9	73,417	7,313.0	**4,442**	645	**2.5**	**58,024**
R206	8,759.0	1,435	519	2.6	18,251	8,759.0	**1,362**	**515**	**2.4**	**17,473**
R207	7,940.0	3,034	654	2.4	31,611	7,940.0	**2,219**	**523**	2.5	**29,865**
R208	6,994.1	**196,397**	**594**	5.3	1,125,610	**7,000.3**	234,324	710	**4.3**	**1,124,685**
R209	8,548.0	**2,071**	**580**	2.4	24,821	8,548.0	2,095	591	**2.3**	**24,635**
R210	9,005.0	4,354	674	2.3	**30,501**	9,005.0	**3,812**	**635**	2.3	30,887
R211	7,467.0	**6,785**	**357**	4.2	**138,224**	7,467.0	8,549	380	**3.8**	149,502
RC204	7,835.0	18,715	546	6.5	413,895	7,835.0	**12,956**	**535**	**5.9**	**249,233**
RC207	9,629.0	1,683	459	4.6	34,981	9,629.0	**1,628**	**457**	**4.3**	**31,885**
RC208	7,761.0	4,626	373	10.0	347,683	7,761.0	**4,107**	**370**	**9.6**	**283,936**
#Best	13/14	3/14	4/14	6/14	3/14	13/14	11/14	10/14	12/14	11/14

Table A.3 Detailed results for comparing `Default` vs `Pairwise` settings using `only-cycle` DSSR before adding non-robust cuts

Instance	Default					Pairwise				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	5,887.0	4,454	800	**2.1**	**13,516**	5,887.0	**2,504**	**360**	3.0	15,993
C204	5,881.0	25,207	1,239	2.4	75,340	5,881.0	**17,959**	**981**	2.4	**62,472**
R202	10,164.8	290	130	4.2	10,223	10,164.8	**271**	130	**3.9**	**7,618**
R203	8,597.0	785	169	5.1	31,121	**8,598.1**	**606**	**168**	**3.9**	**21,073**
R204	7,201.6	2,434	175	6.9	121,553	7,201.6	**1,791**	**174**	**5.4**	**82,007**
R206	8,600.1	582	**103**	7.6	38,749	8,600.1	**531**	109	**6.3**	**28,842**
R207	7,805.0	1,152	152	**5.9**	54,037	7,805.0	**907**	**117**	6.3	**53,843**
R208	6,910.9	7,539	234	8.7	291,927	6,910.9	**5,204**	**213**	**8.0**	**206,766**
R209	8,371.8	340	81	7.2	29,284	8,371.8	**300**	**73**	**6.8**	**25,270**
R210	8,813.2	463	93	**8.5**	34,247	**8,813.4**	**375**	**73**	8.7	**33,084**
R211	7,311.0	641	115	7.8	75,489	**7,311.1**	**529**	**105**	**7.1**	**62,460**
RC204	7,764.9	17,166	**269**	11.9	789,806	7,764.9	**10,507**	279	**10.1**	**425,550**
RC207	9,415.8	808	112	14.3	84,070	9,415.8	**620**	**104**	**13.0**	**62,470**
RC208	7,636.4	3,071	148	22.0	632,136	7,636.4	**2,425**	**141**	**21.1**	**470,504**
#Best	11/14	0/14	3/14	4/14	1/14	14/14	14/14	12/14	11/14	13/14

Table A.4 Detailed results for comparing `Default` vs `Pairwise` settings using `only-cycle` DSSR after adding non-robust cuts

Instance	Default					Pairwise				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	5,887.0	4,454	800	**2.1**	**13,516**	5,887.0	**2,505**	**360**	3.0	15,993
C204	5,881.0	25,207	1,239	2.4	75,340	5,881.0	**17,960**	**981**	2.4	**62,472**
R202	**10,296.0**	796	736	1.7	3,788	10,292.2	**641**	**632**	1.7	**3,548**
R203	8,708.0	1,239	725	2.1	11,040	8,708.0	**1,011**	**652**	**1.8**	**9,182**
R204	7,313.0	5,352	**638**	2.9	73,417	7,313.0	**4,789**	661	**2.6**	**58,075**
R206	8,759.0	**1,435**	519	2.6	**18,251**	8,759.0	1,441	**503**	**2.5**	18,458
R207	7,940.0	3,034	654	2.4	31,611	7,940.0	**2,682**	**614**	2.3	**30,571**
R208	6,994.1	196,397	**594**	5.3	1,125,610	**6,998.3**	**186,267**	636	**4.7**	**1,027,970**
R209	8,548.0	2,071	**580**	2.4	24,821	8,548.0	**2,067**	584	**2.3**	**22,306**
R210	9,005.0	**4,354**	674	2.3	**30,501**	9,005.0	4,504	**612**	2.3	32,925
R211	7,467.0	**6,785**	357	4.2	**138,224**	7,467.0	9,054	**355**	4.2	160,883
RC204	7,835.0	18,715	**546**	6.5	413,895	7,835.0	**12,217**	570	**5.6**	**233,377**
RC207	9,629.0	1,683	459	4.6	34,981	9,629.0	**1,622**	**444**	**4.3**	**29,640**
RC208	7,761.0	4,626	**373**	10.0	347,683	7,761.0	**3,940**	374	**9.3**	**262,534**
#Best	13/14	3/14	5/14	6/14	4/14	13/14	11/14	9/14	13/14	10/14

Table A.5 Detailed results for comparing `Default` vs `SetPair` settings using `only-cycle` DSSR before adding non-robust cuts

Instance	Default					SetPair				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	5,887.0	4,454	800	2.1	13,516	5,887.0	**2,782**	**529**	**2.0**	**12,556**
C204	5,881.0	25,207	1,239	**2.4**	75,340	5,881.0	**15,712**	**722**	2.8	**75,129**
R202	10,164.8	290	130	4.2	10,223	10,164.8	**264**	**120**	4.2	**8,734**
R203	8,597.0	785	169	5.1	31,121	**8,597.9**	**631**	**151**	**4.4**	**24,619**
R204	7,201.6	2,434	175	6.9	121,553	7,201.6	**1,662**	**157**	**5.8**	**85,759**
R206	8,600.1	582	103	7.6	38,749	8,600.1	**515**	**91**	**7.2**	**37,482**
R207	7,805.0	1,152	152	**5.9**	54,037	**7,806.1**	**874**	**116**	6.2	**51,982**
R208	6,910.9	7,539	234	8.7	291,927	6,910.9	**5,928**	**218**	**8.1**	**230,242**
R209	8,371.8	340	81	7.2	29,284	8,371.8	**301**	**79**	**6.6**	**24,265**
R210	8,813.2	463	93	**8.5**	**34,247**	8,813.2	**407**	**72**	9.3	34,768
R211	7,311.0	641	115	7.8	75,489	**7,311.5**	**491**	**107**	**6.7**	**57,156**
RC204	7,764.9	17,166	**269**	11.9	789,806	7,764.9	**11,550**	277	**10.0**	**443,300**
RC207	9,415.8	808	112	14.3	84,070	9,415.8	**655**	**102**	**13.6**	**64,651**
RC208	7,636.4	3,071	148	22.0	632,136	7,636.4	**2,476**	**140**	**21.3**	**477,627**
#Best	11/14	0/14	1/14	4/14	1/14	14/14	14/14	13/14	11/14	13/14

Table A.6 Detailed results for comparing `Default` vs `SetPair` settings using `only-cycle` DSSR after adding non-robust cuts

Instance	Default					SetPair				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	5,887.0	4,454	800	2.1	13,516	5,887.0	**2,783**	**529**	**2.0**	**12,556**
C204	5,881.0	25,207	1,239	**2.4**	75,340	5,881.0	**15,712**	**722**	2.8	**75,129**
R202	**10,296.0**	796	736	**1.7**	3,788	10,292.3	**631**	**592**	1.8	**3,602**
R203	8,708.0	1,239	725	2.1	11,040	8,708.0	**1,053**	**663**	**1.9**	**9,486**
R204	7,313.0	**5,352**	**638**	2.9	73,417	7,313.0	6,055	737	**2.4**	**60,685**
R206	8,759.0	**1,435**	519	2.6	**18,251**	8,759.0	1,541	**503**	**2.4**	18,905
R207	7,940.0	3,034	654	2.4	31,611	7,940.0	**2,374**	**579**	2.3	**27,824**
R208	**6,994.1**	**196,397**	594	5.3	1,125,610	6,989.0	199,145	**574**	**4.9**	**897,083**
R209	8,548.0	**2,071**	**580**	2.4	24,821	8,548.0	2,335	616	**2.2**	**22,051**
R210	9,005.0	**4,354**	674	2.3	30,501	9,005.0	4,487	**631**	2.3	**27,684**
R211	7,467.0	**6,785**	**357**	4.2	**138,224**	7,467.0	10,761	385	**3.7**	140,496
RC204	7,835.0	18,715	**546**	6.5	413,895	7,835.0	**13,344**	558	**5.7**	**244,941**
RC207	9,629.0	**1,683**	459	4.6	34,981	9,629.0	1,810	**442**	**4.4**	**31,714**
RC208	7,761.0	4,626	373	10.0	347,683	7,761.0	**4,172**	**366**	**9.5**	**269,875**
#Best	14/14	7/14	4/14	4/14	2/14	12/14	7/14	10/14	12/14	12/14

Table A.7 Detailed results for comparing `Default` vs `SetBased` settings using `all-arcs` DSSR before adding non-robust cuts

Instance	Default					SetBased				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	5,887.0	1,416	**247**	2.2	12,776	5,887.0	**1,364**	324	**1.7**	**8,603**
C204	5,881.0	**8,871**	**552**	1.8	59,903	5,881.0	21,033	1,462	**1.3**	**51,157**
R202	10,164.8	228	116	**3.2**	**7,485**	10,164.8	**205**	**96**	3.4	7,641
R203	8,597.0	509	140	3.8	22,172	8,597.0	**454**	**118**	3.8	**22,035**
R204	7,202.0	1,221	147	4.0	64,842	7,202.0	**1,075**	147	**3.6**	**58,091**
R206	8,600.1	390	99	4.4	22,766	8,600.1	**342**	**91**	4.4	**21,152**
R207	7,805.0	**686**	**120**	**4.1**	**37,176**	**7,851.4**	851	134	4.8	50,241
R208	6,910.9	3,379	**168**	4.5	178,741	6,910.9	**2,909**	177	**3.9**	**131,674**
R209	8,371.8	260	90	**4.2**	**16,451**	8,371.8	**242**	**79**	4.4	17,052
R210	8,813.2	315	82	**5.6**	**24,038**	**8,813.4**	**300**	**73**	6.0	25,711
R211	7,311.0	**343**	**85**	5.2	48,010	7,311.0	365	97	**4.8**	**44,054**
RC204	7,764.9	4,193	**178**	4.6	237,054	7,764.9	**3,474**	212	**3.6**	**163,647**
RC207	9,415.8	338	90	**6.6**	**34,715**	9,415.8	**319**	**81**	7.0	37,632
RC208	7,636.4	904	111	**8.1**	218,110	7,636.4	**786**	**95**	8.5	**203,826**
#Best	12/14	3/14	7/14	8/14	5/14	14/14	11/14	8/14	8/14	9/14

Table A.8 Detailed results for comparing `Default` vs `SetBased` settings using `all-arcs` DSSR after adding non-robust cuts

Instance	Default					SetBased				
	lb	T(s)	#iters	#DSSR	#labels	lb	T(s)	#iters	#DSSR	#labels
C203	5,887.0	1,417	**247**	2.2	12,776	5,887.0	**1,364**	324	**1.7**	**8,603**
C204	5,881.0	**8,871**	**552**	1.8	59,903	5,881.0	21,033	1,462	**1.3**	**51,157**
R202	10,283.2	**488**	**554**	1.5	3,370	**10,289.2**	586	685	**1.4**	**3,056**
R203	8,708.0	856	574	1.7	9,160	8,708.0	**828**	**568**	1.6	**8,764**
R204	7,313.0	3,653	566	1.9	**49,541**	7,313.0	**3,434**	**559**	**1.7**	50,281
R206	8,759.0	1,056	**429**	1.9	14,763	8,759.0	**995**	436	1.8	**14,239**
R207	7,940.0	1,618	481	**1.8**	**21,936**	7,940.0	**1,324**	**299**	2.9	34,890
R208	7,002.0	203,902	**480**	2.4	806,409	**7,005.1**	**187,043**	503	**2.2**	**782,421**
R209	8,548.0	1,784	541	1.7	18,585	8,548.0	**1,631**	**534**	1.7	**18,336**
R210	9,005.0	4,918	**598**	1.7	29,395	9,005.0	**4,244**	620	1.7	**27,285**
R211	7,467.0	5,820	**295**	2.6	95,989	7,467.0	**4,020**	305	2.6	**82,622**
RC204	7,835.0	5,498	**420**	2.6	128,015	7,835.0	**4,953**	488	**2.2**	**95,283**
RC207	9,629.0	1,084	**399**	2.4	**19,037**	9,629.0	**965**	411	2.4	19,383
RC208	7,761.0	1,961	314	3.6	133,620	7,761.0	**1,746**	**308**	**3.4**	**121,038**
#Best	12/14	2/14	9/14	7/14	3/14	14/14	12/14	5/14	13/14	11/14

Table A.9 Detailed results for comparing `Default` vs `Pairwise` settings using `all-arcs` DSSR before adding non-robust cuts

Instance	Default					Pairwise				
	lb	T(s)	#iters	#DSSR	#labels	lb	T(s)	#iters	#DSSR	#labels
C203	5,887.0	**1,416**	**247**	2.2	12,776	5,887.0	1,518	309	**1.8**	**10,577**
C204	5,881.0	8,871	**552**	1.8	59,903	5,881.0	**8,276**	595	**1.6**	**50,065**
R202	10,164.8	228	116	**3.2**	7,485	10,164.8	**211**	**98**	3.3	**6,732**
R203	8,597.0	509	**140**	3.8	22,172	8,597.0	**502**	147	**3.3**	**18,966**
R204	**7,202.0**	1,221	147	4.0	64,842	7,201.7	**1,065**	**142**	**3.8**	**60,584**
R206	8,600.1	390	99	**4.4**	22,766	8,600.1	**339**	**83**	4.5	**21,907**
R207	7,805.0	**686**	**120**	**4.1**	**37,176**	**7,851.4**	933	145	4.7	51,794
R208	6,910.9	3,379	168	4.5	178,741	6,910.9	**2,794**	**157**	**4.3**	**142,616**
R209	8,371.8	260	90	**4.2**	**16,451**	8,371.8	**229**	**69**	4.9	18,478
R210	8,813.2	315	82	5.6	24,038	8,813.2	**298**	**77**	**5.5**	**21,949**
R211	7,311.0	**343**	**85**	5.2	48,010	7,311.0	348	89	**5.0**	**44,801**
RC204	7,764.9	4,193	**178**	4.6	237,054	7,764.9	**3,351**	181	**4.4**	**178,990**
RC207	9,415.8	338	90	**6.6**	**34,715**	9,415.8	**313**	**71**	7.9	39,903
RC208	7,636.4	904	111	8.1	218,110	7,636.4	**826**	**101**	8.1	**191,750**
#Best	13/14	3/14	6/14	6/14	3/14	13/14	11/14	8/14	9/14	11/14

Table A.10 Detailed results for comparing `Default` vs `Pairwise` settings using `all-arcs` DSSR after adding non-robust cuts

Instance	Default					Pairwise				
	lb	T(s)	#iters	#DSSR	#labels	lb	T(s)	#iters	#DSSR	#labels
C203	5,887.0	**1,417**	**247**	2.2	12,776	5,887.0	1,518	309	**1.8**	**10,577**
C204	5,881.0	8,871	**552**	1.8	59,903	5,881.0	**8,276**	595	**1.6**	**50,065**
R202	**10,283.2**	488	554	**1.5**	3,370	10,276.6	**397**	**408**	1.6	**3,103**
R203	8,708.0	**856**	**574**	1.7	9,160	8,708.0	920	701	**1.5**	**7,577**
R204	7,313.0	3,653	566	1.9	**49,541**	7,313.0	**3,336**	**495**	1.9	51,287
R206	8,759.0	1,056	**429**	1.9	14,763	8,759.0	**943**	430	**1.7**	**12,851**
R207	7,940.0	1,618	481	**1.8**	**21,936**	7,940.0	**1,483**	**325**	2.9	37,612
R208	7,002.0	203,902	**480**	2.4	**806,409**	**7,006.0**	**194,571**	492	**2.3**	839,060
R209	8,548.0	**1,784**	541	1.7	18,585	8,548.0	2,035	**530**	1.7	**18,008**
R210	9,005.0	4,918	598	1.7	29,395	9,005.0	**3,394**	**561**	1.7	**25,382**
R211	7,467.0	5,820	295	2.6	95,989	7,467.0	**4,912**	295	2.6	**86,102**
RC204	7,835.0	5,498	**420**	2.6	128,015	7,835.0	**4,683**	437	**2.4**	**96,170**
RC207	9,629.0	**1,084**	399	2.4	**19,037**	9,629.0	1,096	**394**	2.4	19,045
RC208	7,761.0	1,961	**314**	3.6	133,620	7,761.0	**1,766**	315	**3.4**	**116,031**
#Best	13/14	4/14	8/14	7/14	4/14	13/14	10/14	7/14	12/14	10/14

Table A.11 Detailed results for comparing `Default` vs `SetPair` settings using `all-arcs` DSSR before adding non-robust cuts

Instance	Default					SetPair				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	5,887.0	**1,416**	**247**	2.2	12,776	5,887.0	1,620	395	**1.5**	**9,900**
C204	5,881.0	**8,871**	**552**	1.8	59,903	5,881.0	14,809	1,073	**1.4**	**48,300**
R202	10,164.8	228	116	**3.2**	7,485	10,164.8	**208**	**101**	3.4	**7,441**
R203	8,597.0	509	140	3.8	22,172	8,597.0	**479**	**122**	3.7	**21,951**
R204	7,202.0	1,221	147	4.0	64,842	7,202.0	**1,104**	**142**	**3.8**	**60,708**
R206	8,600.1	390	99	4.4	22,766	8,600.1	**375**	99	**4.3**	**22,188**
R207	7,805.0	**686**	**120**	**4.1**	**37,176**	**7,851.4**	1,023	154	4.6	53,288
R208	6,910.9	3,379	**168**	4.5	178,741	6,910.9	**2,889**	184	**3.8**	**130,503**
R209	8,371.8	260	90	**4.2**	**16,451**	8,371.8	**227**	**75**	4.5	16,458
R210	8,813.2	315	82	**5.6**	**24,038**	8,813.2	**296**	**70**	6.1	24,779
R211	7,311.0	**343**	**85**	5.2	48,010	7,311.0	357	101	**4.5**	**40,691**
RC204	7,764.9	4,193	**178**	4.6	237,054	7,764.9	**3,212**	185	**4.3**	**163,170**
RC207	9,415.8	338	90	6.6	34,715	9,415.8	**331**	**87**	6.6	**33,335**
RC208	7,636.4	904	**111**	8.1	218,110	7,636.4	**814**	113	**7.2**	**169,089**
#Best	13/14	4/14	8/14	6/14	3/14	14/14	10/14	7/14	10/14	11/14

Table A.12 Detailed results for comparing `Default` vs `SetPair` settings using `all-arcs` DSSR after adding non-robust cuts

Instance	Default					SetPair				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C203	5,887.0	**1,417**	**247**	2.2	12,776	5,887.0	1,620	395	**1.5**	**9,900**
C204	5,881.0	**8,871**	**552**	1.8	59,903	5,881.0	14,810	1,073	**1.4**	**48,300**
R202	**10,283.2**	488	554	1.5	3,370	10,281.9	**479**	**499**	1.5	**3,325**
R203	8,708.0	856	574	1.7	9,160	8,708.0	**832**	**563**	1.6	**8,416**
R204	7,313.0	3,653	566	1.9	49,541	7,313.0	**3,585**	**520**	1.9	**46,253**
R206	8,759.0	1,056	429	1.9	14,763	8,759.0	**1,026**	**421**	1.9	**14,288**
R207	7,940.0	1,618	481	**1.8**	**21,936**	7,940.0	**1,549**	**322**	2.9	37,969
R208	**7,002.0**	203,902	480	2.4	806,409	6,998.6	**197,787**	**466**	2.3	**691,352**
R209	8,548.0	1,784	**541**	1.7	18,585	8,548.0	**1,739**	557	1.6	**17,125**
R210	9,005.0	**4,918**	**598**	1.7	29,395	9,005.0	6,411	599	1.7	**28,997**
R211	7,467.0	5,820	295	2.6	95,989	7,467.0	**5,707**	**292**	2.6	**87,119**
RC204	7,835.0	5,498	**420**	2.6	128,015	7,835.0	**4,857**	471	**2.3**	**90,537**
RC207	9,629.0	**1,084**	**399**	2.4	19,037	9,629.0	1,131	403	2.3	**17,888**
RC208	7,761.0	1,961	**314**	3.6	133,620	7,761.0	**1,852**	315	**3.3**	**112,734**
#Best	14/14	4/14	7/14	10/14	1/14	12/14	10/14	7/14	13/14	13/14

Gehring and Homberger's instances

Table A.13 Detailed results for comparing `Default` vs `SetBased` settings using `only-cycle` DSSR before adding non-robust cuts

Instance	Default					SetBased				
	lb	T(s)	#iters	#DSSR	#labels	lb	T(s)	#iters	#DSSR	#labels
C1_2_10	**26,197.3**	655	**72**	29.3	560,811	26,196.9	**602**	83	**22.8**	**454,343**
C1_2_3	26,597.4	476	**132**	12.4	193,347	26,597.4	**403**	145	**9.8**	**147,833**
C1_2_4	26,197.7	1,066	220	13.9	400,174	26,197.7	**885**	**209**	**12.3**	**325,856**
C1_2_9	26,396.0	219	**58**	13.6	124,522	26,396.0	**211**	59	**11.7**	**106,689**
C2_2_1	19,150.3	**349**	8	1.0	388	19,150.3	355	8	1.0	388
C2_2_2	18,396.0	**9,504**	**794**	2.5	10,412	18,396.0	9,856	919	**1.9**	**7,868**
C2_2_5	18,607.1	1,339	165	3.4	**3,829**	18,607.1	**1,300**	**150**	3.5	4,049
C2_2_6	18,406.7	4,173	324	5.3	15,499	18,406.7	**3,429**	**302**	**4.6**	**13,644**
C2_2_7	18,356.4	6,342	**429**	5.3	22,082	18,356.4	**6,322**	454	**4.9**	**20,152**
C2_2_8	18,082.4	7,977	363	8.0	42,366	18,082.4	**6,458**	**296**	8.0	**40,892**
C2_2_9	18,011.7	13,272	486	**6.7**	56,421	**18,011.9**	**9,980**	**366**	7.2	**54,808**
R1_2_10	32,450.2	153	**26**	16.3	210,725	**32,451.2**	**134**	27	**13.9**	**177,834**
R1_2_4	**30,010.4**	270	43	**13.2**	**357,645**	30,009.9	**215**	**40**	13.4	364,716
R1_2_5	40,068.1	42	12	**2.0**	**9,881**	40,068.1	**41**	**11**	2.2	10,827
R1_2_6	34,930.6	101	41	4.6	**53,930**	34,930.6	**92**	**37**	**4.4**	54,508
R1_2_7	31,000.6	237	51	8.7	190,129	31,000.6	**190**	51	**7.9**	**170,515**
R1_2_8	29,032.2	356	**34**	19.8	589,966	**29,032.3**	**261**	36	**17.0**	**487,423**
R1_2_9	36,788.3	68	**17**	7.4	52,206	36,788.3	**64**	20	**6.2**	**44,247**
R2_2_1	34,625.1	**1,645**	**88**	2.7	1,232	34,625.1	1,734	110	**2.4**	**1,097**
R2_2_2	30,006.1	5,397	282	5.1	12,466	30,006.1	**4,943**	**270**	**4.6**	**11,252**
R2_2_5	30,290.3	2,946	188	**3.9**	**4,670**	30,290.3	**2,798**	**144**	4.2	4,843
R2_2_6	26,455.6	9,738	414	5.9	39,427	26,455.6	**8,397**	**392**	**5.2**	**30,201**
R2_2_9	28,206.0	4,217	252	4.7	11,431	28,206.0	**3,836**	**207**	**4.5**	**9,902**
RC1_2_1	34,595.9	54	18	**7.4**	**43,828**	34,595.9	**52**	**17**	7.6	44,773
RC1_2_2	31,774.7	127	44	**7.7**	**103,660**	31,774.8	**118**	**43**	8.0	111,973
RC1_2_6	32,554.7	160	**20**	34.2	402,772	32,554.8	**137**	23	**26.7**	**324,806**
RC1_2_8	30,216.1	462	**34**	45.3	1,202,445	30,216.1	**361**	38	**38.2**	**995,481**
RC2_2_1	27,891.0	1,927	**104**	7.3	6,676	**27,892.7**	**1,913**	115	**5.6**	**5,196**
RC2_2_2	24,588.6	11,588	529	7.5	41,155	**24,588.7**	**9,785**	**511**	**6.2**	**31,888**
RC2_2_5	24,788.5	7,383	493	8.3	88,212	24,788.5	**6,263**	**475**	**7.5**	**67,406**
RC2_2_6	24,724.4	5,859	433	8.2	49,520	24,724.4	**5,067**	**388**	**7.7**	**41,217**
#Best	26/31	3/31	14/31	9/31	8/31	29/31	28/31	19/31	25/31	24/31

Table A.14 Detailed results for comparing `Default` vs `SetBased` settings using `only-cycle` DSSR after adding non-robust cuts

Instance	Default					SetBased				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,247.0	822	**192**	13.9	265,467	26,247.0	**782**	201	**12.4**	**247,868**
C1_2_3	26,719.2	123,963	**433**	5.8	152,581	**26,721.9**	**39,138**	505	**4.7**	**129,423**
C1_2_4	26,256.0	1,357	374	10.1	304,547	26,256.0	**1,151**	**357**	**9.2**	**260,764**
C1_2_9	26,396.0	220	**58**	13.6	124,522	26,396.0	**212**	59	**11.7**	**106,689**
C2_2_1	19,152.8	**402**	**70**	1.0	455	**19,154.4**	414	74	1.0	**436**
C2_2_2	**18,514.0**	13,515	**1,637**	1.8	7,683	18,492.1	**13,421**	1,699	**1.5**	**6,113**
C2_2_5	18,696.0	2,091	562	1.9	**2,717**	18,696.0	**2,031**	**530**	1.8	2,997
C2_2_6	18,448.0	5,729	852	2.8	**8,996**	18,448.0	**4,328**	**619**	2.9	9,145
C2_2_7	18,422.0	**7,449**	**790**	3.5	14,261	18,422.0	7,748	933	**3.0**	**12,193**
C2_2_8	**18,137.0**	9,221	696	4.8	25,811	18,135.8	**7,987**	**654**	**4.5**	**23,848**
C2_2_9	18,150.0	21,618	1,142	3.9	44,040	18,150.0	**17,313**	**957**	3.9	**42,650**
R1_210	32,792.4	2,041	390	3.1	86,138	**32,793.1**	**1,958**	390	**2.9**	**81,608**
R1_2_4	**30,367.6**	32,462	576	**3.6**	410,047	30,363.9	**17,440**	**507**	3.8	**386,889**
R1_2_5	40,457.1	423	**253**	1.1	7,508	40,457.0	**411**	258	1.1	**7,360**
R1_2_6	35,454.7	**1,980**	**436**	2.0	**55,897**	**35,457.1**	2,014	467	1.9	56,159
R1_2_7	**31,393.5**	3,589	452	3.2	175,536	31,390.9	**2,872**	**412**	3.3	**164,077**
R1_2_8	29,328.1	**6,201**	**452**	4.7	379,879	**29,330.1**	6,478	468	**4.5**	**368,905**
R1_2_9	37,267.4	688	**291**	1.7	**21,969**	**37,268.2**	**676**	301	1.7	22,490
R2_2_1	34,680.0	**1,756**	**268**	1.6	710	34,680.0	1,860	294	1.6	**709**
R2_2_2	30,082.0	6,215	**568**	3.1	7,089	30,082.0	**6,069**	665	**2.6**	**5,664**
R2_2_5	30,609.2	5,358	1,405	1.5	**3,232**	**30,609.5**	**5,294**	**1,336**	1.4	3,312
R2_2_6	26,750.2	**17,642**	**2,167**	2.0	13,086	**26,750.5**	19,404	2,765	**1.7**	**10,458**
R2_2_9	**28,433.0**	5,808	992	2.0	5,578	28,415.2	**5,188**	**877**	2.0	**4,926**
RC1_2_1	34,993.5	**632**	**312**	1.4	**14,926**	**34,998.0**	728	329	1.4	15,720
RC1_2_2	32,068.1	**25,064**	465	2.5	101,532	**32,068.8**	34,489	**453**	2.5	**100,049**
RC1_2_6	32,936.9	5,602	**498**	3.8	127,355	**32,939.9**	**5,157**	544	**3.5**	**126,882**
RC1_2_8	**30,502.0**	125,068	585	7.3	**926,295**	30,499.4	**109,219**	**562**	7.4	945,591
RC2_2_1	27,960.8	2,280	638	2.1	**1,904**	**27,961.8**	**2,231**	**531**	2.1	2,009
RC2_2_2	**24,780.0**	**49,224**	6,730	1.6	10,137	24,772.0	55,776	**6,448**	**1.5**	**8,972**
RC2_2_5	24,914.0	15,122	**3,160**	2.3	30,460	24,914.0	**14,721**	3,309	**2.1**	**26,173**
RC2_2_6	24,951.0	10,970	2,403	2.6	19,502	24,951.0	**8,951**	**2,053**	2.5	**17,478**
#Best	19/31	9/31	17/31	17/31	8/31	24/31	22/31	15/31	30/31	23/31

Table A.15 Detailed results for comparing `Default` vs `Pairwise` settings using `only-cycle` DSSR before adding non-robust cuts

Instance	Default					Pairwise				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,197.3	655	**72**	29.3	560,811	26,197.3	**595**	80	**23.9**	**427,899**
C1_2_3	26,597.4	476	132	12.4	193,347	26,597.4	**390**	**111**	**8.5**	**120,149**
C1_2_4	26,197.7	1,066	220	13.9	400,174	26,197.7	**965**	**196**	**13.8**	**353,881**
C1_2_9	26,396.0	219	**58**	13.6	124,522	26,396.0	**215**	68	**9.5**	**84,917**
C2_2_1	19,150.3	**349**	8	1.0	388	19,150.3	350	8	1.0	388
C2_2_2	18,396.0	9,504	794	2.5	10,412	18,396.0	**5,983**	**610**	**2.2**	**8,299**
C2_2_5	18,607.1	1,339	165	3.4	3,829	18,607.1	**1,258**	**158**	3.3	**3,662**
C2_2_6	18,406.7	4,173	324	**5.3**	15,499	18,406.7	**3,168**	**243**	5.5	**14,472**
C2_2_7	18,356.4	6,342	**429**	5.3	22,082	18,356.4	**5,605**	436	**4.7**	**17,806**
C2_2_8	18,082.4	7,977	363	8.0	42,366	18,082.4	**6,428**	**334**	**7.1**	**32,202**
C2_2_9	18,011.7	13,272	486	6.7	56,421	18,011.8	**10,553**	**417**	**6.5**	**46,727**
R1_210	32,450.2	**153**	**26**	16.3	210,725	**32,451.3**	157	27	**14.7**	**191,588**
R1_2_4	30,010.4	**270**	43	**13.2**	**357,645**	**30,010.6**	291	**38**	14.1	386,546
R1_2_5	40,068.1	42	12	2.0	9,881	40,068.1	**42**	12	2.0	9,881
R1_2_6	34,930.6	101	**41**	4.6	53,930	34,930.6	**98**	42	**4.3**	**49,923**
R1_2_7	31,000.6	**237**	51	**8.7**	**190,129**	31,000.6	243	**47**	9.8	213,354
R1_2_8	29,032.2	356	**34**	19.8	589,966	29,032.1	**327**	38	**16.8**	**476,911**
R1_2_9	36,788.3	**68**	**17**	7.4	52,206	36,788.3	68	18	**6.7**	**47,286**
R2_2_1	34,625.1	**1,645**	**88**	2.7	1,232	34,625.1	1,718	92	**2.5**	**1,156**
R2_2_2	30,006.1	5,397	**282**	5.1	12,466	30,006.1	**4,965**	290	**4.4**	**10,221**
R2_2_5	30,290.3	2,946	188	3.9	4,670	30,290.3	**2,764**	**163**	3.8	**4,325**
R2_2_6	26,455.6	9,738	414	5.9	39,427	26,455.6	**8,909**	**387**	**5.4**	**31,820**
R2_2_9	28,206.0	4,217	252	**4.7**	11,431	28,206.1	**3,639**	**187**	5.0	**11,062**
RC1_2_1	34,595.9	**54**	18	7.4	43,828	34,595.9	54	18	7.4	43,828
RC1_2_2	31,774.7	127	44	7.7	103,660	31,774.7	**123**	44	**7.2**	**96,327**
RC1_2_6	32,554.7	160	20	34.2	402,772	32,554.7	**152**	20	**31.2**	**361,162**
RC1_2_8	30,216.1	462	**34**	45.3	1,202,445	30,216.1	**439**	36	**41.7**	**1,062,486**
RC2_2_1	27,891.0	1,927	**104**	7.3	6,676	**27,892.7**	**1,890**	111	**5.9**	**5,281**
RC2_2_2	24,588.6	11,588	**529**	7.5	41,155	**24,592.5**	**10,071**	549	**6.0**	**29,763**
RC2_2_5	24,788.5	7,383	493	**8.3**	88,212	24,788.5	**6,327**	**387**	9.6	**84,634**
RC2_2_6	24,724.4	5,859	433	8.2	49,520	24,724.5	**5,259**	**381**	**8.0**	**43,544**
#Best	27/31	7/31	17/31	10/31	5/31	31/31	24/31	19/31	26/31	29/31

Table A.16 Detailed results for comparing `Default` vs `Pairwise` settings using `only-cycle` DSSR after adding non-robust cuts

Instance	Default					Pairwise				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,247.0	822	**192**	13.9	265,467	26,247.0	**763**	206	**12.3**	**220,337**
C1_2_3	26,719.2	123,963	**433**	5.8	152,581	**26,720.1**	**57,438**	458	**4.5**	**117,110**
C1_2_4	26,256.0	1,357	374	10.1	304,547	26,256.0	**1,250**	**347**	**10.0**	**272,266**
C1_2_9	26,396.0	220	**58**	13.6	124,522	26,396.0	**216**	68	**9.5**	**84,917**
C2_2_1	19,152.8	**402**	70	1.0	455	19,152.8	402	70	1.0	455
C2_2_2	18,514.0	13,515	1,637	1.8	7,683	18,514.0	**9,767**	**1,393**	**1.6**	**7,116**
C2_2_5	18,696.0	2,091	562	1.9	**2,717**	18,696.0	**1,957**	**517**	1.8	2,903
C2_2_6	18,448.0	5,729	852	2.8	8,996	18,448.0	**4,559**	**761**	**2.6**	**7,490**
C2_2_7	18,422.0	7,449	790	3.5	14,261	18,422.0	**6,624**	**785**	**3.2**	**12,041**
C2_2_8	18,137.0	9,221	**696**	4.8	25,811	18,137.0	**8,008**	781	**3.8**	**18,482**
C2_2_9	18,150.0	21,618	1,142	3.9	44,040	18,150.0	**18,888**	**1,001**	3.8	**41,764**
R1_210	32,792.4	**2,041**	**390**	3.1	86,138	**32,794.6**	2,377	410	**2.8**	**79,223**
R1_2_4	**30,367.6**	32,462	576	**3.6**	410,047	30,362.2	**22,112**	**509**	3.8	**399,613**
R1_2_5	40,457.1	423	253	1.1	7,508	40,457.1	**422**	253	1.1	7,508
R1_2_6	35,454.7	**1,980**	436	2.0	**55,897**	**35,457.0**	2,235	**430**	2.0	58,068
R1_2_7	**31,393.5**	3,589	452	**3.2**	**175,536**	31,392.3	**3,379**	**413**	3.4	177,490
R1_2_8	29,328.1	6,201	**452**	4.7	379,879	**29,328.3**	**5,554**	466	**4.4**	**340,491**
R1_2_9	37,267.4	**688**	**291**	1.7	**21,969**	**37,271.3**	807	307	1.6	22,444
R2_2_1	34,680.0	**1,756**	268	1.6	**710**	34,680.0	1,823	**230**	1.7	749
R2_2_2	30,082.0	6,215	568	3.1	7,089	**30,088.0**	**5,687**	**561**	**2.8**	**6,153**
R2_2_5	30,609.2	5,358	**1,405**	1.5	**3,232**	**30,610.3**	**5,225**	1,434	1.4	3,339
R2_2_6	**26,750.2**	**17,642**	**2,167**	2.0	13,086	26,748.6	18,751	2,462	**1.8**	**11,163**
R2_2_9	**28,433.0**	5,808	**992**	2.0	5,578	28,429.1	**5,232**	1,005	**1.8**	**4,638**
RC1_2_1	34,993.5	632	312	1.4	14,926	34,993.5	**631**	312	1.4	14,926
RC1_2_2	32,068.1	**25,064**	465	2.5	**101,532**	**32,072.1**	28,047	**448**	2.5	102,412
RC1_2_6	32,936.9	**5,602**	**498**	3.8	**127,355**	**32,941.0**	6,413	532	**3.6**	129,387
RC1_2_8	**30,502.0**	**125,068**	585	**7.3**	**926,295**	30,500.0	137,624	**547**	7.7	964,933
RC2_2_1	**27,960.8**	2,280	638	2.1	**1,904**	27,932.2	**2,153**	**491**	2.2	1,964
RC2_2_2	**24,780.0**	**49,224**	**6,730**	1.6	10,137	24,776.2	57,708	7,092	**1.4**	**8,741**
RC2_2_5	24,914.0	15,122	3,160	**2.3**	30,460	24,914.0	**10,197**	**1,915**	2.8	**29,270**
RC2_2_6	24,951.0	10,970	2,403	2.6	19,502	24,951.0	**9,325**	**2,325**	**2.3**	**15,720**
#Best	22/31	10/31	15/31	15/31	13/31	24/31	21/31	19/31	27/31	21/31

Table A.17 Detailed results for comparing `Default` vs `SetPair` settings using `only-cycle` DSSR before adding non-robust cuts

Instance	Default					SetPair				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,197.3	655	**72**	29.3	560,811	26,197.3	**546**	74	**24.9**	**445,636**
C1_2_3	26,597.4	476	**132**	12.4	193,347	26,597.4	**412**	147	**10.3**	**162,955**
C1_2_4	26,197.7	1,066	220	13.9	400,174	26,197.7	**821**	**209**	**12.5**	**328,362**
C1_2_9	26,396.0	219	58	13.6	124,522	26,396.0	**198**	**52**	**12.2**	**108,642**
C2_2_1	19,150.3	**349**	8	1.0	388	19,150.3	357	8	1.0	388
C2_2_2	18,396.0	9,504	794	2.5	10,412	18,396.0	**6,668**	**739**	**1.8**	**6,935**
C2_2_5	18,607.1	1,339	165	3.4	3,829	18,607.1	**1,257**	**155**	3.4	**3,698**
C2_2_6	18,406.7	4,173	324	5.3	15,499	18,406.7	**3,605**	**322**	**4.6**	**12,457**
C2_2_7	18,356.4	6,342	**429**	5.3	22,082	18,356.4	**5,688**	496	**4.1**	**15,732**
C2_2_8	18,082.4	7,977	363	8.0	42,366	18,082.4	**6,843**	**333**	**7.4**	**34,009**
C2_2_9	18,011.7	13,272	486	**6.7**	56,421	**18,011.9**	**9,361**	**340**	7.6	**54,178**
R1_210	32,450.2	153	26	16.3	210,725	**32,451.3**	**135**	**25**	**16.0**	**206,761**
R1_2_4	**30,010.4**	270	43	13.2	357,645	30,009.9	**213**	**41**	**12.6**	**330,678**
R1_2_5	40,068.1	42	12	2.0	9,881	40,068.1	**42**	12	2.0	**9,874**
R1_2_6	34,930.6	101	41	4.6	**53,930**	34,930.6	**97**	**40**	4.7	56,384
R1_2_7	31,000.6	237	51	8.7	190,129	**31,000.8**	**201**	**49**	**8.4**	**179,043**
R1_2_8	29,032.2	356	**34**	19.8	589,966	29,032.2	**276**	39	**15.7**	**448,479**
R1_2_9	36,788.3	68	17	7.4	52,206	**36,788.6**	**65**	17	**7.1**	**50,041**
R2_2_1	34,625.1	**1,645**	88	**2.7**	**1,232**	34,625.1	1,679	**70**	3.0	1,396
R2_2_2	30,006.1	5,397	282	5.1	12,466	30,006.1	**4,832**	**270**	**4.6**	**11,151**
R2_2_5	30,290.3	2,946	188	**3.9**	**4,670**	30,290.3	**2,657**	**131**	4.5	5,092
R2_2_6	26,455.6	9,738	414	**5.9**	39,427	26,455.5	**8,142**	**329**	6.1	**35,795**
R2_2_9	28,206.0	4,217	252	4.7	11,431	28,206.0	**3,938**	**223**	**4.3**	**9,247**
RC1_2_1	34,595.9	54	18	**7.4**	**43,828**	34,595.9	**52**	**17**	7.8	45,483
RC1_2_2	31,774.7	127	44	7.7	103,660	31,774.7	**115**	44	7.6	**97,615**
RC1_2_6	32,554.7	160	**20**	34.2	402,772	32,554.8	**143**	23	**28.1**	**331,637**
RC1_2_8	**30,216.1**	462	**34**	45.3	1,202,445	30,215.7	**384**	35	**41.9**	**1,044,786**
RC2_2_1	27,891.0	**1,927**	**104**	7.3	6,676	**27,892.7**	1,956	127	**5.1**	**4,572**
RC2_2_2	24,588.6	11,588	529	7.5	41,155	**24,590.9**	**9,829**	**499**	**6.6**	**31,844**
RC2_2_5	24,788.5	7,383	493	**8.3**	88,212	**24,797.9**	**6,244**	**384**	9.8	**88,198**
RC2_2_6	24,724.4	5,859	433	8.2	49,520	24,724.4	**5,233**	**374**	**8.1**	**42,073**
#Best	24/31	3/31	11/31	11/31	5/31	29/31	28/31	24/31	25/31	27/31

Table A.18 Detailed results for comparing `Default` vs `SetPair` settings using `only-cycle` DSSR after adding non-robust cuts

Instance	Default					SetPair				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,247.0	822	**192**	13.9	265,467	26,247.0	**725**	203	**12.2**	**218,519**
C1_2_3	26,719.2	**123,963**	**433**	5.8	152,581	**26,721.5**	241,285	481	**5.0**	**141,088**
C1_2_4	26,256.0	1,357	374	10.1	304,547	26,256.0	**1,096**	**355**	**9.3**	**256,656**
C1_2_9	26,396.0	220	58	13.6	124,522	26,396.0	**199**	**52**	**12.2**	**108,642**
C2_2_1	19,152.8	**402**	**70**	1.0	**455**	**19,156.4**	430	98	1.0	475
C2_2_2	18,514.0	13,515	1,637	1.8	7,683	18,514.0	**10,401**	**1,630**	**1.4**	**5,195**
C2_2_5	18,696.0	2,091	562	1.9	2,717	18,696.0	**1,994**	**556**	1.8	**2,642**
C2_2_6	18,448.0	5,729	852	2.8	8,996	18,448.0	**4,944**	**831**	**2.6**	**7,256**
C2_2_7	18,422.0	7,449	**790**	3.5	14,261	18,422.0	**7,322**	1,054	**2.6**	**9,629**
C2_2_8	18,137.0	9,221	696	4.8	25,811	18,137.0	**8,071**	**674**	**4.3**	**20,740**
C2_2_9	18,150.0	21,618	1,142	**3.9**	44,040	18,150.0	**16,638**	**928**	4.1	**41,527**
R1_210	32,792.4	**2,041**	**390**	3.1	86,138	**32,794.9**	2,365	418	**2.9**	**82,495**
R1_2_4	**30,367.6**	**32,462**	576	3.6	410,047	30,366.9	40,007	**559**	3.5	**374,689**
R1_2_5	**40,457.1**	423	253	1.1	7,508	40,451.6	**328**	**221**	1.1	**7,251**
R1_2_6	35,454.7	**1,980**	**436**	2.0	55,897	**35,462.4**	2,497	509	1.9	**55,871**
R1_2_7	**31,393.5**	**3,589**	452	3.2	175,536	31,392.5	4,044	**430**	3.3	**172,441**
R1_2_8	29,328.1	**6,201**	**452**	4.7	379,879	**29,331.0**	8,925	487	**4.3**	**371,794**
R1_2_9	37,267.4	**688**	**291**	1.7	**21,969**	**37,272.2**	806	323	1.6	22,272
R2_2_1	34,680.0	**1,756**	268	**1.6**	**710**	34,680.0	1,803	**222**	1.7	751
R2_2_2	30,082.0	6,215	568	3.1	7,089	30,082.0	**5,723**	**563**	**2.8**	**6,366**
R2_2_5	**30,609.2**	5,358	1,405	1.5	3,232	30,603.9	**4,552**	**1,206**	1.5	**2,790**
R2_2_6	**26,750.2**	17,642	**2,167**	2.0	13,086	26,749.5	**17,406**	2,348	**1.8**	**11,234**
R2_2_9	**28,433.0**	5,808	**992**	2.0	5,578	28,428.0	**5,487**	1,018	**1.8**	**4,622**
RC1_2_1	34,993.5	**632**	**312**	1.4	14,926	**34,994.6**	659	318	1.4	**14,533**
RC1_2_2	32,068.1	**25,064**	**465**	2.5	101,532	**32,074.2**	57,720	494	**2.4**	**101,310**
RC1_2_6	32,936.9	**5,602**	**498**	3.8	127,355	**32,942.6**	8,596	558	**3.4**	**125,762**
RC1_2_8	**30,502.0**	**125,068**	585	**7.3**	926,295	30,497.4	136,304	**541**	7.8	**898,946**
RC2_2_1	27,960.8	**2,280**	638	2.1	**1,904**	**27,961.1**	2,283	**555**	2.1	1,916
RC2_2_2	24,780.0	**49,224**	**6,730**	1.6	10,137	24,780.0	68,376	7,839	**1.4**	**8,530**
RC2_2_5	24,914.0	15,122	3,160	2.3	30,460	24,914.0	**13,006**	**2,856**	2.3	**27,022**
RC2_2_6	24,951.0	10,970	**2,403**	2.6	19,502	24,951.0	**9,706**	2,543	**2.2**	**14,405**
#Best	21/31	15/31	15/31	14/31	4/31	24/31	16/31	16/31	28/31	27/31

Table A.19 Detailed results for comparing `Default` vs `SetBased` settings using `all-arcs` DSSR before adding non-robust cuts

Instance	Default					SetBased				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	**26,197.3**	477	74	**16.1**	**310,753**	26,197.2	**381**	**61**	16.5	319,202
C1_2_3	26,597.4	323	113	**4.1**	**73,609**	26,597.4	**253**	**105**	4.8	82,096
C1_2_4	26,197.7	525	**157**	6.2	197,897	26,197.7	**459**	177	**4.9**	**152,005**
C1_2_9	26,396.0	**184**	55	11.1	**97,994**	26,396.0	200	**51**	11.1	103,308
C2_2_1	19,150.3	**359**	8	1.0	388	19,150.3	362	8	1.0	388
C2_2_2	18,394.0	**3,905**	**518**	1.4	6,134	**18,396.0**	4,645	561	1.3	**5,711**
C2_2_5	18,607.1	**1,259**	**151**	3.3	3,707	18,607.1	1,272	165	**2.9**	**3,512**
C2_2_6	18,406.7	3,079	252	**4.1**	**11,771**	18,406.7	**2,684**	**211**	4.6	13,276
C2_2_7	18,356.4	3,905	360	3.4	13,757	18,356.4	**3,612**	360	**3.2**	**13,574**
C2_2_8	18,082.4	3,564	232	**5.0**	**23,873**	18,082.4	**3,412**	**200**	5.5	27,222
C2_2_9	**18,011.9**	6,020	306	**4.5**	**35,977**	18,011.2	**5,490**	**268**	4.8	40,270
R1_210	32,451.3	132	**21**	14.2	183,530	32,451.3	**123**	25	**12.0**	**160,451**
R1_2_4	**30,010.5**	198	**36**	8.7	238,385	30,009.9	**158**	40	**7.5**	**193,054**
R1_2_5	40,068.1	43	12	**2.0**	**9,881**	40,068.1	**42**	**11**	2.2	10,827
R1_2_6	34,930.6	93	44	**3.5**	**39,224**	34,930.6	**88**	**36**	3.9	48,222
R1_2_7	**31,001.0**	189	51	**5.6**	**117,367**	31,000.4	**163**	**47**	5.8	124,032
R1_2_8	29,032.3	250	35	**10.2**	**302,465**	29,032.3	**194**	**31**	11.0	318,416
R1_2_9	36,788.5	68	18	6.4	**46,453**	36,788.5	**66**	**17**	6.5	47,789
R2_2_1	34,625.1	1,751	**106**	2.3	1,060	34,625.1	**1,682**	115	**2.1**	**979**
R2_2_2	30,006.1	4,391	291	**3.2**	**7,647**	30,006.1	**3,933**	**217**	3.8	9,269
R2_2_5	30,290.3	**2,681**	**146**	3.4	4,062	30,290.3	2,713	150	**3.2**	**3,787**
R2_2_6	26,455.5	6,253	278	4.3	**28,234**	26,455.6	**5,382**	**239**	**4.2**	28,634
R2_2_9	28,206.0	3,616	205	**3.4**	**8,047**	28,206.0	**3,471**	**173**	3.6	8,488
RC1_2_1	34,595.9	54	18	**7.4**	**43,828**	34,595.9	**51**	**17**	7.6	44,773
RC1_2_2	31,774.7	112	**41**	6.5	**84,161**	31,774.7	**101**	42	**6.3**	84,595
RC1_2_6	32,554.8	121	20	21.9	255,845	**32,554.9**	**103**	20	**20.8**	**243,494**
RC1_2_8	30,216.1	236	30	19.3	517,598	30,216.1	**194**	**29**	**18.9**	**495,194**
RC2_2_1	27,891.0	1,873	117	4.9	4,509	27,891.0	**1,831**	**111**	**4.5**	**4,382**
RC2_2_2	24,588.6	7,141	490	3.7	22,238	**24,588.8**	**6,532**	**450**	3.6	**21,101**
RC2_2_5	24,788.5	**3,338**	**263**	4.9	**47,954**	24,788.5	3,432	273	4.9	48,149
RC2_2_6	24,724.4	3,484	276	4.7	28,380	24,724.4	**3,311**	**269**	4.7	**26,591**
#Best	28/31	6/31	12/31	19/31	18/31	27/31	25/31	22/31	19/31	14/31

Table A.20 Detailed results for comparing `Default` vs `SetBased` settings using `all-arcs` DSSR after adding non-robust cuts

Instance	Default					SetBased				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,247.0	616	184	8.1	**19,180**	26,247.0	**492**	**167**	**8.0**	19,513
C1_2_3	**26,723.2**	**34,047**	498	**2.0**	**32,563**	26,717.7	44,182	**410**	2.3	33,328
C1_2_4	26,256.0	724	**283**	4.2	34,120	26,256.0	**654**	319	**3.4**	**33,587**
C1_2_9	26,396.0	**185**	55	11.1	**8,836**	26,396.0	201	**51**	11.1	9,292
C2_2_1	19,152.8	**413**	**70**	1.0	455	**19,154.4**	421	74	1.0	**436**
C2_2_2	**18,514.0**	12,368	1,415	1.2	5,863	18,489.1	**7,611**	**1,234**	1.2	**4,423**
C2_2_5	18,696.0	**2,008**	**561**	1.7	**1,602**	18,696.0	2,034	596	**1.6**	1,685
C2_2_6	18,448.0	4,111	**628**	2.3	**3,102**	18,448.0	**3,768**	653	**2.2**	3,104
C2_2_7	18,422.0	**4,968**	**737**	2.2	**4,033**	18,422.0	4,998	833	**2.0**	4,449
C2_2_8	18,137.0	4,633	577	**2.7**	5,321	18,137.0	**4,390**	**513**	2.8	**5,038**
C2_2_9	18,150.0	11,608	977	**2.2**	**10,730**	18,150.0	**10,306**	**800**	2.4	12,342
R1_210	**32,794.9**	2,140	417	2.2	**29,849**	32,793.0	**1,836**	**415**	2.2	30,208
R1_2_4	**30,364.6**	18,415	**533**	2.0	**127,556**	30,364.1	**15,173**	551	1.9	128,498
R1_2_5	40,457.1	427	**253**	1.1	7,141	40,457.0	**418**	258	1.1	**7,007**
R1_2_6	35,456.9	**1,995**	461	1.6	**29,240**	**35,461.9**	2,074	**458**	1.5	31,203
R1_2_7	**31,392.8**	2,506	422	2.1	54,672	31,391.1	**2,333**	**406**	2.1	**54,599**
R1_2_8	29,328.1	**4,897**	**451**	2.4	**100,373**	**29,330.2**	5,120	462	2.3	103,922
R1_2_9	**37,270.4**	750	308	1.5	13,958	37,266.9	**640**	**285**	1.6	**13,359**
R2_2_1	34,680.0	1,851	**230**	1.6	**452**	34,680.0	**1,806**	277	**1.5**	460
R2_2_2	30,082.0	5,271	594	**2.1**	**2,213**	**30,102.2**	**4,501**	**464**	2.3	2,287
R2_2_5	**30,609.8**	**4,836**	1,302	1.3	**2,128**	30,609.6	4,842	**1,292**	1.3	2,140
R2_2_6	**26,748.4**	13,208	**1,972**	1.5	**6,390**	26,747.2	**13,188**	2,240	**1.4**	6,780
R2_2_9	**28,433.0**	**4,994**	**911**	1.6	**2,725**	28,430.3	5,080	962	1.5	2,850
RC1_2_1	34,993.5	**634**	**312**	1.4	**10,372**	**34,998.0**	734	329	1.4	11,170
RC1_2_2	32,072.6	28,285	495	1.8	**41,646**	**32,073.3**	**25,273**	**474**	1.9	45,399
RC1_2_6	**32,938.2**	5,123	503	2.7	36,610	32,936.1	**3,672**	**476**	2.7	**36,397**
RC1_2_8	30,499.5	59,534	**543**	3.1	**133,139**	**30,500.3**	**45,722**	553	3.0	141,567
RC2_2_1	**27,966.5**	2,242	682	1.7	**909**	27,961.8	**2,124**	**550**	1.8	954
RC2_2_2	**24,780.0**	31,302	4,948	1.3	**6,565**	24,769.8	**26,760**	**4,129**	1.3	6,795
RC2_2_5	24,914.0	**7,532**	**1,817**	1.6	**14,139**	24,914.0	9,052	2,334	**1.5**	15,662
RC2_2_6	24,951.0	7,288	2,026	1.6	**8,280**	24,951.0	**6,840**	**1,953**	1.5	9,069
#Best	24/31	11/31	14/31	23/31	23/31	19/31	20/31	17/31	27/31	8/31

Table A.21 Detailed results for comparing `Default` vs `Pairwise` settings using `all-arcs` DSSR before adding non-robust cuts

Instance	Default					Pairwise				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,197.3	477	74	**16.1**	310,753	26,197.3	**432**	**66**	16.4	**295,977**
C1_2_3	26,597.4	323	**113**	4.1	73,609	26,597.4	**260**	118	4.2	**67,797**
C1_2_4	26,197.7	525	**157**	6.2	197,897	26,197.7	**423**	167	**5.2**	**147,922**
C1_2_9	26,396.0	**184**	**55**	11.1	97,994	26,396.0	187	60	**9.5**	**82,941**
C2_2_1	19,150.3	359	8	1.0	388	19,150.3	**351**	8	1.0	388
C2_2_2	18,394.0	**3,905**	**518**	1.4	6,134	**18,396.0**	4,871	615	1.4	**5,671**
C2_2_5	18,607.1	1,259	151	3.3	3,707	18,607.1	**1,167**	**141**	**3.1**	**3,436**
C2_2_6	18,406.7	3,079	252	**4.1**	**11,771**	18,406.7	**2,695**	**204**	4.6	12,090
C2_2_7	18,356.4	**3,905**	360	**3.4**	**13,757**	18,356.4	4,026	**339**	3.6	13,981
C2_2_8	18,082.4	3,564	232	**5.0**	23,873	18,082.4	**3,405**	**203**	5.3	**23,366**
C2_2_9	18,011.9	6,020	306	4.5	35,977	18,012.0	**4,696**	**285**	**4.3**	**31,165**
R1_210	**32,451.3**	132	**21**	14.2	183,530	32,450.6	**128**	24	**11.6**	**150,038**
R1_2_4	**30,010.5**	198	**36**	8.7	238,385	30,009.9	**188**	43	**6.9**	**178,591**
R1_2_5	40,068.1	43	12	2.0	9,881	40,068.1	**42**	12	2.0	9,881
R1_2_6	34,930.6	**93**	44	3.5	**39,224**	34,930.6	94	**42**	3.5	40,691
R1_2_7	31,001.0	189	51	**5.6**	**117,367**	**31,001.2**	**187**	**46**	5.9	122,804
R1_2_8	29,032.3	250	**35**	10.2	302,465	29,032.3	**245**	36	**9.8**	**286,602**
R1_2_9	**36,788.5**	68	**18**	6.4	46,453	36,788.3	**67**	19	**5.9**	**42,793**
R2_2_1	34,625.1	**1,751**	**106**	2.3	1,060	34,625.1	1,782	124	**2.0**	**899**
R2_2_2	30,006.1	4,391	291	**3.2**	**7,647**	30,006.1	**4,016**	**246**	3.4	8,324
R2_2_5	30,290.3	2,681	**146**	3.4	4,062	30,290.3	**2,640**	153	**3.1**	**3,579**
R2_2_6	26,455.5	6,253	**278**	4.3	28,234	26,455.5	**5,762**	294	**3.5**	**22,616**
R2_2_9	28,206.0	3,616	205	**3.4**	8,047	28,206.0	**3,339**	**162**	3.6	**8,044**
RC1_2_1	34,595.9	**54**	18	7.4	43,828	34,595.9	55	18	7.4	43,828
RC1_2_2	31,774.7	**112**	**41**	6.5	84,161	31,774.7	117	45	**5.8**	**76,159**
RC1_2_6	32,554.8	121	**20**	21.9	255,845	**32,554.9**	**116**	21	**20.3**	**233,524**
RC1_2_8	**30,216.1**	236	30	**19.3**	517,598	30,215.7	**224**	**29**	19.4	**500,651**
RC2_2_1	27,891.0	1,873	**117**	4.9	4,509	**27,892.7**	**1,761**	126	**4.0**	**3,736**
RC2_2_2	24,588.6	7,141	490	**3.7**	22,238	**24,589.0**	**5,930**	**421**	3.8	**20,558**
RC2_2_5	24,788.5	**3,338**	**263**	4.9	47,954	24,788.5	3,435	306	**4.5**	**42,039**
RC2_2_6	24,724.4	3,484	276	**4.7**	28,380	24,724.4	**3,193**	**242**	5.0	**26,089**
#Best	26/31	8/31	18/31	16/31	8/31	27/31	23/31	16/31	21/31	26/31

Table A.22 Detailed results for comparing `Default` vs `Pairwise` settings using `all-arcs` DSSR after adding non-robust cuts

Instance	Default					Pairwise				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,247.0	616	184	8.1	19,180	26,247.0	**571**	**174**	**7.9**	**18,195**
C1_2_3	**26,723.2**	**34,047**	498	**2.0**	**32,563**	26,722.6	45,995	**446**	2.2	36,151
C1_2_4	26,256.0	724	**283**	4.2	34,120	26,256.0	**581**	304	**3.6**	**30,712**
C1_2_9	26,396.0	**185**	**55**	11.1	8,836	26,396.0	187	60	**9.5**	**8,731**
C2_2_1	19,152.8	413	70	1.0	455	19,152.8	**404**	70	1.0	455
C2_2_2	**18,514.0**	12,368	1,415	1.2	5,863	18,471.6	**6,580**	**1,060**	1.2	**4,238**
C2_2_5	18,696.0	2,008	561	1.7	**1,602**	18,696.0	**1,956**	**493**	1.8	1,718
C2_2_6	18,448.0	4,111	**628**	2.3	3,102	18,448.0	**3,730**	638	**2.2**	**2,783**
C2_2_7	18,422.0	**4,968**	737	2.2	4,033	18,422.0	5,067	**704**	2.3	**3,986**
C2_2_8	18,137.0	4,633	577	2.7	5,321	18,137.0	**4,494**	**569**	2.6	**4,814**
C2_2_9	18,150.0	11,608	977	2.2	10,730	18,150.0	**9,256**	**872**	2.2	**10,514**
R1_210	**32,794.9**	2,140	417	2.2	29,849	32,794.1	**1,940**	**388**	2.2	**29,589**
R1_2_4	30,364.6	18,415	**533**	2.0	127,556	**30,366.1**	**18,143**	543	2.0	**123,662**
R1_2_5	40,457.1	427	253	1.1	7,141	40,457.1	**421**	253	1.1	7,141
R1_2_6	**35,456.9**	1,995	461	1.6	29,240	35,450.9	**1,570**	**416**	1.6	**27,229**
R1_2_7	31,392.8	**2,506**	**422**	2.1	**54,672**	**31,393.2**	2,935	462	**2.0**	57,162
R1_2_8	29,328.1	**4,897**	451	2.4	100,373	**29,329.3**	5,023	**443**	2.3	**99,639**
R1_2_9	**37,270.4**	**750**	**308**	1.5	13,958	37,270.1	779	316	1.5	**13,649**
R2_2_1	34,680.0	**1,851**	**230**	1.6	**452**	34,680.0	1,925	325	**1.4**	465
R2_2_2	30,082.0	5,271	594	2.1	**2,213**	**30,086.1**	**4,653**	**516**	2.1	2,287
R2_2_5	**30,609.8**	4,836	1,302	1.3	2,128	30,607.0	**4,341**	**1,182**	1.3	**1,994**
R2_2_6	26,748.4	**13,208**	**1,972**	1.5	**6,390**	**26,750.5**	13,681	2,188	**1.4**	6,464
R2_2_9	**28,433.0**	4,994	911	1.6	2,725	28,424.9	**4,674**	**879**	1.5	**2,625**
RC1_2_1	34,993.5	**634**	312	1.4	10,372	34,993.5	641	312	1.4	10,372
RC1_2_2	**32,072.6**	28,285	495	1.8	41,646	32,070.7	**21,917**	**488**	1.8	**39,992**
RC1_2_6	32,938.2	5,123	**503**	2.7	36,610	**32,938.8**	**4,720**	519	2.6	**36,117**
RC1_2_8	30,499.5	**59,534**	**543**	3.1	**133,139**	**30,502.3**	81,561	552	3.1	147,774
RC2_2_1	**27,966.5**	2,242	682	1.7	**909**	27,954.3	**2,025**	**538**	1.8	948
RC2_2_2	**24,780.0**	31,302	4,948	1.3	6,565	24,779.7	**25,767**	**4,458**	1.3	**6,430**
RC2_2_5	24,914.0	**7,532**	**1,817**	1.6	14,139	24,914.0	8,040	2,278	**1.5**	**11,975**
RC2_2_6	24,951.0	7,288	2,026	1.6	8,280	24,951.0	**6,701**	**1,994**	1.5	**7,296**
#Best	24/31	11/31	14/31	23/31	11/31	21/31	20/31	20/31	30/31	23/31

Table A.23 Detailed results for comparing `Default` vs `SetPair` settings using `all-arcs` DSSR before adding non-robust cuts

Instance	Default					SetPair				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,197.3	477	74	**16.1**	**310,753**	26,197.2	**383**	**56**	18.2	324,638
C1_2_3	26,597.4	323	**113**	4.1	73,609	26,597.4	**275**	127	**3.9**	**64,213**
C1_2_4	26,197.7	525	**157**	6.2	197,897	26,197.7	**468**	166	**5.1**	**158,242**
C1_2_9	26,396.0	**184**	55	11.1	97,994	26,396.0	190	**48**	**10.9**	**94,478**
C2_2_1	19,150.3	**359**	8	1.0	388	19,150.3	361	8	1.0	388
C2_2_2	18,394.0	**3,905**	**518**	1.4	6,134	**18,396.0**	4,507	562	1.4	**5,752**
C2_2_5	18,607.1	1,259	**151**	3.3	3,707	18,607.1	**1,229**	154	**2.9**	**3,209**
C2_2_6	18,406.7	3,079	252	**4.1**	**11,771**	18,406.7	**2,680**	**209**	4.4	11,894
C2_2_7	18,356.4	3,905	360	**3.4**	13,757	18,356.4	**3,495**	**323**	3.6	**13,573**
C2_2_8	18,082.4	3,564	232	**5.0**	**23,873**	18,082.4	**3,144**	**169**	6.2	27,415
C2_2_9	**18,011.9**	6,020	306	4.5	35,977	18,010.9	**4,818**	**289**	**4.1**	**29,724**
R1_210	32,451.3	132	**21**	14.2	183,530	32,451.2	**121**	24	**12.0**	**150,794**
R1_2_4	**30,010.5**	198	**36**	8.7	238,385	30,010.3	**161**	40	**7.6**	**201,241**
R1_2_5	40,068.1	43	12	2.0	9,881	40,068.1	**42**	12	2.0	**9,874**
R1_2_6	34,930.6	93	44	**3.5**	**39,224**	34,930.6	**88**	**36**	3.9	46,483
R1_2_7	31,001.0	189	**51**	5.6	117,367	31,001.0	**168**	52	**5.1**	**106,912**
R1_2_8	29,032.3	250	35	**10.2**	302,465	29,032.4	**202**	**32**	10.5	**297,856**
R1_2_9	**36,788.5**	68	**18**	6.4	46,453	36,788.3	**64**	19	**5.9**	**42,799**
R2_2_1	34,625.1	1,751	106	2.3	1,060	34,625.1	**1,668**	**105**	2.3	**1,032**
R2_2_2	30,006.1	4,391	291	**3.2**	**7,647**	30,006.1	**4,028**	**239**	3.6	8,582
R2_2_5	30,290.3	2,681	146	3.4	4,062	30,290.3	**2,514**	**137**	3.4	**3,998**
R2_2_6	26,455.5	6,253	**278**	4.3	28,234	26,455.6	**5,772**	303	**3.6**	**21,723**
R2_2_9	28,206.0	3,616	205	**3.4**	8,047	28,206.0	**3,352**	**163**	3.6	**7,994**
RC1_2_1	34,595.9	54	18	**7.4**	**43,828**	34,595.9	**52**	**17**	7.8	45,483
RC1_2_2	31,774.7	112	41	6.5	**84,161**	31,774.7	**108**	41	6.4	86,447
RC1_2_6	32,554.8	121	20	**21.9**	**255,845**	32,554.8	**108**	**19**	22.2	257,405
RC1_2_8	**30,216.1**	236	30	19.3	517,598	30,215.7	**195**	30	19.2	**499,774**
RC2_2_1	27,891.0	1,873	117	**4.9**	**4,509**	27,891.0	**1,864**	**77**	6.3	5,872
RC2_2_2	24,588.6	7,141	490	3.7	22,238	**24,588.7**	**6,172**	**428**	3.7	**19,722**
RC2_2_5	24,788.5	3,338	263	**4.9**	**47,954**	24,788.5	**3,297**	**242**	5.3	50,913
RC2_2_6	24,724.4	3,484	276	4.7	28,380	24,724.4	**3,281**	**259**	4.8	**24,910**
#Best	29/31	3/31	13/31	21/31	11/31	27/31	28/31	22/31	19/31	21/31

Table A.24 Detailed results for comparing `Default` vs `SetPair` settings using `all-arcs` DSSR after adding non-robust cuts

Instance	Default					SetPair				
	lb	**T(s)**	**#iters**	**#DSSR**	**#labels**	**lb**	**T(s)**	**#iters**	**#DSSR**	**#labels**
C1_210	26,247.0	616	184	**8.1**	19,180	26,247.0	**518**	**165**	8.4	**18,126**
C1_2_3	**26,723.2**	**34,047**	498	**2.0**	**32,563**	26,721.7	95,535	**458**	2.2	33,154
C1_2_4	26,256.0	724	**283**	4.2	34,120	26,256.0	**626**	298	**3.6**	**32,310**
C1_2_9	26,396.0	**185**	55	11.1	8,836	26,396.0	191	**48**	**10.9**	**8,671**
C2_2_1	19,152.8	**413**	**70**	1.0	**455**	**19,156.4**	434	98	1.0	475
C2_2_2	**18,514.0**	12,368	1,415	1.2	5,863	18,495.7	**7,666**	**1,285**	1.2	**4,424**
C2_2_5	18,696.0	2,008	561	1.7	**1,602**	18,696.0	**1,913**	**523**	1.6	1,635
C2_2_6	18,448.0	4,111	628	**2.3**	3,102	18,448.0	**3,372**	**498**	2.5	**2,756**
C2_2_7	18,422.0	4,968	737	**2.2**	4,033	18,422.0	**4,234**	**600**	2.4	**3,850**
C2_2_8	18,137.0	4,633	577	2.7	5,321	18,137.0	**4,282**	**561**	2.6	**4,752**
C2_2_9	18,150.0	11,608	977	2.2	**10,730**	18,150.0	**10,600**	**837**	2.3	11,053
R1_210	**32,794.9**	2,140	417	2.2	29,849	32,793.1	**1,923**	**399**	2.2	**28,930**
R1_2_4	30,364.6	**18,415**	533	2.0	**127,556**	**30,367.1**	28,779	**526**	2.0	130,872
R1_2_5	**40,457.1**	427	253	1.1	7,141	40,451.6	**331**	**221**	1.1	**6,848**
R1_2_6	**35,456.9**	1,995	461	1.6	**29,240**	35,455.3	**1,835**	**426**	1.6	29,301
R1_2_7	31,392.8	**2,506**	**422**	2.1	**54,672**	**31,393.6**	3,797	442	**2.0**	55,468
R1_2_8	29,328.1	**4,897**	451	2.4	100,373	**29,328.5**	5,961	**447**	2.3	**99,820**
R1_2_9	**37,270.4**	750	308	1.5	13,958	37,267.6	**691**	**294**	1.6	**13,053**
R2_2_1	34,680.0	1,851	**230**	1.6	**452**	34,680.0	**1,794**	282	**1.5**	456
R2_2_2	30,082.0	5,271	594	**2.1**	**2,213**	30,082.0	**4,735**	**508**	2.2	2,229
R2_2_5	30,609.8	**4,836**	**1,302**	1.3	**2,128**	**30,611.0**	4,998	1,343	1.3	2,233
R2_2_6	26,748.4	**13,208**	**1,972**	1.5	6,390	**26,750.5**	14,960	2,536	**1.3**	**6,206**
R2_2_9	28,433.0	4,994	911	1.6	2,725	28,433.0	**4,747**	**847**	1.6	**2,583**
RC1_2_1	34,993.5	**634**	**312**	1.4	10,372	**34,994.6**	658	318	1.4	**10,135**
RC1_2_2	32,072.6	**28,285**	**495**	1.8	**41,646**	32,072.5	43,875	497	1.8	42,617
RC1_2_6	32,938.2	**5,123**	**503**	2.7	**36,610**	**32,941.2**	6,784	544	**2.5**	38,856
RC1_2_8	**30,499.5**	59,534	543	3.1	133,139	30,497.7	**57,887**	**534**	3.1	**130,149**
RC2_2_1	**27,966.5**	2,242	682	1.7	**909**	27,961.8	**2,194**	**571**	1.8	926
RC2_2_2	24,780.0	31,302	4,948	1.3	**6,565**	24,780.0	**25,755**	**4,177**	1.3	6,745
RC2_2_5	24,914.0	**7,532**	**1,817**	1.6	14,139	24,914.0	7,993	2,053	1.5	**13,719**
RC2_2_6	24,951.0	7,288	2,026	1.6	8,280	24,951.0	**6,431**	**1,920**	1.5	**7,976**
#Best	23/31	12/31	10/31	25/31	14/31	23/31	19/31	21/31	26/31	17/31