Branch-price-and-cut algorithms for integrated optimisation problems in transportation and healthcare

Matteo Petris

CENTRALE LILLE

THÈSE

Présentée en vue d'obtenir le grade de

DOCTEUR

En

Spécialité : Informatique

Par

Matteo PETRIS

DOCTORAT DELIVRE PAR CENTRALE LILLE

Titre de la thèse :

Algorithmes de branch-price-and-cut pour des problèmes d'optimisation intégrés en transport et santé

Branch-price-and-cut algorithms for integrated optimisation problems in transportation and healthcare

Soutenue le 20 septembre 2023 devant le jury d'examen :

Président	M. Yves CRAMA	Professeur, HEC Liège, Belgique
Rapporteur	Mme Renata MANSINI	Professeur, Università degli Studi di Brescia, Italie
Rapporteur	M. Stefan RØPKE	Professeur, Danmarks Tekniske Universitet, Danemark
Examinateur	M. Roberto ROBERTI	Chercheur, Università degli Studi di Padova, Italie
Directeur de thèse	M. Frédéric SEMET	Professeur, Centrale Lille, France
Co-encadrant	Mme Claudia ARCHETTI	Professeur, ESSEC Paris, France
Co-encadrant	M. Diego CATTARUZZA	Maître de conférences, Centrale Lille, France
Co-encadrant	M. Maxime OGIER	Maître de conférences, Centrale Lille, France

Thèse préparée au Centre de Recherche en Informatique, Signal et Automatique de Lille
CRIStAL - UMR CNRS 9189 et au Centre Inria de l'Université de Lille - Centrale Lille
École Doctorale MADIS 631

Acknowledgements

First, I would like to thank my supervisors Frédéric Semet, Claudia Archetti, Diego Cattaruzza and Maxime Ogier for their continuous support and guidance. In particular, I would like to thank Frédéric for having directed the thesis sharply and Claudia for her willingness even from far away. There are no words to express how much I appreciated the patience and the daily support of Diego and Maxime. I tried to learn as much as I could from you all.

I would like to express my gratitude to the PhD jury committee: Renata Mansini, Stefan Røpke, Yves Crama and Roberto Roberti. Their comments helped to improve the thesis and the discussion during the defense was interesting and full of insights. Especially, I would like to thank Yves Crama who was not appointed as reviewer, but reviewed the thesis thoroughly and Renata Mansini whose kindness relieved the stress before the defense.

In addition, I would like to thank Martine Labbé, Marius Roland, Martin Schmidt, Andrea Schaerf, Luca Di Gaspero and Roberto Maria Rosati with whom I had the pleasure to work on side projects during the PhD. For the same reason, I also want to thank Paola Pellegrini, Bianca Pascariu, Federico Naldini, Sonia Sobieraj Richard, Grégory Marliere and Joaquin Rodriguez from the COSYS-LEOST laboratory in Lille.

Special thanks are due to Raffaele Pesenti, without whom I would not have started the PhD and to Franca Rinaldi whose words I have been carrying for all these years.

Finally, I thank my colleagues and friends in the INOCS team: Luis S., Nathalia, Juan, Francesco, Wenjiao, Luis R., Gael, Pablo, Tifaout, Mathieu, Arnaud, Yuan, Wenjuan and Daniel.

ACKNOWLEDGEMENTS

Contents

List of Figures

LIST OF FIGURES

List of Tables

LIST OF TABLES

Chapter 1

Introduction

Mathematical optimisation provides tools to help decision-makers in solving problems arising in different fields of applications. Recent advances in the exact algorithms allow to solve large instances of well-known problems to optimality, even if they are theoretically NP-hard. Examples of such problems are the *Travelling Salesman Problem* or the *Capacitated Vehicle Routing Problem*. Nowadays, the more challenging problems are those that integrate two optimisation problems or those whose resolution entails solving other optimisation problems, which are usually complex on their own. In the first case, the problem either integrates problems at two different decision levels, e.g., tactical and operational, or two subproblems at the same decision level. In the second case, solving the main optimisation problem requires the computation of some values obtained by the resolution of other optimisation problems.

Such problems may arise in different fields of application, here, we focus on transportation and healthcare areas. In transportation, we consider an integrated problem which integrates two problems at the same operational level. Precisely, we consider a problem arising in the short and local food supply chain (Berti & Mulligan, 2016): high-quality agricultural products need to be delivered from medium-sized farms to customers whose primary concerns are product quality and traceability (King *et al.*, 2014). Such supply chains may involve several suppliers (farms) and customers, however, their locations have to be within a maximal distance of less than 100km (Blanquart *et al.*, 2010). In this context, suppliers produce several agricultural products, such as fruits, vegetables, and/or meat. In addition, such types of products may only be provided by some of the suppliers, but they may differ from one to the other. Similarly, customers have a demand for different types of products. Hence, these supply chains commonly

involve intermediate facilities, known as distribution centres, whose role is to consolidate the products before the deliveries and, consequently, reduce the transportation costs (Berti & Mulligan, 2016). Indeed, in this application, suppliers are interpreted as small-sized farms which do not have the resources to manage the transportation and logistics on their own. Hence, commonly, the collection and delivery of the products are coordinated by a central decision-maker embodied by, e.g., an association of farmers or a political authority. The central decision-maker aims to minimise the total transportation cost of the system, which arises from the following operations. Suppliers possess their own vehicles with large capacities to bring agricultural products to the distribution centres via direct trips. Once demand has been consolidated at each distribution centre, it is delivered to customers by a fleet of smaller capacity vehicles managed by the distribution centre itself. The vehicles follow routes that begin and end at the distribution centre to which they belong.

The optimisation problem arising in this specific application is the *Multi-Commodity two-echelon Distribution Problem* (MC2DP), a routing problem defined on two echelons where the multiple commodities have to be considered explicitly. Indeed, only some of the suppliers provide the same commodities, and the demands of the customers may be delivered by different vehicles.

In healthcare applications, we consider an integrated problem where the main problem is enriched with additional conditions that entail solving other optimisation problems. Precisely, we consider a problem arising in the context of kidney transplants which involve living donors. Such practice gives patients affected with a severe kidney disease an additional transplant option when a kidney from a deceased donor is not promptly available (Lentine *et al.*, 2023; Nemati *et al.*, 2014). Nowadays, it is common that each country runs its own *kidney exchange programme* to coordinate the kidney transplants between patients and living donors affiliated with all the hospitals of that country (see, e.g., Biró *et al.*, 2019a). Such programmes are often run periodically, typically every 3 or 4 months. One of the major concerns in kidney transplantation is the risk of rejection. To decrease this risk, transplants are performed only between patients and donors, compatible with each other according to several medical requirements (Kälble *et al.*, 2005). Finding patients and donors who meet such requirements is often challenging, even when the transplants are managed at the country level. Hence, in recent years, some countries have started to collaborate by merging their pool of patients and donors to increase the possibilities of performing more transplants and/or

transplants of better quality (see, e.g., Böhmig *et al.*, 2017; Scandiatransplant, 2023; Valentín *et al.*, 2019). As for the single countries, programmes involving multiple countries are run periodically. Under this collaborative setting, other than determining the transplants to perform, one aims to ensure: (i) the stability of the system at each run: no country should be incentivised to leave the system; (ii) the fairness of the system: the disparities in the number or quality of the transplants assigned to the countries are smoothed over the course of the programme. We call the underlying optimisation problem *Iterative International Kidney Exchange Problem* (IIKEP).

Both the MC2DP and the IIKEP fall in the class of integrated optimisation problems. The MC2DP integrates the activities of the collection and delivery echelons at the operational level. In addition, the delivery echelon is composed of different delivery problems, one per distribution centre, referred to as the *Commodity constrained Split Delivery Vehicle Routing Problem* (C-SDVRP). The C-SDVRP is an extension of the CVRP, which deals explicitly with multiple commodities. Each customer requires several commodities which are compatible and can be mixed in the vehicles. To reduce the cost, the demand of a customer can be served by several vehicles, but for convenience reasons, the delivery of a single commodity cannot be split. The IIKEP integrates the well-known *Kidney Exchange Problem* (KEP) with stability and fairness conditions. The KEP aims to determine a set of kidney transplants in a pool of patients and donors such that the number or the quality of the transplants is maximal.

Integrated optimisation problems may be tackled by means of different methods: metaheuristics, dedicated heuristics with or without a performance guarantee, or exact algorithms. The aim of this thesis is to design exact algorithms or methods which are able to provide good bounds on the optimal values of problem instances, that is heuristic algorithms with a performance guarantee. Well-known techniques to achieve such results are based on column generation and the Branch-Price-and-Cut (BPC) paradigm (Barnhart *et al.*, 1998). The main challenge in devising efficient algorithms for integrated problems lies in the ability to solve the core problems, in our case the C-SDVRP and the KEP, very efficiently.

Contributions

In what follows, we summarise the main contributions we achieve in this thesis.

1. To solve the C-SDVRP very efficiently, we devise a heuristic algorithm with a performance guarantee based on column generation. The algorithm embeds a new pricing heuristic dedicated to the multi-commodity aspect of the problem. Extensive experiments on benchmark instances from the literature show that our algorithm provides several new (best-known) solutions and significantly improves the computational time compared with a state-of-the-art heuristic for the C-SDVRP. The new pricing heuristic helps in reducing computational time.

2. For the MC2DP, we propose an extended formulation based on the one for the C-SDVRP, and we develop an exact BPC algorithm to solve the problem. The algorithm includes two families of robust valid inequalities which exploit the multi-commodity aspect of the problem. We test the BPC algorithm on benchmark instances previously introduced in the literature. Almost 60% of the instances are solved to optimality, while the remaining ones are left with an average positive gap equal to 2.1%.

3. In order to solve the KEP very efficiently, we propose a BPC algorithm which includes two families of non-robust valid inequalities. We assess the performances of the proposed BPC on three sets of benchmark instances from the literature. Our BPC is the first to be tested on instances for the KEP with different characteristics. For each set, we test the BPC against a reference algorithm. The BPC produces comparable results with the literature on the easiest set of instances and outperforms the results on the remaining two sets.

4. For the IIKEP, we propose an iterative procedure, where, the problem arising at each iteration is a KEP with stability and fairness conditions. Such a problem is solved by adapting the BPC algorithm we designed for the KEP. We provide a managerial analysis of the impact of the stability and fairness conditions on the solutions. In the analysis of the experiments, we show that the instability of the system is heavily reduced and the deviations w.r.t. an ideal fair scenario are small. Remark that under this collaborative setting, the improvement in terms of medical benefit is great. Compared to such improvement, the price to pay for the stability and fairness conditions is small. Hence, under these assumptions, countries are incentivised to join an international kidney exchange programme.

Structure of the thesis

In this section, we summarise the content of the thesis in more detail. The thesis is composed of seven chapters, introduction included. Table 1.1 reports the main information about the chapters' content. Each row corresponds to a chapter. The columns specify the application associated with the chapter content, the name of the problem and the method applied to solve it. We use the notation CG for column generation.

Table 1.1: Content of the chapters.

	application	name of the problem	method
Chapter 2	-	general	tutorial on BPC algorithms
Chapter 3	transportation	C-SDVRP	CG heuristic with performance guarantee
Chapter 4		MC2DP	BPC algorithm
Chapter 5	healthcare	KEP	BPC algorithm
Chapter 6		IIKEP	BPC algorithm

In what follows, we describe the content of each chapter in more detail.

- In Chapter 2, we review the main techniques to embed in a BPC algorithm to solve a generic class of problems where both our applications fall. The presentation is pedagogical-oriented, with several examples and implementation details to help understand these techniques.

- Chapters 3 and 4 are devoted to the application in transportation. Precisely, in Chapter 3, we study a heuristic with a performance guarantee based on column generation for the C-SDVRP. Then, in Chapter 4, we present a BPC algorithm for the MC2DP, which exploits the work done for the C-SDVRP.

  Chapter 3 has been submitted to *Networks*. Chapter 4 has been submitted to *Computers & Industrial Engineering*.

- Chapters 5 and 6 are dedicated to the application in healthcare. In Chapter 5, we develop an efficient BPC algorithm to solve the KEP. Then, in Chapter 6, we devise an iterative procedure to solve the IIKEP. At each iteration of the procedure, a KEP enriched with stability and fairness conditions is solved by a modified version of the BPC algorithm designed to solve the KEP.

- Finally, we discuss the main contributions of the thesis, and we provide future research directions.

Chapter 2

A tutorial on Branch-Price-and-Cut algorithms for vehicle routing-like problems

2.1 Introduction

In this chapter, we provide an overview on Branch-and-Price-and-Cut (BPC) algorithms to solve a generic class of problems whose aim is to find a set of feasible paths in a graph from a given source to a given destination such that all the vertices of the graph are visited at least (or at most) once while a cost function associated with the paths is minimised (or maximised). A path is said to be *feasible* if it respects some resource consumption constraints, where the resource type is a problem-specific feature. The presentation of the concepts is done for a minimisation problem, knowing that they can be trivially adapted to the maximisation case. A typical example of a problem which falls in this class is the *Capacitated Vehicle Routing Problem* (CVRP). In the CVRP, we are given a depot and a set of customers $\mathcal{J}$ having a positive demand $D_j > 0$, $j \in \mathcal{J}$. A fleet of K homogeneous and capacitated vehicles of capacity Q performs the deliveries from the depot to the customers. The objective of the problem is to determine a set of K feasible paths such that all customers are visited exactly once, and the total cost is minimised. A feasible path starts and ends at the depot and has to respect the capacity constraint, i.e., the sum of demands of customers visited in the path has to be lower than or equal to the capacity of the vehicles.

Deriving *compact formulations* for the problems in this class usually leads to weak

linear relaxations. Hence, standard branch-and-bound algorithms are rather inefficient. For this reason, it is common to apply *Dantzig-Wolfe decomposition* to model these problems using an *extended formulation* where the exponentially-many variables correspond to the feasible paths. Remark that the number of variables is exponential w.r.t. to the size of the instance, e.g., in the CVRP variables correspond to routes which are exponentially-many w.r.t. the number of customers. To handle the exponential number of variables in such formulations, methods based on column generation are commonly adopted. Precisely, several state-of-the-art exact algorithms are based on a BPC paradigm.

In the first decade of the 2000's, surveys on BPC algorithms started to appear in the literature. Ribeiro *et al.* (2002) provides implementation details for accelerating column generation methods for vehicle routing and crew scheduling problems. The book of Desaulniers *et al.* (2006) reviews BPC algorithms for different applications. The first chapter of this book (Desrosiers & Lübbecke, 2005) focuses on a didactic introduction to BPC algorithms emphasising the relation between Dantzig-Wolfe decomposition and Lagrangian relaxation. A similar focus is considered by Lübbecke & Desrosiers (2005). A tutorial on BPC algorithms for vehicle routing problems is proposed by Feillet (2010). The author provides insights on how to design a BPC algorithm. However, he does not consider implementation details.

More recently, two works by Costa *et al.* (2019) and Pessoa *et al.* (2020), whose aim is similar to the one of this chapter, appeared in the literature. Precisely, Costa *et al.* (2019) propose a survey which covers several modeling and methodological contributions made over the years on BPC algorithms for *Vehicle Routing Problems* (VRPs). Whereas, Pessoa *et al.* (2020) present their generic BPC solver for vehicle routing and related problems along with the state-of-the-art techniques embedded in it. The large amount of contributions reviewed in both these works allows only a high-level presentation. The lack of implementation details is often critical when one wants to develop a well-performing BPC. Therefore, in this chapter, we focus on fewer of these state-of-the-art techniques and provide a more pedagogical-oriented presentation of them, following to some extent the tutorial of Feillet (2010). Specifically, we give more room to examples and implementation details (as in Ribeiro *et al.* (2002)), which are sometimes necessary to understand such techniques and how they interact.

The remainder of the paper is organised as follows. In Section 2.2, we provide a generic description of a BPC algorithm by presenting its main components: the

Branch-and-Bound algorithm, the column generation procedure and the Branch-and-Cut algorithm*. In Section 2.3, we formally describe the class of problems we consider and provide a generic extended formulation. Finally, in Section 2.4, we present the main techniques that might be implemented in a BPC algorithm to solve the problems of the class we consider. Finally, Section 2.5 reports some final remarks.

2.2 From Branch-and-Bound to Branch-Price-and-Cut

In this section, we describe the building blocks of a Branch-Price-and-Cut (BPC) algorithm, namely the Branch-and-Bound algorithm (B&B), the column generation procedure and the Branch-and-Cut algorithm (B&C). Precisely, BPC algorithms are used to solve integer programming models defined by means of an exponential number of variables. They can be seen as variants of the B&B algorithm where the *bounding* step is performed using a column generation procedure. As in a B&C algorithm, the value of the relaxation at each node of the branch and bound tree may be strengthened by including cuts.

2.2.1 Branch-and-Bound algorithm

The Branch-and-Bound (B&B) algorithm was introduced in 1960 by Land & Doig (1960) to solve integer programming models exactly. Without loss of generality, we consider an integer programming model of the following form

$$[\mathrm{M}^{IP}] \quad \min \; cx \tag{2.1}$$

$$\text{s.t. } Ax \geq b \tag{2.2}$$

$$x \in \mathbb{Z}^n_{\geq 0}, \tag{2.3}$$

where $c \in \mathbb{R}^n$ is a cost vector, A is the $m \times n$ constraints matrix, $b \in \mathbb{R}^m$ is the vector of right hand-sides and $x \in \mathbb{Z}^n_{\geq 0}$ is the vector of integer variables. We denote by $\mathcal{S} = \{x \in \mathbb{Z}^n_{\geq 0} : Ax \geq b\}$ the feasible region of $[\mathrm{M}^{IP}]$. The idea behind the B&B algorithm is to perform an implicit and efficient exploration of set $\mathcal{S}$. Precisely, problem $[\mathrm{M}^{IP}]$ is recursively decomposed in subproblems such that the union of their feasible regions is at least a *covering* of $\mathcal{S}$ (*branching*). The linear relaxation of these

*The presentation of these algorithms is inspired by the course notes of Optimisation I and II by Franca Rinaldi, Università degli Studi di Udine.

smaller subproblems is then solved instead of $[\mathrm{M}^{IP}]$. Dominance conditions allow us to avoid solving some of the linear relaxations without giving up the correctness of the algorithm (*bounding*). The recursive procedure at the base of a B&B algorithm can be represented by a search tree, which is referred to as *branch-and-bound search tree*.

The pseudocode of a B&B algorithm is reported in Algorithm 1. The following notation is used. We denote z^* the optimal value of $[\mathrm{M}^{IP}]$, and $\bar{x}$ and $\bar{z}$ the current best integer solution of $[\mathrm{M}^{IP}]$ and its value, respectively. Let $\mathcal{L}$ be the list of problems to be solved, i.e., of problems associated with nodes of the branch-and-bound tree yet to be explored. Each problem $\mathcal{Q}$ in list $\mathcal{L}$ is associated with a lower bound $\underline{z}_{\mathcal{Q}}$. Value $\underline{z} = \min\{\underline{z}_{\mathcal{Q}} : \mathcal{Q} \in \mathcal{L}\}$ is the lower bound of z^* computed during the exploration of the branch-and-bound tree.

Algorithm 1: Pseudocode of a B&B algorithm.

Input: $\mathcal{L} := \{[M^{IP}]\}$, $\underline{z}_{[M^{IP}]} := -\infty$, $\underline{z} := -\infty$, $\bar{x} := NULL$, $\bar{z} := \infty$

while $\mathcal{L} \neq \emptyset$ **do**
  select a problem $\mathcal{Q}$ from $\mathcal{L}$;
  remove $\mathcal{Q}$ from $\mathcal{L}$;
  if $\underline{z}_{\mathcal{Q}} < \bar{z}$ **then** // pruning
    solve the linear relaxation ($\mathcal{RQ}$) of $\mathcal{Q}$; // bounding
    if ($\mathcal{RQ}$) *admits a feasible solution* **then**
      let $\tilde{x}_{(\mathcal{RQ})}$ and $\tilde{z}_{(\mathcal{RQ})}$ be an optimal solution of ($\mathcal{RQ}$) and its value;
      if $\tilde{z}_{(\mathcal{RQ})} < \bar{z}$ **then** // pruning
        if *solution* $\tilde{x}_{(\mathcal{RQ})}$ *is integer* **then**
          update $\bar{x} := \tilde{x}_{(\mathcal{RQ})}$ and $\bar{z} := \tilde{z}_{(\mathcal{RQ})}$;
        else // branching
          let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be the problems associated with the two selected branches;
          add $\mathcal{Q}_1$ and $\mathcal{Q}_2$ to $\mathcal{L}$;
          set $\underline{z}_{\mathcal{Q}_1} := \tilde{z}_{\mathcal{RQ}}$ and $\underline{z}_{\mathcal{Q}_2} := \tilde{z}_{(\mathcal{RQ})}$;
          update $\underline{z} := \min\{\underline{z}_{\mathcal{Q}} : \mathcal{Q} \in \mathcal{L}\}$;
        end
      end
    end
  end
end
update $\underline{z} := \bar{z}$;

The list of problems is initialized with problem $[\mathrm{M}^{IP}]$. The lower bound $\underline{z}_{[M^{IP}]}$ associated with $[\mathrm{M}^{IP}]$ is set to minus infinity. The same initialization is done for lower bound $\underline{z}$ on z^*. The current best incumbent integer solution $\bar{x}$ and its value $\bar{z}$ are set to $NULL$ and infinity, respectively.

At each iteration of the algorithm, a problem $\mathcal{Q}$ is selected and removed from list $\mathcal{L}$. First, a pruning step is performed to verify whether problem $\mathcal{Q}$ is worthy to be processed. Precisely, we check whether lower bound $\underline{z}_{\mathcal{Q}}$ is strictly less than the best incumbent value $\bar{z}$. Indeed, $\underline{z}_{\mathcal{Q}} \geq \bar{z}$ means that no feasible solution in the subtree

rooted at the current node (problem $\mathcal{Q}$) can improve the current best incumbent value. Then, the bounding step is performed, i.e., the linear relaxation ($\mathcal{RQ}$) of $\mathcal{Q}$ is solved, e.g., using the simplex method. If ($\mathcal{RQ}$) does not admit any feasible solution, we move to the next iteration. Otherwise, we denote by $\tilde{x}_{(\mathcal{RQ})}$ and by $\tilde{z}_{(\mathcal{RQ})}$ an optimal solution and the optimal value of ($\mathcal{RQ}$), respectively. A second pruning step is performed to avoid exploring non promising subtrees: again, we check whether $\tilde{z}_{(\mathcal{RQ})} < \bar{z}$. Let us suppose this condition is verified. Now, if solution $\tilde{x}_{(\mathcal{RQ})}$ is integer, we found a new best incumbent solution for [M^{IP}]. Otherwise, branching rules are applied to build two subproblems of $\mathcal{Q}$, denoted as $\mathcal{Q}_1$ and $\mathcal{Q}_2$, which are included in list $\mathcal{L}$. Precisely, let $\tilde{x}_{(\mathcal{RQ})i}$, $i \in \{1, \ldots, n\}$, be a fractional value in solution $\tilde{x}_{(\mathcal{RQ})}$. Then, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ are the two subproblems of $\mathcal{Q}$ characterised by the inclusion of constraints $x_i \leq \lfloor \tilde{x}_{(\mathcal{RQ})i} \rfloor$ and $x_i \geq \lceil \tilde{x}_{(\mathcal{RQ})i} \rceil$, respectively. Commonly, the fractional value in the solution is selected according to maximal fractional part, i.e., $\max_{i=1,\ldots,n}\{\min\{\tilde{x}_{(\mathcal{RQ})i} - \lfloor \tilde{x}_{(\mathcal{RQ})i} \rfloor, \lceil \tilde{x}_{(\mathcal{RQ})i} \rceil - \tilde{x}_{(\mathcal{RQ})i}\}\}$. The lower bounds $\underline{z}_{\mathcal{Q}_1}$ and $\underline{z}_{\mathcal{Q}_2}$ associated with these two subproblems are set to be equal to $\tilde{z}_{(\mathcal{RQ})}$. Here, the lower bound $\underline{z}$ on z^* is also updated. When all the problems in $\mathcal{L}$ are processed, such lower bound is set to be equal to the incumbent value $\bar{z}$ and incumbent solution $\bar{x}$ is proven to be optimal for [M^{IP}].

We conclude the section with some remarks.

Remark 2.1 (Exploration of the branch-and-bound tree)**.** *The order in which problems are stored in list $\mathcal{L}$ models the way the branch-and-bound tree is explored. Common exploration strategies are depth-first search, breadth-first search and best-first search. In depth-first search, list $\mathcal{L}$ is managed using* Last In First Out *(LIFO) rule: at each iteration, we consider the last problem inserted in $\mathcal{L}$. In breadth-first search, list $\mathcal{L}$ is managed according to a* First In First Out *(FIFO) rule: at each iteration, we consider the first problem inserted in $\mathcal{L}$. In best-first search, problems in list $\mathcal{L}$ are arranged by increasing value of lower bounds and at each iteration, the problem with the minimal lower bound is considered. The interested reader may refer to Libralesso (2020) for a more comprehensive study on branch-and-bound tree exploration strategies.*

Remark 2.2 (Early termination of the algorithm)**.** *Note that if the B&B algorithm is terminated before list $\mathcal{L}$ is empty, lower and upper bounds $\underline{z}$ and $\bar{z}$ provide an estimation of the error arising by the early termination. Precisely, the error of accepting the best incumbent solution in lieu of the optimal one is $\bar{z}/z^*$, which is bounded from above by value $\bar{z}/\underline{z}$.*

2.2.2 Column generation

In this section, we present the column generation procedure as a variant of the simplex method to solve linear programming models with an exponential number of variables (Gilmore & Gomory, 1961). In this context, it is impossible to enumerate all the variables a priori to solve the model using the standard simplex method. The idea behind the column generation procedure is to iteratively generate only the variables needed to solve the model to optimality. As in the simplex method, the search for such meaningful variables is guided by information from the dual problem. We now recall some basic concepts about the standard simplex method.

Without loss of generality, let us consider a linear programming model expressed in standard form:

$$[\mathrm{M}^{LP}] \quad \min \ cx \tag{2.4}$$

$$\text{s.t. } Ax = b \tag{2.5}$$

$$x \in \mathbb{R}^n_{\geq 0}, \tag{2.6}$$

where $c \in \mathbb{R}^n$ is a cost vector, A is the $m \times n$ constraints matrix, $b \in \mathbb{R}^m$ is the vector of right hand-sides and $x \in \mathbb{R}^n$ is the variables vector. Without loss of generality, we suppose that $m \leq n$ and the rank of matrix A is equal to m. The simplex method exploits the fundamental theorem of linear programming: $[\mathrm{M}^{LP}]$ is either infeasible, unbounded or its optimum is attained on at least one extreme vertex of the feasible region of $[\mathrm{M}^{LP}]$, i.e., of the polyhedron $\mathcal{F} = \{x \in \mathbb{R}^n_{\geq 0} : Ax = b\}$. In the first phase, the method detects if $[\mathrm{M}^{LP}]$ is infeasible ($\mathcal{F} = \emptyset$) or, otherwise, it determines one of the extreme vertices of $\mathcal{F}$. Then, starting from such vertex, the vertices of $\mathcal{F}$ are iteratively explored until $[\mathrm{M}^{LP}]$ is proved to be unbounded or optimality conditions are met.

It is out of the scope of this section to provide a thorough description of the simplex method. We focus only on showing how the extreme vertices of $\mathcal{F}$ can be characterised from an algebraic point of view and on the conditions to state that one of these vertices is optimal. We introduce the following notation and definitions. Given $j \in \{1, \ldots, n\}$, we denote A^j the j-column of matrix A. A *basis* is a subset of the columns indices $\mathcal{B} \subseteq \{1, \ldots, n\}$, $|\mathcal{B}| = m$ such that the submatrix $A_\mathcal{B} = (A^j)_{j \in \mathcal{B}}$ of A obtained by considering the columns of A with index in $\mathcal{B}$ is non-singular. Matrix $A_\mathcal{B}$ is called *basic matrix*. Given a basis $\mathcal{B}$, the columns of A can be permuted so that $A = (A_\mathcal{B}, A_\mathcal{N})$, where $A_\mathcal{N} = (A^j)_{j \in \mathcal{N}}$ and $\mathcal{N} = \{1, \ldots, n\} \setminus \mathcal{B}$. Analogously, we write $c = (c_\mathcal{B}, c_\mathcal{N})$

and $x = (x_{\mathcal{B}}, x_{\mathcal{N}})$, where $x_{\mathcal{B}}$ and $x_{\mathcal{N}}$ are called *basic variables* and *non-basic variables*, respectively.

Remark that a basis $\mathcal{B}$ always exists since the rank of A is equal to m. In addition, $A_{\mathcal{B}}$ is non-singular, hence, model $[\mathrm{M}^{LP}]$ can be rewritten in the following manner:

$$[\mathrm{M}^{LP}] \quad \min \; c_{\mathcal{B}} x_{\mathcal{B}} + c_{\mathcal{N}} x_{\mathcal{N}} \tag{2.7}$$

$$\text{s.t. } x_{\mathcal{B}} = A_{\mathcal{B}}^{-1} b - A_{\mathcal{B}}^{-1} A_{\mathcal{N}} x_{\mathcal{N}} \tag{2.8}$$

$$x_{\mathcal{B}} \in \mathbb{R}^m_{\geq 0}, \; x_{\mathcal{N}} \in \mathbb{R}^{n-m}_{\geq 0}. \tag{2.9}$$

It is easy to see that vector $\hat{x} = (\hat{x}_{\mathcal{B}} = A_{\mathcal{B}}^{-1} b, \hat{x}_{\mathcal{N}} = \mathbf{0})$ satisfies $A\hat{x} = b$, where we denote by $\mathbf{0}$ the vector of zeros of the required size. In addition, if $\hat{x}_{\mathcal{B}} \geq \mathbf{0}$ holds, then $\hat{x}$ belongs to the feasible region of $[\mathrm{M}^{LP}]$. We call $\hat{x}$ *basic feasible solution*. It is well-known that each basic feasible solution $\hat{x}$ corresponds to an extreme vertex of $\mathcal{F}$ and each extreme vertex of $\mathcal{F}$ corresponds to a basic feasible solution for at least one basis. This characterises the extreme vertices of $\mathcal{F}$ from an algebraic point of view.

Now, let us suppose that vertex $\hat{x}$ is visited at a certain iteration of the simplex method. We need to test if $\hat{x}$ is an optimal solution for $[\mathrm{M}^{LP}]$, i.e., if $c\hat{x} \leq cx$, for each feasible solution $x \in \mathcal{F}$, holds. For the algebraic characterisation of the vertices, it exists a basis $\mathcal{B}$ such that $\hat{x} = (\hat{x}_{\mathcal{B}} = A_{\mathcal{B}}^{-1} b, \hat{x}_{\mathcal{N}} = \mathbf{0})$. The cost of $\hat{x}$ is $c\hat{x} = c_{\mathcal{B}} A_{\mathcal{B}}^{-1} b$. In light of (2.8), the cost of any feasible solution x can be expressed in terms of basis $\mathcal{B}$ in the following manner:

$$cx = c_{\mathcal{B}} x_{\mathcal{B}} + c_{\mathcal{N}} x_{\mathcal{N}} = c_{\mathcal{B}} A_{\mathcal{B}}^{-1}(b - A_{\mathcal{N}} x_{\mathcal{N}}) + c_{\mathcal{N}} x_{\mathcal{N}} = c\hat{x} + (c_{\mathcal{N}} - c_{\mathcal{B}} A_{\mathcal{B}}^{-1} A_{\mathcal{N}}) x_{\mathcal{N}}.$$

Hence, since $x_{\mathcal{N}} \geq \mathbf{0}$, inequality $c\hat{x} \leq cx$ holds if $c_{\mathcal{N}} - c_{\mathcal{B}} A_{\mathcal{B}}^{-1} A_{\mathcal{N}} \geq \mathbf{0}$. Given $j \in \mathcal{N}$, we call $\bar{c}_j = c_j - c_{\mathcal{B}} A_{\mathcal{B}}^{-1} A^j$ the *reduced cost of* j. If it exists a non-basic variable (i.e. its index is in $\mathcal{N}$) such that $\bar{c}_j < 0$, then the current basic feasible solution is not optimal.

We highlight the following remarks, which are helpful to formally introduce the column generation procedure as a variant of the standard simplex method.

Remark 2.3. *The simplex method explores the extreme vertices of polyhedron $\mathcal{F}$. Such vertices are characterised as basic feasible solutions. The number of non-zero components in such solutions is at most equal to the number m of constraints. Indeed, the cardinality of a basis is m, and the value of all the non-basic variables, i.e., the ones with indices in $\mathcal{N}$, is set to zero. Hence, even if the number of variables n is exponential, only a limited number of them will appear with a positive value in a basic solution.*

Remark 2.4. *In the simplex method, the explicit knowledge of the entire constraint matrix is required only when the reduced costs have to be computed to prove the optimality of a basic feasible solution. Hence, it is clear that if the number of variables is exponential, this step needs to be performed "implicitly".*

Remark 2.5. *We explicit the relation between the simplex method and the dual problem $[D^{LP}]$ of $[M^{LP}]$:*

$$[D^{LP}] \max\ ub \tag{2.10}$$

$$s.t.\ uA \leq c \tag{2.11}$$

$$u \in \mathbb{R}^m. \tag{2.12}$$

Recall that the expression of the reduced cost of $j \in \mathcal{N}$ is $\bar{c}_j = c_j - c_{\mathcal{B}} A_{\mathcal{B}}^{-1} A^j$. If $\mathcal{B}$ is optimal for $[M^{LP}]$, it is easy to see that vector $\hat{u} = c_{\mathcal{B}} A_{\mathcal{B}}^{-1}$ is the optimal solution of $[D^{LP}]$ associated with $\hat{x} = (A_{\mathcal{B}}^{-1} b, \mathbf{0})$. Indeed, the feasibility of $\hat{u}$ follows from the optimality conditions on $\hat{x}$ ($c_{\mathcal{N}} - c_{\mathcal{B}} A_{\mathcal{B}}^{-1} A_{\mathcal{N}} \geq \mathbf{0}$):

$$\hat{u}A = \hat{u}(A_{\mathcal{B}}, A_{\mathcal{N}}) = (c_{\mathcal{B}} A_{\mathcal{B}}^{-1} A_{\mathcal{B}}, c_{\mathcal{B}} A_{\mathcal{B}}^{-1} A_{\mathcal{N}}) = (c_{\mathcal{B}}, c_{\mathcal{B}} A_{\mathcal{B}}^{-1} A_{\mathcal{N}}) \leq (c_{\mathcal{B}}, c_{\mathcal{N}}) = c.$$

Then, we observe that the value of solution $\hat{x}$ in $[M^{LP}]$ and the value of solution $\hat{u}$ in $[D^{LP}]$ are the same:

$$\hat{u}b = c_{\mathcal{B}} A_{\mathcal{B}}^{-1} b = (c_{\mathcal{B}}, c_{\mathcal{N}})(A_{\mathcal{B}}^{-1} b, \mathbf{0}) = c\hat{x}.$$

The reduced cost of $j \in \mathcal{N}$ can be stated in terms of the dual solution $\hat{u}$ as $\bar{c}_j = c_j - \hat{u}A^j$. Hence, there is no explicit need to know matrix $A_{\mathcal{B}}^{-1}$ to compute such reduced costs.

Now, we formally introduce the column generation procedure. We consider again model $[\mathrm{M}^{LP}]$ expressed as in (2.4)-(2.6), but, this time, we suppose that the number of variables n is exponential and that $[\mathrm{M}^{LP}]$ admits a feasible solution. In this context, model $[\mathrm{M}^{LP}]$ is called *Master Problem* (MP). As in the simplex method, we consider a subset of variables indices $\mathcal{K} \subseteq \{1, \ldots, n\}$ such that the restriction of model $[\mathrm{M}^{LP}]$ to the variables with indices in $\mathcal{K}$ admits a feasible solution. Such restriction is referred

to as Restricted Master Problem (RMP) and is modelled as follows:

$$[\mathrm{M}^{LP}(\mathcal{K})] \quad \min \; c_{\mathcal{K}} x_{\mathcal{K}} \tag{2.13}$$

$$\text{s.t.} \; A_{\mathcal{K}} x_{\mathcal{K}} = b \tag{2.14}$$

$$x_{\mathcal{K}} \in \mathbb{R}^{|\mathcal{K}|}_{\geq 0}, \tag{2.15}$$

where $A_{\mathcal{K}} = (A^j)_{j \in \mathcal{K}}$, $c_{\mathcal{K}} = (c_j)_{j \in \mathcal{K}}$ and $x_{\mathcal{K}} = (x_j)_{j \in \mathcal{K}}$.

Remark 2.6 (Dummy variables)**.** *Identifying a set $\mathcal{K}$ in order to ensure the feasibility of $[M^{LP}(\mathcal{K})]$ can be not trivial in some applications. The introduction of one or many dummy variables may come in handy to overcome this issue (see, e.g., Feillet, 2010). These variables must satisfy Constraints (2.14) and appear in the objective function with a very high cost. For example, in the case of the CVRP, a single dummy variable representing a path visiting all the customers can be considered. Such variables ensure the feasibility of the model. However, its associated path violates the capacity constraint. Hence, it must appear in the objective function with a very high cost.*

In addition, if a dummy variable still appears in the optimal solution at the end of the column generation procedure, the model $[M^{LP}]$ is proved to be infeasible.

The size of model $[\mathrm{M}^{LP}(\mathcal{K})]$ allows it to be solved by a standard simplex method. Let $\hat{x}'$ be one of its optimal solutions. We easily obtain a basic feasible solution $\hat{x}$ for model $[\mathrm{M}^{LP}]$ from $\hat{x}'$. Indeed, it suffices to assign the same value as $\hat{x}_{\mathcal{K}}$ to the variables with indices in $\mathcal{K}$ and value zero to the variables with indices in $\bar{\mathcal{K}} = \{1, \ldots, n\} \setminus \mathcal{K}$.

At this point, we want to verify whether $\hat{x}$ is an optimal solution of $[\mathrm{M}^{LP}]$. To do so, we exploit the optimality condition arising from the reduced costs and Remark 2.5. Let $\hat{u}'$ be the optimal solution of the dual of $[\mathrm{M}^{LP}(\mathcal{K})]$. We need to verify whether it exists a variable with index in $j \in \bar{\mathcal{K}}$ whose reduced cost $\bar{c}_j = c_j - \hat{u}' A^j$ is negative. If such an index exists, $\hat{x}$ is not proven optimal for $[\mathrm{M}^{LP}]$, and the corresponding non-basic variable has to be included in $[\mathrm{M}^{LP}(\mathcal{K})]$. At this point, a new iteration has to start.

Remark 2.7 (Column generation from the dual problem point of view)**.** *Note that problem $[D^{LP}]$ has an exponential number of constraints. Indeed, it has a constraint for each variable in $[M^{LP}]$. Hence, generating negative reduced cost columns for problem $[M^{LP}]$ converts into generating violated constraints for problem $[D^{LP}]$. Column generation can be interpreted as a Kelley's cutting plane algorithm (Kelley, 1960) from the point of view of the dual.*

As mentioned in Remark 2.4, verifying the optimality conditions requires knowledge of the entire constraint matrix A. However, verifying such conditions is equivalent to solving the optimisation problem $\bar{c}^* = \min\{\bar{c}_j : j \in \bar{\mathcal{K}}\} = \min\{c_j - \hat{u}'A^j : j \in \bar{\mathcal{K}}\}$, referred to as *pricing problem*. Indeed, if $\bar{c}^* \geq 0$, no negative reduced cost variable exists, and $\hat{x}$ is an optimal solution of $[\mathrm{M}^{LP}]$. Otherwise, the pricing problem identifies a new column A^j to include in matrix $A_{\mathcal{K}}$ defining the RMP. Such a column can typically be associated with an optimal solution for the pricing problem. Note that any column corresponding to a variable with negative reduced cost could also be added to $A_{\mathcal{K}}$.

It is clear that the optimality conditions can be verified implicitly if we are able to solve the pricing problem without enumerating all columns A^j, $j \in \bar{\mathcal{K}}$. Since $\hat{x}_{\mathcal{K}}$ is optimal for $[\mathrm{M}^{LP}(\mathcal{K})]$, the reduced cost of variables whose indices are in $\mathcal{K}$ are non-negative. We can state the pricing problem as $\bar{c}^* = \min\{\bar{c}_j : j \in \{1, \ldots, n\}\}$. Finally, the structure of the pricing problem depends on the particular problem under consideration. Hence, the performances of the column generation procedure highly rely on how efficiently we are able to solve such pricing problem.

Algorithm 2 reports the pseudocode of the column generation procedure. The RMP and the pricing problem are solved sequentially, while the latter provides negative reduced cost columns. Once the optimality conditions are met, the pricing problem certifies that the current basic feasible solution is optimal, and the procedure stops.

Algorithm 2: Column generation procedure

Input: Subset $\mathcal{K} \subseteq \{1, \ldots, n\}$ of columns.
do
  solve the RMP $[\mathrm{M}^{LP}(\mathcal{K})]$;
  get the optimal solution of dual of $[\mathrm{M}^{LP}(\mathcal{K})]$ and build pricing problem [PP];
  solve [PP] $\bar{c}^* = \min\{\bar{c}_j : j \in \{1, \ldots, n\}\}$;
  add negative reduced cost columns to $[\mathrm{M}^{LP}(\mathcal{K})]$;
while $\bar{c}^* < 0$;

2.2.3 Branch-and-Cut algorithm

In this section, we describe the Branch-and-Cut algorithm (B&C), introduced by Padberg & Rinaldi (1991), to solve exactly integer programming models, as, e.g., $[\mathrm{M}^{IP}]$ considered in Section 2.2.1.

B&C algorithms are variants of the B&B algorithm where the linear relaxation at each node of the branch-and-bound tree may be strengthened by including inequalities.

Hence, to be able to develop a B&C algorithm, we need to identify a priori a family $\mathscr{L}$ of *valid inequalities* for the feasible region $\mathcal{S}$ of $[\mathrm{M}^{IP}]$, i.e., inequalities satisfied by all the points in $\mathcal{S}$.

The procedure is very similar to the one of the B&B algorithm. In the following, we highlight the differences making references to Algorithm 1. First, among the inputs, we also provide a set $\mathscr{L}$ of valid inequalities. Then, every time a problem $\mathcal{Q}$ is selected from list $\mathcal{L}$, instead of solving its linear relaxation once (Algorithm 1, line 5), the iterative procedure described in Algorithm 3 starts. Precisely, each iteration entails solving the linear relaxation $(\mathcal{RQ})$ and the so-called *separation problem*, the latter only if the optimal solution $\tilde{x}_{(\mathcal{RQ})}$ of $(\mathcal{RQ})$ is fractional.

Remark 2.8 (Separation problem)**.** *In its general form, the separation problem can be formally stated as follows:*

Given a family $\mathscr{L}$ of inequalities defined in $\mathbb{R}^n$ and $\tilde{x} \in \mathbb{R}^n$, the aim of the separation problem is to determine whether $\tilde{x}$ satisfies all the inequalities in $\mathscr{L}$ or identify an inequality of $\mathscr{L}$ violated by $\tilde{x}$.

The computational complexity of the separation problem is related to the nature of the inequalities in $\mathscr{L}$. In a B&C algorithm, such a problem can be solved exactly or heuristically. Indeed, the role of valid inequalities is to strengthen the linear relaxations. The guarantee that the B&C algorithm terminates with an optimal solution for $[M^{IP}]$ is ensured by the branching rules.

Hence, in a B&C algorithm, the role of the separation problem is to identify at least one inequality in $\mathscr{L}$ which is violated by $\tilde{x}_{(\mathcal{RQ})}$, if it exists. Such inequality is then added to problem $\mathcal{Q}$ to cut $\tilde{x}_{(\mathcal{RQ})}$ from the feasible region of $(\mathcal{RQ})$. The next iteration starts. The procedure terminates if either solution $\tilde{x}_{(\mathcal{RQ})}$ is integer or it is fractional and the separation problem does not provide any violated inequality. In this latter case, branching rules are applied (Algorithm 1, line 12). Remark, including valid inequalities to cut the current optimal fractional solution of $(\mathcal{RQ})$ ensures that the value of $(\mathcal{RQ})$ at one iteration is at least as good as the one of the previous iteration. This means that the quality of the lower bound associated with the subproblems of $\mathcal{Q}$ increases. Hence, we also increase the chances for these subproblems to be pruned in future iterations and consequently, we may reduce the number of nodes of the branch-and-bound to be explored.

We conclude the section with a remark.

Algorithm 3: Pseudocode of a B&C algorithm.

```
   Input: Problem 𝒬, termination = false
1  while termination = false do
2  |  solve the linear relaxation (ℛ𝒬) of 𝒬;
3  |  if (ℛ𝒬) is infeasible then
4  |  |  termination = true;
5  |  else
6  |  |  let x̃ be an optimal solution of (ℛ𝒬);
7  |  |  if x̃ is integer then
8  |  |  |  termination = true;
9  |  |  else
10 |  |  |  solve the separation problem w.r.t. x̃ and ℒ;          // separation
11 |  |  |  if no inequality is detected then
12 |  |  |  |  termination = true;
13 |  |  |  else
14 |  |  |  |  add the detected inequality to (ℛ𝒬);
15 |  |  |  end
16 |  |  end
17 |  end
18 end
```

Remark 2.9 (Cutting plane method)**.** *In the description of the B&C algorithm, we do not make any assumption on the valid inequalities of $\mathscr{L}$. In case set $\mathscr{L}$ provides a complete external description of convex hull conv($\mathcal{S}$) of $\mathcal{S}$, the B&C algorithm reduces to a cutting plane method (Kelley, 1960). Such methods can be seen as a B&C algorithm where no branching is needed to ensure that an optimal solution of [M^{IP}] is found at termination. The correctness of the cutting plane method relies on a geometric observation: convex hull conv($\mathcal{S}$) is a polyhedron whose vertices are elements of $\mathcal{S}$. For these reasons, remark that in a cutting plane method, the separation problem must be solved exactly.*

2.2.4 Branch-Price-and-Cut algorithm

In this section, we consider the integer model [M^{IP}] defined over an exponential number of variables. We put together the methods described in Sections 2.2.1, 2.2.2 and 2.2.3 to present the Branch-Price-and-Cut (BPC) algorithm.

Classical branch-and-bound algorithms cannot efficiently solve formulation [M^{IP}] due to the exponential number of variables involved. To overcome this issue, Barnhart *et al.* (1998) proposed a Branch-Price-and-Cut (BPC) algorithm, a variant of the branch-and-bound algorithm to deal with integer programming models that have an exponential number of variables. Precisely, at each node of the branch-and-bound tree, the linear relaxation of the formulation [M^{IP}], referred to as the Master Problem, is solved by means of a column generation procedure. If the solution of the MP is frac-

tional, a separation problem dynamically identifies violated inequalities from a set $\mathscr{L}$ to insert in the MP. Then, the column generation procedure is repeated while violated valid inequalities are found or a termination condition is satisfied. As in a B&C algorithm, remark that including valid inequalities is not necessary to ensure the correctness of the algorithm (see Remark 2.8). However, they permit to provide better bounds and, consequently, reduce the exploration of the branch-and-bound tree. Differently from a B&C algorithm, including valid inequalities in the MP induces modifications in the pricing problem. Such modifications may be troublesome in case they modify the structure of the pricing problem and increase its complexity. Finally, branching rules are applied to ensure the integrality of the solution.

We report the pseudocode of a BPC algorithm in Algorithm 4.

Algorithm 4: Pseudocode of a BPC algorithm.

Input: $\mathcal{L} := \{[\mathrm{M}]^{IP}\}$, $\underline{z}_{[\mathrm{M}]^{IP}} := -\infty$, $\underline{z} := -\infty$, $\bar{x} := NULL$, $\bar{z} := \infty$

```
while L ≠ ∅ do
    select a problem Q from L;
    remove Q from L;
    let F = ∅ be the set of violated valid inequalities;
    if z_Q < z̄ then                                                    // pruning
        do
            insert inequalities in F in problem Q;
            set F = ∅;
            solve the linear relaxation (RQ) of Q ;                    // bounding by column generation
            if (RQ) admits a feasible solution then
                let x̃_(RQ) and z̃_(RQ) be an optimal solution of (RQ) and its value;
                if z̃_(RQ) < z̄ then                                     // pruning
                    if solution x̃_(RQ) is integer then
                        update x̄ := x̃_(RQ) and z̄ := z̃_(RQ);
                    else
                        detect violated valid inequalities and add them to F;   // separation
                        if F = ∅ then                                  // branching
                            let Q_1 and Q_2 be the problems associated with the two selected branches;
                            add Q_1 and Q_2 to L;
                            set z_Q1 := z̃_RQ and z_Q2 := z̃_(RQ);
                            update z := min{z_Q : Q ∈ L};
                        end
                    end
                end
            end
        while F ≠ ∅;
    end
end
update z := z̄;
```

2.3 Problem description and formulation

In this section, we provide a formal description and an extended formulation of the problem class we consider. Let $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ be a directed weighted graph, where $\mathcal{V} = \mathcal{J} \cup \{s, t\}$, $\mathcal{J}$ is a non-empty set, s and t are the source and destination vertices, respectively. Hence, $\mathcal{G}$ can be either a complete or a sparse graph. Each arc $(i, j) \in \mathcal{A}$ is associated with a cost $C_{ij} \geq 0$. In addition, we consider a unique resource whose maximal available quantity is denoted by $\bar{R}$. A consumption R_j of such a resource is associated with vertices $j \in \mathcal{J}$. Last, a path p in graph $\mathcal{G}$ starting at s and ending in t is *feasible* if the total amount of resource consumed along the path does not exceed the maximal available resource quantity, i.e., if $\sum_{j \in \mathcal{J}(p)} R_j \leq \bar{R}$, where $\mathcal{J}(p)$ is the subset of vertices visited by the path. Let $\mathcal{P}$ be the set of feasible paths in $\mathcal{G}$. The cost of a path $p \in \mathcal{P}$ is $C_p = \sum_{(i,j) \in \mathcal{A}(p)} C_{ij}$, where $\mathcal{A}(p)$ is the subset of arcs traversed by the path. Remark that we do not make any assumptions on the structure of the paths, e.g., elementarity constraints (a path cannot visit the same vertex more than once). Hence, collections $\mathcal{V}(p)$ and $\mathcal{A}(p)$ may include multiple times a vertex and an arc, respectively. The constraints imposed on a path are particularly important from an algorithmic point of view, so we will discuss this issue later in section 2.4.2.

The class of problems we consider can be stated as follows. It is to determine a set of feasible paths from s to t in graph $\mathcal{G}$ such that the vertices of $\mathcal{J}$ are visited at least once with the objective of minimising the total cost of the paths. A similar definition arises in case of maximisation: it is to determine a set of feasible paths from s to t in graph $\mathcal{G}$ such that the vertices of $\mathcal{J}$ are visited at most once with the objective of maximizing the total profit of the paths. Moreover, note that it is usual to consider that the number of paths selected has to be between a minimum $\underline{v}$ and a maximum $\bar{v}$. In addition, the special case where the vertices of $\mathcal{J}$ must be visited exactly once also falls in our problem class as a particular case. Remark that if the cost of the arcs satisfies the triangular inequality, an optimal solution exists that visits each vertex exactly once, even when the problem is formulated as each vertex has to be visited at least once.

Remark 2.10. *We consider a unique resource associated with the vertices whose consumption along the path is additive. The presentation in the next sections can be easily adapted to the case of multiple resources of this type. Note that resources with different*

types of consumption can be considered: for example, see the Vehicle Routing Problem with Time Windows *(VRPTW), where the resource associated with time is not additive. (Feillet, 2010).*

In the following, we provide an extended formulation for the class of problems we consider, which can be modelled by means of a set covering formulation as follows. For each path $p \in \mathcal{P}$, we introduce an integer parameter a_j^p representing the number of times path p visits vertex $j \in \mathcal{J}$. Then, we define a binary variable λ_p for each $p \in \mathcal{P}$ taking value one if path p is selected in the solution and zero otherwise.

The Set Covering formulation [SC] reads as follows:

$$[\text{SC}]\ z^* = \min \sum_{p \in \mathcal{P}} C_p \lambda_p \tag{2.16}$$

$$\text{s.t.} \sum_{p \in \mathcal{P}} a_j^p \lambda_p \geq 1 \qquad \forall j \in \mathcal{J} \tag{2.17}$$

$$\sum_{p \in \mathcal{P}} \lambda_p \geq \underline{v} \tag{2.18}$$

$$\sum_{p \in \mathcal{P}} \lambda_p \leq \bar{v} \tag{2.19}$$

$$\lambda_p \in \{0, 1\} \qquad \forall p \in \mathcal{P}. \tag{2.20}$$

Objective function (2.16) minimises the total cost of the selected paths. Constraints (2.17) are the so-called *covering constraints* and state that all the vertices in set $\mathcal{J}$ must be visited (that is *covered*) by at least one path in $\mathcal{P}$. Constraints (2.18) and (2.19) impose, respectively, a lower and an upper bound on the number of paths in the solution. Finally, Constraints (2.20) define variables λ_p as binary.

Remark 2.11 (Compact formulations)**.** *We stress that compact formulations may be considered as well to model the problems in this class. In our context, such formulations are commonly defined over arc-flow variables. Binary variable x_{ij} indicates whether arc $(i, j) \in \mathcal{A}$ is used in the solution ($x_{ij} = 1$) or not ($x_{ij} = 0$). However, these formulations usually suffer from weak linear relaxations and inner symmetries. Hence, standard branch-and-bound methods may perform poorly (see Barnhart et al., 1998, for further details). Employing extended formulations with an exponential number of variables usually yields better bounds and eliminates symmetries.*

Example 2.12 (Symmetries in the CVRP)**.** *We consider the* three-index (vehicle-flow) formulation *for the CVRP introduced by Golden et al. (1977) where the number*

of vehicles to be used is given. Arc flow variable x_{ij}^k takes value one if vehicle k traverses arc (i, j) and zero otherwise. Hence, given a solution of the formulation, it is possible to determine as many equivalent solutions as the permutations of the vehicles. Indeed, one can permute the indices of the vehicles of variables x_{ij}^k to obtain a different solution with the same value.

2.4 Solving by a Branch-Price-and-Cut algorithm

In this section, we apply the generic description of a BPC algorithm presented in Section 2.2 to solve formulation [SC]. In Section 2.4.1 we present the RMP associated with formulation [SC] and its dual problem. In Sections 2.4.2, 2.4.3 and 2.4.4, we show how the pricing problem can be formulated and solved. In Section 2.4.6, we discuss termination conditions based on the computation of a Lagrangian bound. Valid inequalities and branching rules are presented later in Sections 2.4.7 and 2.4.8, respectively. Some additional speed-up techniques are discussed in Section 2.4.9.

2.4.1 Restricted master problem

In this section, we present the RMP used when solving the linear relaxation of [SC] by means of a column generation algorithm. To do so, let $\mathcal{P}' \subseteq \mathcal{P}$ be a subset of paths.

The RMP reads as follows:

$$[\text{RMP}(\mathcal{P}')] \ \tilde{z}' = \min \sum_{p \in \mathcal{P}'} C_p \lambda_p \tag{2.21}$$

$$\text{s.t.} \sum_{p \in \mathcal{P}'} a_j^p \lambda_p \geq 1 \qquad (\rho_j) \quad \forall j \in \mathcal{J} \tag{2.22}$$

$$\sum_{p \in \mathcal{P}'} \lambda_p \geq \underline{v} \qquad (\tau_1) \tag{2.23}$$

$$\sum_{p \in \mathcal{P}'} \lambda_p \leq \bar{v} \qquad (\tau_2) \tag{2.24}$$

$$\lambda_p \geq 0 \qquad \forall p \in \mathcal{P}'. \tag{2.25}$$

Remark 2.13. *Note that we relaxed constraints $\lambda_p \leq 1$, for all $p \in \mathcal{P}$. We do so to avoid managing the dual variables associated to those constraints in the pricing problem. As it will be explained later, such task is troublesome because each dual variable is*

associated with a specific path. Removing such constraints is valid because we are solving a relaxation of [SC].

We denote by $\rho_j \geq 0$, $j \in \mathcal{J}$ the dual variables associated with Constraints (2.22) and by $\tau_1 \geq 0$ and $\tau_2 \leq 0$ the dual variables associated with Constraints (2.23) and (2.24), respectively (in blue in [RMP($\mathcal{P}'$)]). The dual problem D-RMP of the RMP reads as follows:

$$[\text{D-RMP}(\mathcal{P}')] \ \tilde{w}' = \max \sum_{j \in \mathcal{J}} \rho_j + \underline{v}\tau_1 + \bar{v}\tau_2 \tag{2.26}$$

$$\text{s.t.} \sum_{j \in \mathcal{J}} a_j^p \rho_j + \tau_1 + \tau_2 \leq C_p \qquad \forall p \in \mathcal{P}' \tag{2.27}$$

$$\rho_j \geq 0 \qquad \forall j \in \mathcal{J} \tag{2.28}$$

$$\tau_1 \geq 0 \tag{2.29}$$

$$\tau_2 \leq 0. \tag{2.30}$$

In Table 2.1, we report the notation we use for the optimal values of [SC], the [MP], the dual [D-MP] of the [MP], the [RMP($\mathcal{P}'$)] and its dual [D-RMP($\mathcal{P}'$)].

Table 2.1: Notation.

Notation	meaning
z^*	optimal value of [SC]
$\tilde{z}$	optimal value of the MP
$\tilde{w}$	optimal value of the D-MP
$\tilde{z}'$	optimal value of [RMP($\mathcal{P}'$)]
$\tilde{w}'$	optimal value of [D-RMP($\mathcal{P}'$)]

2.4.2 Pricing problem: the (E)SPPRC

In this section, we describe the structure of the pricing problem arising in the column generation procedure. Hereinafter, given a set of vertices $\mathcal{V}' \subseteq \mathcal{V}$, we denote by $\delta^-(\mathcal{V}') = \{(i,j) \in \mathcal{A} : i \notin \mathcal{V}', j \in \mathcal{V}'\}$ the set of arcs entering in vertices of $\mathcal{V}'$ and by $\delta^+(\mathcal{V}') = \{(i,j) \in \mathcal{A} : i \in \mathcal{V}', j \notin \mathcal{V}'\}$ the set of arcs exiting from vertices of $\mathcal{V}'$. To lighten the notation if $\mathcal{V}' = \{i\}$ is a singleton, we write $\delta^-(i)$ and $\delta^+(i)$.

At each iteration of the procedure, λ_p, $p \in \mathcal{P}$ variables are priced out. Specifically, the pricing problem to solve is [PP] $\bar{C}^* = \min\{\bar{C}_p : p \in \mathcal{P}\}$, where

$$\bar{C}_p = C_p - \sum_{j \in \mathcal{J}} a_j^p \rho_j - \tau_1 - \tau_2$$

is the reduced cost associated with λ_p variable. Note that $\bar{C}_p$ represents the slack of Constraints (2.27), and having a negative value of $\bar{C}_p$ means that the corresponding constraint in [D-MP] is violated. λ_p variables are associated with feasible paths in graph $\mathcal{G}$ starting in vertex s and ending in vertex t. Hence, solving the pricing problem entails searching for negative reduced cost feasible paths from source s to destination t in graph $\mathcal{G}$ where suitable costs on the arcs are defined. Precisely, for each arc $(i, j) \in \mathcal{A}$ we define its cost as follows:

$$\bar{C}_{ij} = \begin{cases} C_{ij} - \rho_j, & \text{if } i \neq s \text{ and } j \neq t \\ C_{ij} - \tau_1 - \tau_2, & \text{if } i = s \\ C_{ij}, & \text{if } j = t. \end{cases}$$

In our context, the resource consumption is associated with vertices $j \in \mathcal{J}$, that is, resource R_j is consumed every time a path visits vertex j.

Elementarity constraints, i.e., each node must be visited at most once, have a major impact on the formulation of the pricing problem and the efficiency of its solution. If such requirements are not imposed, the pricing problem can be formulated as a *Shortest Path Problem with Resource Constraints* (SPPRC).

The SPPRC can be formulated by means of an integer programming model (Beasley & Christofides, 1989). Let x_{ij} be an integer variable representing the number of times arc $(i, j) \in \mathcal{A}$ is traversed by the path. We denote by M a large positive constant. The model reads as follows.

$$[\text{PP}]\ \bar{C}^* = \min \sum_{(i,j)\in\mathcal{A}} \bar{C}_{ij} x_{ij} \tag{2.31}$$

$$\text{s.t.} \sum_{(s,j)\in\delta^+(s)} x_{sj} = 1 \tag{2.32}$$

$$\sum_{(j,t)\in\delta^-(t)} x_{jt} = 1 \tag{2.33}$$

$$\sum_{(i,j)\in\delta^-(j)} x_{ij} = \sum_{(j,i)\in\delta^+(j)} x_{ji} \qquad \forall j \in \mathcal{J} \tag{2.34}$$

$$\sum_{j\in\mathcal{J}} \sum_{(i,j)\in\delta^-(j)} R_j x_{ij} \leq \bar{R} \tag{2.35}$$

$$\sum_{(i,j)\in\mathcal{A}:i\in\mathcal{S},j\in\mathcal{S}} x_{ij} \leq M \sum_{(i,j)\in\delta^+(\mathcal{S})} x_{ij} \qquad \forall \mathcal{S} \subseteq \mathcal{V}\setminus\{t\} \tag{2.36}$$

$$x_{ij} \in \mathbb{N} \qquad \forall (i,j) \in \mathcal{A}. \tag{2.37}$$

Objective function (2.31) minimises the reduced cost of a path. Constraints (2.32) and (2.33) force a path to start in vertex s and to end in vertex t. Constraints (2.34) are the *flow balance constraints* stating that if a vertex $j \in \mathcal{J}$ is visited, the flow entering j has to be equal to the flow going out from j. Constraint (2.35) ensure the feasibility of the path with respect to the resource constraint. Constraints (2.36) forbid having a cycle disconnected from the path. More precisely, given a subset of vertices $\mathcal{S} \subseteq \mathcal{V}\setminus\{t\}$, Constraints (2.36) state that if some arcs are used between vertices of $\mathcal{S}$, then at least one arc has to exit from set $\mathcal{S}$. Note that these constraints do not forbid having vertices visited several times in the path. Finally, Constraints (2.37) define the variables involved in the model as integer.

The SPPRC is NP-hard even if graph $\mathcal{G}$ is acyclic (Di Puglia Pugliese & Guerriero, 2013). However, dynamic programming techniques based on the Bellman-Ford algorithm are particularly well-adapted to efficiently solve the problem with a pseudo-polynomial complexity (see, e.g., Desrochers, 1988).

There are many applications where paths are required to be elementary. This is typically the case of the CVRP, where a customer is visited only once in a path. To satisfy this elementarity requirement, the pricing problem can be formulated as an *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC). To formulate the ESPPRC by means of an integer programming model, we need to modify model (2.31)-

(2.37) to prevent the existence of cycles in the solution. To do so, Constraints (2.36) are replaced by the following constraints:

$$\sum_{(i,j)\in\mathcal{A}:i\in\mathcal{S},j\in\mathcal{S}} x_{ij} \leq |\mathcal{S}| - 1 \qquad \forall \mathcal{S} \subseteq \mathcal{V}. \tag{2.38}$$

Such constraints are referred to as *subtour elimination constraints*. Remark that other subtour elimination constraints may be used, such as the ones introduced in Dantzig *et al.* (1954) or in Miller *et al.* (1960) to obtain a compact formulation. Note that to model the ESPPRC, variables x_{ij} will never take a value greater than one, so constraints (2.37) can be updated to define the variables as binary.

In the following sections, we discuss solution algorithms for the SPPRC and the ESPPRC. Note that the ESPPRC is strongly NP-hard (Dror, 1994) and requires more sophisticated techniques to be solved efficiently than the SPPRC. For these reasons, the focus of the next sections is more oriented to the solution of the ESPPRC. Observe that the topology of the graph may simplify the ESPPRC, e.g. if $\mathcal{G}$ is acyclic, then the ESPPRC reduces to a SPPRC.

2.4.3 Algorithms for the SPPRC

The SPPRC can be modelled via the following Bellman's recursive formula:

$$\begin{cases} V(0,s) = 0 \\ V(R,j) = \min_{i\in\mathcal{V}:(i,j)\in\delta^-(j)}\{V(R-R_j,i) + \bar{C}_{ij}\}, \quad \forall j \in \mathcal{V}, \forall R_j \leq R \leq \bar{R}, \end{cases} \tag{2.39}$$

where we denoted by $V(R,j)$ the value of an optimal feasible path starting in vertex s, ending in vertex $j \in \mathcal{V}$ and with a resource consumption equal to R. The spatial and temporal complexity of dynamic program (2.39) are $O(\bar{R}|\mathcal{V}|)$ and $O(\bar{R}|\mathcal{A}|)$, respectively.

As mentioned in the previous section, although the SPPRC is NP-hard, variants of the Bellman-Ford algorithm can efficiently solve the SPPRC in pseudo-polynomial time (see, e.g., Jaumard *et al.*, 1999). We present one of such variants in Algorithm 5.

Algorithm 5: A Bellman-Ford algorithm for the SPPRC

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$.
Initialization: $V(0, s) \coloneqq 0$ and $p_s^0 \coloneqq s$, $V(R, j) \coloneqq \infty$ and $p_j^R = null$ for all $j \in \mathcal{V} \setminus \{t\}$ and $R = 0, \dots, \bar{R}$.

```
forall R = 0, ..., R̄ do
    forall j ∈ V \ {s} do
        if R ≥ R_j then
            forall (i, j) ∈ δ⁻(j) do
                if V(R − R_j, i) + C̄_ij < V(R, j) then
                    set V(R, j) := V(R − R_j, i) + C̄_ij;
                    set p_j^R = i;
                end
            end
        end
    end
end
```

2.4.4 Algorithms for the ESPPRC

The ESPPRC can be formulated with a Bellman's recursive equation:

$$\begin{cases} V(\{s\}, s) = 0 \\ V(\mathcal{S}, j) = \min_{i \in \mathcal{S}: (i,j) \in \delta^-(j)} \{V(\mathcal{S} \setminus \{j\}, i) + \bar{C}_{ij}\}, \\ \qquad\qquad \forall j \in \mathcal{V}, \mathcal{S} \subseteq \mathcal{V} : j \in \mathcal{S}, \sum_{i \in S} R_i \leq \bar{R}, \end{cases} \tag{2.40}$$

where we denoted by $V(\mathcal{S}, j)$ the value of an optimal feasible path starting in vertex s, ending in vertex $j \in \mathcal{S} \subseteq \mathcal{V}$ and visiting all the vertices in $\mathcal{S}$.

Note that the recursive formula does not take into account the resource. Indeed, we consider a resource that has an additive consumption associated with the vertices. Hence, given a set of vertices $\mathcal{S}$, the associated resource consumption is known ($\sum_{i \in \mathcal{S}} R_i$), so only the sets such that the resource consumption does not exceed $\bar{R}$ are considered. The spatial complexity of such a dynamic program is $O(|\mathcal{V}|2^{|\mathcal{V}|})$, and the temporal complexity is $O(|\mathcal{A}|2^{|\mathcal{V}|})$.

Dynamic programming algorithms are commonly employed and well-suited to solve the ESPPRC in the context of column generation. First, they allow to provide not only one negative reduced cost column but several of them. This is known to speed up the convergence of the column generation procedure. Second, it is easy to stop the algorithm prematurely when some negative reduced cost paths have been identified. In this section, we discuss two exact label correcting algorithms to solve the dynamic program (2.40) for the ESPPRC. Such algorithms stem from the Bellman-Ford algorithm. The idea behind them is to associate feasible partial paths starting from s with labels which encode the attributes needed to identify such paths. Labels are then repeatedly

extended from vertex to vertex until there is no more possible extension. All the labels extended to the destination t correspond to feasible paths, the ones with negative reduced cost can be reported as new columns to the RMP, while the ones with the most negative reduced cost are optimal solutions of the ESPPRC.

Note that the interest in using such labelling algorithms to solve the ESPPRC instead of a classical Bellman-Ford algorithm relies on the possibility of encoding the paths in well-suited data structures, the *labels*. Indeed, when solving the SPPRC with a Bellman-Ford algorithm as presented in Algorithm 5, a matrix of size equal to the number of vertices in the graph times the maximum consumption of resources $\bar{R}$ is enough to encode all the states of the dynamic programming. When we are interested in retrieving the optimal path, a matrix of the same size is also needed: for each vertex, the entries of such matrix store the vertex previously visited in the path. Conversely, when solving the ESPPRC, to ensure the elementarity constraints, the states of the dynamic programming have to contain the set of vertices $\mathcal{S}$ visited in the partial path. Doing so in a Bellman-Ford algorithm entails using a matrix whose rows are the vertices of the graph and whose columns are the subsets of vertices of the graph, i.e., a matrix of exponential size. Such a matrix has to be built a priori and traversed at each iteration of the algorithm. In a labelling algorithm, we do not need to build any data structure a priori, paths are encoded in labels generated on the fly.

For the sake of completeness, we should point out that there exist cases where a dynamic programming approach is not well-suited: e.g., in the presence of additional constraints that force a group of vertices to be visited together in the same path. To deal with this case, some recent works use integer programming techniques to solve an ESPPRC (see, e.g., Briant *et al.*, 2020; Hintsch & Irnich, 2020).

In practice, a dynamic programming approach performs well when the total amount of resource $\bar{R}$ is small w.r.t. the resource consumption of the vertices. Indeed, in such cases, only a few partial paths (subsets of $\mathcal{V}$) are feasible, and the number of feasible paths can be rather small in comparison with the total number of subsets of $\mathcal{V}$.

In what follows, we present the main ingredients of the labelling algorithm. Precisely, we formally introduce the labels, the rule to extend them and the dominance rules to prune unpromising labels.

Let p be a partial path starting from source s and ending in vertex $j \in \mathcal{J}$. Path p is represented by label $l = (j, R, \bar{C}, \mathcal{U})$ where j: is the last visited vertex in p; R: is the accumulated resource consumption, i.e., $R = \sum_{i \in \mathcal{J}(p)} R_i$; $\bar{C}$: is the accumulated reduced

cost, i.e., $\bar{C} = \sum_{(h,i)\in\mathcal{A}(p)} \bar{C}_{hi}$; $\mathcal{U}$: is the set of vertices visited by the path, i.e., $\mathcal{U} = \mathcal{J}(p)$. Note that set $\mathcal{U}$ is the label attribute that distinguishes the solution of an ESPPRC from the one of a SPPRC (see Beasley & Christofides, 1989). As mentioned in Feillet *et al.* (2004), the label definition presented above can be improved by generalising the definition of set $\mathcal{U}$ to include also the vertices that cannot be visited in any extension of p due to some other resource consumption, i.e., $\mathcal{U}_I = \mathcal{U} \cup \{i \in \mathcal{J} : R + R_i > \bar{R}\}$. The vertices in $\mathcal{U}_I$ are called *unreachable*, while the ones in $\mathcal{J} \setminus \mathcal{U}_I$ are called *reachable*. In the following, we adopt this extended definition of the labels. A label is said to be *feasible* if the associated partial path is feasible, i.e., if $R \leq \bar{R}$.

Now, we present the *extension rule* to extend a label $l = (j, R, \bar{C}, \mathcal{U}_I)$ encoding a path ending at vertex j along an arc $(j, j') \in \mathcal{A}$. In terms of paths, this operation corresponds to appending vertex j' to the partial path associated with l.

The extended label $l' = (j', R', \bar{C}', \mathcal{U}'_I)$ is defined as follows:

$$\begin{cases} R' = R + R_j \\ \bar{C}' = \bar{C} + \bar{C}_{jj'} \\ \mathcal{U}'_I = \{i \in \mathcal{J} : i \in \mathcal{U}_I \cup \{j'\} \vee R' + R_i > \bar{R}\}. \end{cases} \tag{2.41}$$

Extending all the labels until no further extension can be performed (due to the resource constraint) ensures the correctness of the algorithm. However, extending all the labels may be intractable: in the worst-case scenario, there are $2^{|\mathcal{V}|}$ subsets to consider, and for each subset $\mathcal{S} \subseteq \mathcal{V}$, there are $|\mathcal{S}|!$ possible labels. To limit the explosion of the number of labels, *dominance rules* are applied to prune unpromising labels, i.e., partial paths that would not lead to an optimal solution. We stress that dominance rules are a speedup technique, i.e., removing them has no impact on the correctness of the algorithm. However, it highly affects its effectiveness.

Let us present the dominance rules that can be applied in the labelling algorithm. Let $l = (j, R, \bar{C}, \mathcal{U}_I)$ and $l' = (j, R', \bar{C}', \mathcal{U}'_I)$ be two labels encoding two paths ending at vertex j. We say that l *dominates* l' if

$$\begin{cases} R \leq R' \\ \bar{C} \leq \bar{C}' \\ \mathcal{U}_I \subseteq \mathcal{U}'_I \end{cases} \tag{2.42}$$

and at least one of these inequalities is strict. Here, the existence of label l allows to prune label l' without affecting the correctness of the algorithm. Indeed, let $\tilde{p}$ be a

partial path from j to t such that the concatenation of the path encoded in l' with $\tilde{p}$ leads to a complete feasible path. Observe that all the vertices that are reachable in an extension of l' are also reachable in an extension of l, in addition, l consumes less resource and provides a better reduced cost w.r.t. the ones of l'. Hence, the concatenation of the partial path encoded in l with $\tilde{p}$ leads to a complete feasible path, as well, with a cost which is at least as good as the one provided by the concatenation of the partial path associated with l' and $\tilde{p}$.

Note that verifying the third condition in (2.42) is computationally expensive. One should verify the inclusion relation between two sets whose potential size may be equal to $|\mathcal{J}|$. Hence, although $\mathcal{U}_I$ is defined as a set, at the implementation level, it is usually defined as a *bitset*. Such data structures are well-suited to perform set operations efficiently: the average time complexity to perform tasks such as intersections, unions or verifying inclusion relations is constant.

We now present descriptions of two classical algorithms for the ESPPRC due to Feillet *et al.* (2004) and to Righini & Salani (2006) with some details about their possible implementations.

2.4.4.1 Monodirectional algorithm of Feillet *et al.* (2004)

In the following, we say that a label is *associated with a vertex* j if its partial path ends in vertex j. Further, we denote by $\mathcal{L}_j$, $j \in \mathcal{V}$ the set of labels associated with vertex j and by $\bar{\mathcal{L}}_j \subseteq \mathcal{L}_j$ the subset of the labels of j not extended yet. We report the pseudocode of the monodirectional algorithm in Algorithm 6.

Algorithm 6: Monodirectional algorithm of Feillet *et al.* (2004)

Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$.
Initialization: $\mathcal{L}_j := \emptyset$, for all $j \in \mathcal{V}$; $\mathcal{L}_s = \bar{\mathcal{L}}_s := \{(s, 0, 0, \emptyset)\}$; $termination := false$.
while $termination \neq true$ **do**
  set $termination := true$;
  forall $j \in \mathcal{V} \setminus \{t\}$ **do**
    forall $l = (j, R, \bar{C}, \mathcal{U}_I) \in \bar{\mathcal{L}}_j$ **do**
      forall $(j, j') \in \delta_j^+$ *s.t.* $j' \notin \mathcal{U}_I$ **do**
        let $l' = (j', R', \bar{C}', \mathcal{U}_I')$ be the extension of l to j';
        if `DOM`$(\mathcal{L}_{j'}, l')$ **then**
          set $termination := false$;
        end
      end
      remove l from $\bar{\mathcal{L}}_j$;
    end
  end
end

The algorithm takes in input graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. In the initialisation step, the sets of labels of all the vertices are set to be empty, except for the one of the source vertex s, where a label with zero resource consumption and zero cost is included. In addition, Boolean variable *termination* to detect whether the execution of the algorithm is finished is set to $false$.

At each iteration of the algorithm, the variable *termination* is set to *true*. Then, the algorithm loops over the vertices of graph $\mathcal{G}$, the labels not yet extended, and the arcs going out from the vertices to detect some new non-dominated labels. If it succeeds, *termination* is set to $false$, and the iteration repeats. Now, we discuss how new non-dominated labels may be detected. Let $j \in \mathcal{V} \setminus \{t\}$ be a vertex, $l = (j, R, \bar{C}, \mathcal{U}_I) \in \bar{\mathcal{L}}_j$ a label of j not extended yet and $(j, j') \in \delta_j^+$ be an arc going out of j such that vertex j' is reachable by the partial path represented by l and its inclusion in the path represented by l respect the resource consumption constraints. The extended label $l' = (j', R', \bar{C}', \mathcal{U}'_I)$ is produced as specified by rules (2.41). Then, function $\texttt{DOM}(\mathcal{L}_{j'}, l')$ tests if label l' is not dominated by other labels in $\mathcal{L}_{j'}$ by implementing rules (2.42). If this is the case, it inserts l' in $\mathcal{L}_{j'}$ and removes from $\mathcal{L}_{j'}$ all the labels dominated by l'. Once all the possible extensions of label l are processed, we remove it from set $\bar{\mathcal{L}}_j$.

Function $\texttt{DOM}(\mathcal{L}_{j'}, l')$ is usually the bottleneck of the algorithm. On the one hand, it helps in limiting the explosion of the number of labels, on the other hand, it is called every time a new label is generated and the tests it performs are time consuming. In the following, we provide details on how these two objectives can be balanced. Once again, we would like to stress that the correctness of the algorithm is valid if the function DOM is removed from it. Remark that to perform the dominance tests in DOM, we need to loop over the entire set of labels $\mathcal{L}_{j'}$. Hence, a good practice is to sort the labels stored in sets $\mathcal{L}_j$, $j \in \mathcal{V}$ to avoid useless tests. Precisely, labels are sorted by increasing reduced cost, and if the reduced cost is the same, they are ordered by increasing resource consumption. Hence, in function DOM, the loop over set $\mathcal{L}_{j'}$ can be split in three steps:

1. consider the labels $l \in \mathcal{L}_{j'}$ such that $\bar{C} < \bar{C}'$ and test whether l dominates l';
2. consider the labels $l \in \mathcal{L}_{j'}$ such that $\bar{C} = \bar{C}'$ and test whether l dominates l' and the vice-versa;
3. consider the labels $l \in \mathcal{L}_{j'}$ such that $\bar{C} > \bar{C}'$ and test whether l' dominates l.

Clearly, the loop stops if label l' is dominated in one of the first two steps. Otherwise, label l' has to be inserted right after the second step is performed, and, in the third step, labels in set $\mathcal{L}_{j'}$ which are dominated by l' are erased from the set.

Even this improved management of function `DOM` may not be enough in some applications. Indeed, the dominance function may be called too many times. Hence, one idea to limit the calls to the dominance rule is to organise the labels in sets $\mathcal{L}_j$ in subsets called *buckets*. Each bucket contains labels characterised by similar values of resource consumption. One can discretise interval $[0, \bar{R}]$ corresponding to the possible resource consumption values. Let $\{r_0 = 0, r_1, \ldots, r_m = \bar{R}\}$ be such discretisation. Sets $\mathcal{L}_j$ can be partitioned in m buckets, each corresponding to an interval $[r_l, r_{l+1})$, $l = 0, \ldots, m-1$ (the last interval contains also r_m) of resource consumption. When a new label is generated, we identify the bucket where it should be stored, and we perform the dominance tests only considering the labels in that bucket and, eventually, the ones stored in "adjacent" buckets. Given that dominance rules do not affect the correctness of the algorithm, one can perform the dominance tests partially instead of the entire set of labels. Sadykov *et al.* (2021) extend the idea of the buckets by introducing the concept of *bucket graph*, we refer to their work for further details.

2.4.4.2 Bounded bidirectional algorithm of Righini & Salani (2006)

As already mentioned, labelling algorithms, even with dominance rules, may suffer from the explosion of the number of labels. In this case, the performances of the algorithm are heavily affected. To limit this phenomenon, Righini & Salani (2006) proposed to improve the algorithm of Feillet *et al.* (2004) with two techniques: bidirectional search and bounding. The idea of the bidirectional search is to extend labels *forward* from s to t and *backward* from t to s. By doing so, an algorithm would produce double the labels of the monodirectional search. Hence, Righini & Salani (2006) couples the bidirectional search with the bounding: only labels that consume less than a certain fraction of the available maximal resource quantity are extended forward and backward. This limit value is called *halfway point*. Righini & Salani (2006) set this value to be half of the available maximal resource quantity, i.e., to $\bar{R}/2$. Note that Tilk *et al.* (2017) proposed a more efficient and dynamic way to define the halfway point that is not considered here. Once the forward and backward extensions are performed, forward and backward labels are merged to obtain labels corresponding to complete feasible paths.

In the following, we report some details regarding the bounded bidirectional algorithm. We denote by $\mathcal{L}_j^F$, $j \in \mathcal{V}$ the set of forward labels associated with vertex j and by $\bar{\mathcal{L}}_j^F \subseteq \mathcal{L}_j^F$ the subset of the forward labels of j not yet extended. Analogously, we define sets $\mathcal{L}_j^B$ and $\bar{\mathcal{L}}_j^B$, $j \in \mathcal{V}$ for the backward labels. We do not report the pseudocode of the bounded bidirectional algorithm, but only the procedure to merge forward and backward labels (see Algorithm 7). Indeed the forward and backward extensions are obtained by applying a few modifications to the monodirectional algorithm (see Algorithm 6). Precisely, to produce a set of forward labels, one can apply the procedure of Algorithm 6 where sets $\mathcal{L}_j$ and $\bar{\mathcal{L}}_j$ are replaced with sets $\mathcal{L}_j^F$ and $\bar{\mathcal{L}}_j^F$, respectively, and a label $l \in \bar{\mathcal{L}}_j^F$ is extended only if its consumption is strictly less than $\bar{R}/2$. Similarly, to obtain a set of backward labels, one applies the same procedure where the roles of source s and sink t are inverted and loop over the out-going arcs becomes a loop over the in-going arcs.

Hence, we discuss only function `Merge` whose pseudocode is in Algorithm 7. Let $l = (j, R, \bar{C}, \mathcal{U}_I)$ be a forward label of vertex $j \in \mathcal{J}$ and $l' = (j', R', \bar{C}', \mathcal{U}'_I)$ be a backward label of vertex $j' \in \mathcal{J}$. These two labels can be merged if the total amount of resource consumed is less than the maximal available resource quantity ($R + R' \leq \bar{R}$) and if the vertices in $\mathcal{U} \subseteq \mathcal{U}_I$ visited along the path encoded by l are reachable by the path encoded by l', i.e., $\mathcal{U} \cap \mathcal{U}'_I = \emptyset$, and vice-versa $\mathcal{U}' \cap \mathcal{U}_I = \emptyset$. If this is the case, the merged label l^{merge} is built. Such a label corresponds to the path obtained by merging path $(s, \ldots, j)$ associated with label l with the path $(j', \ldots, t)$ associated with label l'. The resulting merged label is $l^{merge} = (t, R + R', \bar{C} + \bar{C}' + \bar{C}_{jj'}, \mathcal{U}_I \cup \mathcal{U}'_I)$.

Algorithm 7: `Merge` function of algorithm by Righini & Salani (2006)

```
   Input: L_j^F, L_j^B, for all j ∈ V.
1  forall j ∈ J do
2      forall l = (j, R, C̄, U_I) ∈ L_j^F do
3          forall j' ∈ J s.t. (j, j') ∈ A do
4              forall l' = (j', R', C̄', U'_I) ∈ L_j'^B do
5                  if R + R' ≤ R̄ ∧ U ∩ U'_I = U' ∩ U_I = ∅ then // Merge feasibility test
6                      let l^merge be the label resulting from merging l with l';
7                  end
8              end
9          end
10     end
11 end
```

We conclude the section with a remark about a special case where the computational burden to solve the ESPPRC can be further reduced. This special case leads to a variant of the algorithm of Righini & Salani (2006) and is referred to as implicit version of the

bidirectional algorithm and was applied by Bode & Irnich (2012) and Goeke *et al.* (2019). Precisely, we suppose that $\mathcal{G}$ is a symmetric graph (i.e., if $(i,j) \in \mathcal{A}$, then $(j,i) \in \mathcal{A}$) and the costs associated with its arcs are symmetric as well ($\bar{C}_{ij} = \bar{C}_{ji}$). Under these assumptions, forward and backward labels are essentially identical. Hence, it is sufficient to perform only the forward extension and then to merge two forward labels. In such a case, it is also sufficient only to merge labels corresponding to partial paths ending at the same node. Hence some minor modifications have to be considered as described in the following.

First, labels with a resource consumption less than *or equal* to $\bar{R}/2$ are extended in the forward extensions. The equality case has to be considered to avoid missing a path where each half consumes exactly $\bar{R}/2$ units of resource. Then, the function `Merge` has to be modified as in Algorithm 8. It is sufficient to merge labels associated with the same vertex and one of the two must exceed the value of the halfway point. Indeed, all labels corresponding to feasible paths from s to t with a resource consumption less or equal $\bar{R}/2$ are generated during the forward extension. Hence, the merge function needs to look only for labels with a resource consumption greater than $\bar{R}/2$. In addition, a feasible path from s to t can result from merging different partial paths, depending on where the path is split. Hence, filtering partial paths with resource consumption greater than $\bar{R}/2$ does not eliminate any feasible path and permits avoiding computing too many times the same path.

Moreover, since two partial paths ending at the same vertex are merged, the merging condition must be updated as shown in Line 5 of Algorithm 8. The resulting merged label l^{merge} from labels $l = (j, R, \bar{C}, \mathcal{U}_I)$ and $l' = (j, R', \bar{C}', \mathcal{U}'_I)$ is $l^{merge} = (t, R + R' - R_j, \bar{C} + \bar{C}', \mathcal{U}_I \cup \mathcal{U}'_I)$.

Algorithm 8: `Merge` function of implicit version of the bidirectional algorithm

```
   Input: L_j^F, for all j ∈ V.
1  forall j ∈ J do
2      forall l = (j, R, C̄, U_I) ∈ L_j^F do
3          if R > R̄/2 then
4              forall l' = (j, R', C̄', U'_I) ∈ L_j^F do
5                  if R + R' − R_j ≤ R̄ ∧ U ∩ U'_I = U' ∩ U_I = {j} then // Merge feasibility test
6                      let l^merge be the label resulting from merging l with l';
7                  end
8              end
9          end
10     end
11 end
```

2.4.5 Relaxing the elementarity constraint

Requiring the elementarity of the paths makes the pricing problem strongly NP-hard and less tractable from a computational point of view. Hence, a rather vast branch of research was devoted to study efficient relaxations of the elementarity constraint. Note that instead of solving the MP where λ_p variables correspond to elementary feasible paths, one might solve a relaxation of the MP where λ_p variables also correspond to some non-elementary paths. In such a case, coefficients a_j^p in Constraints (2.22) indicate the number of times vertex j is visited in path p. Solving such a relaxation by column generation provides a lower bound for the MP, that is also a lower bound of [SC]. The main interest is to solve a relaxation of the ESPPRC when solving the pricing problem. Solving, for example, a (non-elementary) SPPRC can be done using a pseudo-polynomial algorithm (see Section 2.4.3). Conversely, solving a relaxation might lead to poor quality lower bounds. Hence, the objective of these studies on efficient relaxations of the elementarity constraint was to find a balance between making the solution of the ESPPRC more tractable while not weakening too much the value of the linear relaxation of the RMPs.

It can also be noticed that efficiently solving relaxations of the ESPPRC can be useful in the pricing step, even when solving the MP with elementary paths. Indeed, if the optimal value of the relaxation is positive or zero, it proves there are no more negative reduced cost paths. Also, if solving the relaxation provides some negative reduced cost elementary paths, they can be included in the RMP without solving the ESPPRC. However, if solving the relaxation only provides negative reduced cost non-elementary paths, then the ESPPRC has to be solved. So such approaches make sense only if the relaxation can be solved efficiently and easily provides elementary negative reduced cost paths.

In this section, we briefly report the main relaxations proposed through the years. Then, we focus the discussion on the so-called *ng-path* relaxation introduced in Baldacci *et al.* (2011), nowadays implemented in almost all state-of-the-art BPC algorithms.

The first idea in this direction appeared in Desrochers *et al.* (1992): the authors proposed to relax the elementarity constraint while forbidding 2-cycles, i.e., paths with cycles of size two. Irnich & Villeneuve (2006) extended the idea of forbidding cycles in the paths to the generic case of the k-cycles. Computational experiments showed that a good trade-off between improving the value of the linear relaxation and keeping the solution of the pricing problem tractable was attained by forbidding cycles up to

size four. Boland *et al.* (2006) and Righini & Salani (2008) simultaneously developed a procedure based on state space relaxation. Precisely, the procedure starts by solving the SPPRC and dynamically includes elementarity constraints for some of the vertices. Finally, Baldacci *et al.* (2011) proposed the *ng-path relaxation*. Vertices are assigned with a neighbourhood, and cycles are allowed only if the vertex visited more than once in a path is not in the neighbourhood of its predecessor in that path. We notice that such a neighbourhood can be dynamically adjusted during the execution of the algorithm (Roberti & Mingozzi, 2014). However, we discuss the static case where the neighbourhoods are pre-assigned to the vertices. In the following, we provide a detailed description of the ng-path relaxation.

A neighbourhood $\mathcal{N}_j \subseteq \mathcal{J}$ of size $\Delta > 0$ is assigned to each vertex $j \in \mathcal{J}$. Such neighbourhood must contain vertex j and, e.g., the closest $\Delta - 1$ vertices to j. Set $\mathcal{N}_j$, $j \in \mathcal{J}$ has to be seen as a *memory* of vertex j. Hence, if a path visits vertex j, we keep in memory only the previously visited vertices that are in the neighbourhood of j. When extending the path, the extension to a vertex in the memory is not allowed.

In order to use the ng-path relaxation, the following modifications to the labelling algorithm for the pricing problem have to be implemented. In the definition of the labels, the vector of unreachable vertices $\mathcal{U}_I$ becomes the *memory of the label*. The rule to extend $\mathcal{U}_I$ of label $l = (j, R, \bar{C}, \mathcal{U}_I)$ from j to j' to obtain a new label $l' = (j', R', \bar{C}', \mathcal{U}'_I)$ becomes:

$$\mathcal{U}'_I = (\mathcal{U}_I \cap \mathcal{N}_{j'}) \cup \{j'\}. \tag{2.43}$$

Remark that dominance rules do not need any modifications. We clarify the explanation of the ng-paths with the following example.

Example 2.14 (Ng-path)**.** *We consider a complete graph* $\mathcal{G} = (\mathcal{V} = \mathcal{J} \cup \{s, t\}, \mathcal{A})$, *where* $\mathcal{J} = \{1, 2, 3, 4, 5\}$. *Table 2.2 reports the memory of the vertices in* $\mathcal{J}$ *with* $\Delta = 3$.

Table 2.2: Memory of the vertices in $\mathcal{G}$.

$j \in \mathcal{J}$	$\mathcal{N}_j$
1	$\{1, 2, 3\}$
2	$\{1, 2, 5\}$
3	$\{2, 3, 4\}$
4	$\{1, 2, 4\}$
5	$\{2, 3, 5\}$

Let us consider an ng-path $p = (s, 1, 2, 3, 1, t)$ *where vertex 1 is visited twice. In Table 2.3, we report how the memory of the labels required to build path* p *is extended.*

The first column of Table 2.3 reports the partial path associated with the label and the second column reports the memory of the label.

Table 2.3: Ng-path label extension to obtain path p.

partial path	$\mathcal{U}_I$
(s)	$\emptyset$
$(s,1)$	$\{1\} = (\emptyset \cap \mathcal{N}_1) \cup \{1\}$
$(s,1,2)$	$\{1,2\} = (\{1\} \cap \mathcal{N}_2) \cup \{2\}$
$(s,1,2,3)$	$\{2,3\} = (\{1,2\} \cap \mathcal{N}_3) \cup \{3\}$
$(s,1,2,3,1)$	$\{1,2,3\} = (\{2,3\} \cap \mathcal{N}_1) \cup \{1\}$
$(s,1,2,3,1,t)$	$\emptyset$

Vertex 1 *is allowed to be visited two times: when the path is extended to vertex* 3*, the first visit to* 1 *is forgotten since the memory of the partial path* $(s,1,2,3)$ *is* $\{2,3\}$.

Remark that if neighbourhood $\mathcal{N}_j$ only contains vertex j, we are solving the SPPRC. Conversely, if they contain all the vertices, we are solving the ESPPRC. In practice, small size neighbourhoods, e.g., $\Delta \leq 8$ (Pessoa *et al.*, 2020), yield very good results: many paths generated by the pricing are elementary while the solving time decreases significantly.

2.4.6 Dual bound and termination condition

In the context of a BPC algorithm, column generation is used to solve the MP at each node of the branch-and-bound tree. If one is interested in solving the MP to optimality at each node, an obvious termination condition is given by the solution of the pricing problem. Indeed, the pricing problem certifies the optimality of the current solution of the RMP when no negative reduced cost columns are identified. In a BPC algorithm, column generation may be terminated earlier than the point at which the pricing problem condition is met without giving up its correctness. In this section, we discuss such a termination condition, which exploits the link between column generation and *Lagrangian relaxation*. Precisely, Lagrangian relaxation allows us to compute a lower bound on the value of the MP at each iteration of the column generation procedure. The lower bound can be used to end the column generation procedure before the termination condition given by solving the pricing problem is met.

Let us consider the MP, i.e., the linear relaxation of formulation [SC], see Section 2.3. We relax in a Lagrangian fashion covering Constraints (2.17). Let $L : \mathbb{R}^{|\mathcal{J}|}_{\geq 0} \to \mathbb{R}$ be the *dual function* defined as:

$$L(\rho) = \min \left\{ \sum_{p \in \mathcal{P}} C_p \lambda_p + \sum_{j \in \mathcal{J}} \rho_j \left(1 - \sum_{p \in \mathcal{P}} a_j^p \lambda_p \right) : (2.18), (2.19), \lambda_p \geq 0, \forall p \in \mathcal{P} \right\}.$$

The problem defined by $L(\rho)$, $\rho \in \mathbb{R}^{|\mathcal{J}|}_{\geq 0}$, is a relaxation of the MP. Hence, for each $\rho \in \mathbb{R}^{|\mathcal{J}|}_{+}$, value $L(\rho)$ is a lower bound for the optimal value $\tilde{z}$ of the MP. The associated *Lagrangian dual problem* reads as follows:

$$[\text{LD}] \quad \max\{L(\rho) : \rho \in \mathbb{R}^{|\mathcal{J}|}_{\geq 0}\}$$

and provides the best among these lower bounds.

By duality, problem [LD] can be reformulated as a linear program which corresponds to the MP.

By definition, we have

$$\begin{aligned}
[\text{LD}] \quad & \max\{L(\rho) : \rho \in \mathbb{R}^{|\mathcal{J}|}_{\geq 0}\} \\
&= \max \left\{ \min \left\{ \sum_{p \in \mathcal{P}} C_p \lambda_p + \sum_{j \in \mathcal{J}} \rho_j \left(1 - \sum_{p \in \mathcal{P}} a_j^p \lambda_p \right) : (2.18), (2.19), \lambda_p \geq 0, \forall p \in \mathcal{P} \right\} : \rho \in \mathbb{R}^{|\mathcal{J}|}_{\geq 0} \right\} \\
&= \max \left\{ \sum_{j \in \mathcal{J}} \rho_j + \min \left\{ \sum_{p \in \mathcal{P}} \left(C_p - \sum_{j \in \mathcal{J}} a_j^p \rho_j \right) \lambda_p : (2.18), (2.19), \lambda_p \geq 0, \forall p \in \mathcal{P} \right\} : \rho \in \mathbb{R}^{|\mathcal{J}|}_{\geq 0} \right\}.
\end{aligned}$$

Let us consider the inner minimisation problem defined over the λ variables. We denote by τ_1 and τ_2 the dual variables associated to Constraints (2.18) and (2.19), respectively. The dual of this LP can be written as:

$$\max \left\{ \underline{v} \tau_1 + \bar{v} \tau_2 : \tau_1 + \tau_2 \leq C_p - \sum_{j \in \mathcal{J}} a_j^p \rho_j, \forall p \in \mathcal{P}, \tau_1 \geq 0, \tau_2 \leq 0 \right\}.$$

Hence, problem [LD] can be formulated as the following maximisation problem:

$$\begin{aligned}[\text{LD}]\ \max &\left\{\sum_{j\in\mathcal{J}}\rho_j + \underline{v}\tau_1 + \bar{v}\tau_2 : \tau_1+\tau_2 \le C_p - \sum_{j\in\mathcal{J}} a_j^p\rho_j, \forall p\in\mathcal{P}, \rho\in\mathbb{R}_{\ge 0}^{|\mathcal{J}|}, \tau_1\ge 0, \tau_2\le 0\right\}\\ =\max &\left\{\sum_{j\in\mathcal{J}}\rho_j + \underline{v}\tau_1 + \bar{v}\tau_2 : \sum_{j\in\mathcal{J}} a_j^p\rho_j+\tau_1+\tau_2 \le C_p, \forall p\in\mathcal{P}, \rho\in\mathbb{R}_{\ge 0}^{|\mathcal{J}|}, \tau_1\ge 0, \tau_2\le 0\right\}.\end{aligned}$$

By denoting with λ the dual associated to each constraint $\sum_{j\in\mathcal{J}} a_j^p\rho_j+\tau_1+\tau_2 \le C_p$, the dual of [LD] can be formulated as:

$$\min\left\{\sum_{p\in\mathcal{P}} C_p\lambda_p : (2.17), (2.18), (2.19), \lambda_p\ge 0, \forall p\in\mathcal{P}\right\}$$

This last problem is exactly the MP.

Now, we denote by $(\bar{\rho},\bar{\tau}_1,\bar{\tau}_2)$ an optimal solution of the dual of the MP. $\bar{\rho}$ is an optimal solution also for [LD]. Indeed, it is a feasible solution for [LD] since the values are positive. Then using the strong duality theorem, we have that: (i) the optimal value of the dual of the MP is equal to the optimal value of the MP, and (ii) the optimal value of [LD] is equal to the optimal value of its dual. Since the dual of [LD] is the MP, we have that the optimal value of the dual of the MP is equal to the optimal value of [LD].

Let us define $(\bar{\rho},\bar{\tau}_1,\bar{\tau}_2)$ an optimal solution of the D-RMP. In the following, we show how it is possible to compute, at each iteration of the column generation, $L(\bar{\rho})$ that is a lower bound for the optimal value of the MP.

Value $L(\bar{\rho})$ can be re-written in terms of optimal values of the RMP and of the pricing problem:

$$\begin{aligned}L(\bar{\rho}) =&\min\left\{\sum_{p\in\mathcal{P}} C_p\lambda_p + \sum_{j\in\mathcal{J}}\bar{\rho}_j\left(1-\sum_{p\in\mathcal{P}} a_j^p\lambda_p\right) : (2.18), (2.19), \lambda_p\ge 0, \forall p\in\mathcal{P}\right\}\\ =&\sum_{j\in\mathcal{J}}\bar{\rho}_j + \min\left\{\sum_{p\in\mathcal{P}}\left(C_p - \sum_{j\in\mathcal{J}} a_j^p\bar{\rho}_j\right)\lambda_p : (2.18), (2.19), \lambda_p\ge 0, \forall p\in\mathcal{P}\right\}.\end{aligned}$$

Remark that given $(\bar{\rho},\bar{\tau}_1,\bar{\tau}_2)$ an optimal solution of the D-RMP, the reduced cost of a variable λ_p is $\bar{C}_p = C_p - \sum_{j\in\mathcal{J}} a_j^p\bar{\rho}_j - \bar{\tau}_1 - \bar{\tau}_2$. Hence, we have $C_p - \sum_{j\in\mathcal{J}} a_j^p\bar{\rho}_j = \bar{C}_p + \bar{\tau}_1 + \bar{\tau}_2$. In addition, $\sum_{j\in\mathcal{J}}\bar{\rho}_j + \underline{v}\bar{\tau}_1 + \bar{v}\bar{\tau}_2$ corresponds to the optimal value of the dual of the

RMP, that is equal to $\tilde{z}'$ the optimal value of the RMP by strong duality. Hence, $\sum_{j\in\mathcal{J}} \bar{\rho}_j = \tilde{z}' - \underline{v}\bar{\tau}_1 - \bar{v}\bar{\tau}_2$. $L(\bar{\rho})$ can then be written as:

$$L(\bar{\rho}) = \tilde{z}' - \underline{v}\bar{\tau}_1 - \bar{v}\bar{\tau}_2 + \min\left\{\sum_{p\in\mathcal{P}} \left(\bar{C}_p + \bar{\tau}_1 + \bar{\tau}_2\right)\lambda_p : (2.18), (2.19), \lambda_p \geq 0, \forall p \in \mathcal{P}\right\}.$$

Note that problem

$$\min\left\{\sum_{p\in\mathcal{P}} \left(\bar{C}_p + \bar{\tau}_1 + \bar{\tau}_2\right)\lambda_p : (2.18), (2.19), \lambda_p \geq 0, \forall p \in \mathcal{P}\right\}$$

admits a trivial optimal solution $\hat{\lambda}$ defined as:

$$\hat{\lambda}_p = \begin{cases} \bar{v}, & \text{if } p = p^* \text{ and } \bar{C}_{p^*} < -\bar{\tau}_1 - \bar{\tau}_2 \\ \underline{v}, & \text{if } p = p^* \text{ and } \bar{C}_{p^*} \geq -\bar{\tau}_1 - \bar{\tau}_2 \\ 0, & \text{otherwise,} \end{cases}$$

where p^* is an optimal solution of the pricing problem [PP] defined over $(\bar{\rho}, \bar{\tau}_1, \bar{\tau}_2)$.

Now, knowing the optimal value $\tilde{z}'$ of the RMP and the optimal value of pricing problem [PP] allows to compute $L(\bar{\rho})$, i.e., a lower bound on the optimal value of the MP.

In a BPC algorithm, the MP associated with each node of the branch-and-bound tree is solved by a column generation procedure where a lower bound on the optimal value of the MP can be computed at each iteration. Hence, the column generation procedure can be terminated as soon as the lower bound exceeds the best incumbent value $\hat{z}$ available for [SC], i.e., $L(\bar{\rho}) \geq \hat{z}$. In addition, in the specific case where all coefficients of the objective function of [SC] are integer, the column generation can be terminated when the ceiling of lower bound $L(\bar{\rho})$ is greater than or equal the ceiling of the optimal value of the RMP, i.e., $\lceil L(\bar{\rho}) \rceil \geq \lceil \tilde{z} \rceil$.

2.4.7 Valid inequalities

As mentioned in Section 2.2.3, valid inequalities are commonly used to strengthen the MP. According to the taxonomy introduced in de Aragao & Uchoa (2003), in a BPC algorithm, the inequalities can be classified into *robust* or *non-robust* depending on the impact they have on the pricing problem.

Robust inequalities do not modify the structure of the pricing problem, and their dual variables can be easily included in the cost of the arcs of graph $\mathcal{G}$ when solving the pricing problem (see Section 2.4.2). The generic form of a robust inequality is

$$\sum_{p\in\mathcal{P}}\sum_{(i,j)\in\mathcal{A}} d_{ij}b_{ij}^{p}\lambda_p \geq d, \tag{2.44}$$

where b_{ij}^{p} is a binary parameter with value one if path $p \in \mathcal{P}$ traverses arc $(i,j) \in \mathcal{A}$ and zero otherwise, $d_{ij} \in \mathbb{Z}$ is a coefficient associated with arc $(i,j) \in \mathcal{A}$ and $d \in \mathbb{Z}$ is the right hand-side.

If we denote by β the associated dual variable, the reduced cost of a λ_p variable is then

$$\bar{C}_p = C_p - \sum_{j\in\mathcal{J}} a_j^p\rho_j - \tau_1 - \tau_2 - \beta\sum_{(i,j)\in\mathcal{A}} d_{ij}b_{ij}^{p}.$$

Hence, to take into account the additional term on the cost $\bar{C}_{ij}$ of arcs $(i,j) \in \mathcal{A}$ in graph $\mathcal{G}$, it suffices to subtract $d_{ij}\beta$.

In general, inequalities expressed in terms of the arc flow variables x_{ij} of a compact formulation lead to robust valid inequalities. Indeed, it is easy to see that an inequality of the form $\sum_{(i,j)\in\mathcal{A}} d_{ij}x_{ij} \geq d$ can be transformed into an inequality of the form of (2.44).

Example 2.15 (Rounded capacity cuts for the CVRP)**.** *In the case of the CVRP, the* capacity cuts *(Laporte* et al.*, 1985) are an example of robust valid inequalities. For each subset of customers $\mathcal{S}$, such inequalities impose a lower bound on the number of vehicles to serve the customers in the subset:*

$$\sum_{p\in\mathcal{P}}\left(\sum_{(i,j)\in\delta^{-}(\mathcal{S})} b_{ij}^{p}\right)\lambda_p \geq \left\lceil\frac{\sum_{j\in\mathcal{S}} D_j}{Q}\right\rceil \qquad \forall\mathcal{S}\subseteq\mathcal{J}, \tag{2.45}$$

where D_j denotes the demand of customer j and Q is the vehicle capacity.

Conversely, non-robust inequalities modify the structure of the pricing problem and increase its complexity. Indeed, the states of dynamic programming equation (2.40) get more attributes, and consequently, the state space enlarges. In other words, additional resources are required to manage the dual variables of these constraints when solving the pricing problem via a labelling algorithm. Note that while robust inequalities can also be expressed in terms of the arc flow variables, non-robust inequalities can usually

only be expressed in terms of the λ variables. Hence they have no direct correspondence in a compact formulation. For this reason, such inequalities can only be considered in a BPC algorithm, not in a B&C one. Non-robust inequalities also usually permit to improve the quality of the lower bound significantly, which is why it is worth adding such cuts despite the difficulty of their management in the pricing problem.

The type and the number of additional resources depends on the specific family of valid inequalities. In the following, we report, as an example, the management of the *subset-row inequalities* introduced in Jepsen *et al.* (2008). The interested reader may refer to Contardo *et al.* (2014) or Costa *et al.* (2019) for the management of other families of non-robust valid inequalities.

Example 2.16 (Subset-row inequalities for the CVRP)**.** *Given a subset $\mathcal{S} \subseteq \mathcal{V}$ and a multiplier $p_i \geq 0$ for each $i \in \mathcal{S}$, the subset-row inequalities are expressed as*

$$\sum_{p \in \mathcal{P}} \left\lfloor \sum_{i \in \mathcal{S}} p_i a_i^p \right\rfloor \lambda_p \leq \left\lfloor \sum_{i \in \mathcal{S}} p_i \right\rfloor, \qquad \mathcal{S} \subseteq \mathcal{V}. \tag{2.46}$$

Pecin et al. *(2017) provide a management of the subset-row inequalities, which is also valid in the case where the elementarity of the paths is relaxed. In the label definition, we include an attribute for each inequality whose dual variable is different from zero.*

Given a subset $\mathcal{S} \subseteq \mathcal{V}$ corresponding to a subset-row inequality, let us denote by $\sigma_{\mathcal{S}} \leq 0$ the corresponding dual variable, and by $M(\mathcal{S})$ an attribute added to the label definition. $M(\mathcal{S})$ models the fractional part of the coefficient of the variables in Constraint (2.46). The role of $M(\mathcal{S})$ is to tell when dual variable $\sigma_{\mathcal{S}}$ has to be discounted from the reduced cost of the partial path. Remark that the coefficients of the variables in the left hand-side of Constraint (2.46) are rounded to the nearest integer down. Hence, $\sigma_{\mathcal{S}}$ is to be discounted each time a partial path visits enough vertices of $\mathcal{S}$ to increase $M(\mathcal{S})$ above one. To sum up, when a label is extended to vertex $i \in \mathcal{S}$, attribute $M(\mathcal{S})$ is incremented by p_i and if $M(\mathcal{S}) \geq 1$, dual variable $\sigma_{\mathcal{S}}$ is discounted from the reduced cost, and $M(\mathcal{S})$ is decremented by one unit.

In addition, the dominance rules (2.42) need to be modified. Specifically, The second condition in (2.42) has to be replaced by

$$\bar{C} \leq \bar{C}' + \sum_{\substack{\mathcal{S} \in \mathcal{M} \\ M(\mathcal{S}) > M'(\mathcal{S})}} \sigma_{\mathcal{S}}, \tag{2.47}$$

where $\mathcal{M}$ *is the set of subsets* $\mathcal{S} \subseteq \mathcal{V}$ *representing the subset-row inequalities in the RMP whose dual variable* $\sigma_{\mathcal{S}}$ *is different from zero.*

Think of a subset-row inequality defined over set $\mathcal{S} = \{1, 2, 3\}$ *with multipliers* $p_i = 1/2$, $i = 1, 2, 3$. *Table 2.4 shows the value of attribute* $M(\mathcal{S})$ *and if dual variable* $\sigma_{\mathcal{S}}$ *is discounted from the reduced cost* $\bar{C}$ *for the labels associated to several partial paths permitting to build a path* $p = (s, 1, 4, 2, 1)$.

Table 2.4: Value of attribute $M(\mathcal{S})$ and discount of $\sigma_{\mathcal{S}}$ from the reduced cost for labels to build path p.

partial path	$M(\mathcal{S})$	$-\sigma_{\mathcal{S}}$ added to $\bar{C}$
(s)	0	no
$(s, 1)$	1/2	no
$(s, 1, 4)$	1/2	no
$(s, 1, 4, 2)$	0	yes
$(s, 1, 4, 2, 1)$	1/2	no

Now, we show why the modification in dominance rule is required. To do so, we consider partial path $p = (s, 1, 4, 2, 1)$ *and a second partial path* $p' = (s, 3, 4, 5, 1)$ *and we set* $\sigma_{\mathcal{S}} = -10$. *Attribute* $M(\mathcal{S})$ *is equal to* $1/2$ *for* p *and to* 0 *for* p'. *We assume that the reduced cost of* p *is* $\bar{C} = 95$, *the one of* p' *is* $\bar{C}' = 100$ *and that* p *dominates* p' *according to the dominance rules in (2.42). Hence, with no modification in the dominance rule, path* p' *would be erased from the list of labels associated with vertex* 1. *However, let us suppose to extend both paths to vertex* 2 *and let* $\bar{C}_{12} = -5$ *be the reduced cost associated with arc* $(1, 2)$. *The reduced cost of the extension of* p *becomes* $\bar{C} + \bar{C}_{12} - \sigma_{\mathcal{S}} = 100$, *whereas the one of* p' *becomes* $\bar{C}' + \bar{C}_{12} = 95$, *which contradicts the correctness of the dominance rules. Note that with the modification in Equation (2.47), path* p' *would have not been erased. Indeed, the modified condition is not satisfied:* $\bar{C} = 95 > \bar{C}' + \sigma_{\mathcal{S}} = 100 - 10 = 90$.

2.4.8 Branching scheme

The branching scheme ensures the correctness of a BPC algorithm, i.e., it guarantees that the algorithm terminates with an integer solution. The branching rules to adopt depend on the specific problem under consideration. In this section, we report some generic remarks about the branching rules and the different strategies to select them.

Imposing any branching rule entails including constraints in the MP and also applying modifications in the column generation procedure at the pricing level. In the

following, we denote by $\bar{\lambda}$ the fractional optimal solution at a node of the branch-and-bound tree.

Note that branching on variables $\bar{\lambda}_p$, $p \in \mathcal{P}$, as in a standard branch-and-bound algorithm, is highly inefficient: the resulting branch-and-bound tree is highly unbalanced. Indeed, on the one hand, imposing a constraint like $\bar{\lambda}_p = 1$, is strong: we impose a decision for all the vertices visited by $p \in \mathcal{P}$. On the other hand, imposing $\bar{\lambda}_p = 0$ is rather weak: it only forbids the use of path p. In addition, managing constraint $\bar{\lambda}_p = 0$ in the pricing problem is not easy as we need to prevent the pricing problem [PP] from producing path p again.

More efficient branching rules for the generic class of problems we consider are:

branching on the number of paths , i.e., on fractional value $g = \sum_{p \in \mathcal{P}} \bar{\lambda}_p$; such rule gives rise to two branches identified by the following constraints

$$\sum_{p \in \mathcal{P}} \lambda_p \leq \lfloor g \rfloor \qquad \sum_{p \in \mathcal{P}} \lambda_p \geq \lceil g \rceil.$$

Managing these constraints in the pricing is analogous to Constraints (2.18) and (2.19).

branching on the visits to the vertices given a vertex $j \in \mathcal{J}$, we branch on fractional value $v_j = \sum_{p \in \mathcal{P}} a_j^p \bar{\lambda}_p$. The two branches are identified by the following constraints

$$\sum_{p \in \mathcal{P}} a_j^p \lambda_p \leq \lfloor v_j \rfloor \qquad \sum_{p \in \mathcal{P}} a_j^p \lambda_p \geq \lceil v_j \rceil.$$

The management of these constraints in the pricing is similar to Constraints (2.17), hence, we do not provide further details. Note that if $\lfloor v_j \rfloor = 0$, then vertex j has to be removed from graph $\mathcal{G}$ when solving the pricing problems in the associated node.

branching on the flow on the arcs given an arc $(i, j) \in \mathcal{A}$, we branch on fractional value $f_{ij} = \sum_{p \in \mathcal{P}} b_{ij}^p \bar{\lambda}_p$, where b_{ij}^p is a binary parameter with value one if path $p \in \mathcal{P}$ traverses arc $(i, j) \in \mathcal{A}$ and zero otherwise. The two branches are identified by the following constraints

$$\sum_{p \in \mathcal{P}} b_{ij}^p \lambda_p \leq \lfloor f_{ij} \rfloor \qquad \sum_{p \in \mathcal{P}} b_{ij}^p \lambda_p \geq \lceil f_{ij} \rceil.$$

The dual variables associated with these two constraints are managed in the same manner in the pricing problem. Hence, let $\phi_{ij} \leq 0$ be the dual variable associated with the first of the two constraints. To express the reduced cost of a λ_p variable, it suffices to subtract ϕ_{ij} from the cost $\bar{C}_{ij}$ of arc (i, j) when solving the pricing problem. Again, if $\lfloor f_{ij} \rfloor = 0$, then arc (i, j) has to be removed from graph $\mathcal{G}$ when solving the pricing problem in the associated node.

Such rules might not be enough to ensure the correctness of the algorithm. Additional rules such as the *Ryan Foster* branching rules (Ryan & Foster, 1981) or valid inequalities (see, e.g., Gschwind *et al.*, 2019) may be required.

Branching rules are usually applied hierarchically, i.e., one at a time in a given order. Among the fractional values of a same branching rule, a common strategy is to select the one closest to 0.5.

2.4.9 Additional speed up techniques

In this section, we report some additional techniques to improve the performance of a BPC algorithm.

2.4.9.1 Column generation degeneracy

It is well known that the column generation procedure may suffer from slow convergence (see Desrosiers & Lübbecke, 2005). To explain this phenomenon, we need to look at the interpretation of column generation procedure from the dual point of view. In Section 2.2.2, we highlighted that a column generation procedure for a primal problem is interpreted as a cutting plane method in the associated dual problem. At each iteration of the procedure, new variables are added to the RMP, which correspond to new constraints included in the D-RMP. Recall that each time a constraint is included in the D-RMP, the current optimal dual solution is cut, i.e., becomes infeasible. At the early iterations of the column generation procedure, the D-RMP is poor in constraints. Hence, it is likely that several very different solutions of the D-RMP with the same costs (or very similar costs) exist. Consequently, when, at a given iteration, a constraint is included in the D-RMP to cut its current optimal solution, the new optimal solution of the D-RMP may have a similar cost, but the dual variable values may be completely different. They may oscillate between extreme values, i.e., given a dual variable, its

value is either very high or very low. This makes the convergence to the optimal solution of the D-MP not smooth. We clarify this phenomenon in the following example.

Example 2.17. *We consider an instance of the problem class with five vertices, i.e.,* $\mathcal{J} = \{1, 2, 3, 4, 5\}$. *For simplicity, we do not consider Constraints* (2.18) *and* (2.19) *in formulation [SC]. We initialise the MP with two paths: one visiting vertices* 1, 2 *and* 3 *with cost 100 and the other one visiting vertices* 1, 4 *and* 5 *with cost 120. In this example, we mimic a possible oscillating behaviour at the first iterations of a column generation procedure to solve the MP. We focus on the dual point of view. At the beginning of the column generation procedure the D-RMP is:*

$$\begin{aligned} \max \ & \rho_1 + \rho_2 + \rho_3 + \rho_4 + \rho_5 \\ s.t. \ & \rho_1 + \rho_2 + \rho_3 \leq 100 \\ & \rho_1 + \rho_4 + \rho_5 \leq 120 \\ & \rho_1, \rho_2, \rho_3, \rho_4, \rho_5 \geq 0, \end{aligned}$$

where dual variables ρ_1, ρ_2, ρ_3, ρ_4, *and* ρ_5 *are associated to the vertices in* $\mathcal{J}$. *At the first iteration of the procedure, the RMP and D-RMP optimal value is* 220. *An optimal dual solution associated to it is:* $\tilde{\rho}_1 = \tilde{\rho}_3 = \tilde{\rho}_5 = 0$, $\tilde{\rho}_2 = 100$ *and* $\tilde{\rho}_4 = 120$. *We suppose that the pricing problem returns a path of reduced cost equal to* -10 *and visiting vertices* 2 *and* 4. *This translates in adding constraint* $\rho_2 + \rho_4 \leq 210$ *in the D-RMP. At the second iteration, again, the optimal value of the RMP and D-RMP is* 220. *An optimal dual solution to attain such value is:* $\tilde{\rho}_1 = \tilde{\rho}_2 = \tilde{\rho}_4 = 0$, $\tilde{\rho}_3 = 100$ *and* $\tilde{\rho}_5 = 120$. *Now, we suppose the pricing problem returns a path of cost* -5 *and visiting vertices* 3 *and* 5. *Constraint* $\rho_3 + \rho_5 \leq 215$ *is added in the D-RMP. Finally, at the third iteration, the optimal value of the RMP and D-RMP is* 220. *An optimal dual solution to attain such value is:* $\tilde{\rho}_1 = \tilde{\rho}_3 = \tilde{\rho}_4 = 0$, $\tilde{\rho}_2 = 100$ *and* $\tilde{\rho}_5 = 120$.

We observe that in these three iterations of the column generation procedure no progress is made: the optimal value of the RMP did not change. In addition, the values of the dual variables take extreme values from one iteration to the next one.

Now, recall that the convergence of the primal is guided by the optimal dual solutions of the D-RMP. Indeed, as mentioned in Remark 2.5 in Section 2.2.2, the expression of the reduced cost of the primal variables involves the optimal dual solution. In our context, dual variables are associated to vertices. Therefore, when solving the pricing problem, few vertices may have a very high dual price while the others have a very

small one, and the generated negative reduced cost paths would tend to contain only those vertices with an overestimated cost w.r.t. their optimal cost at the end of the column generation. This slows down the process since the quality of the columns reported in the RMP is not good. So even if the pricing problem seems to provide very interesting columns, the value of the RMP decreases very slightly.

A simple idea to limit this phenomenon is to provide a good initialisation of the column subset before starting the execution of the BPC algorithm. How to compute such columns depends on the specific problem at hand. Heuristic or greedy algorithms should be considered to generate columns such that each vertex is covered by several of them. Note that in the dual this allows us to have enough constraints to guide the solution. However, the initial number of columns must remain small w.r.t. the size of the instance to avoid solving too large RMPs in the column generation procedure.

A more sophisticated methodology to deal with this phenomenon is to consider *stabilisation techniques*. Such techniques aim to guide the trajectory towards a dual optimal solution, avoiding jumps from one iteration to the other. Several different stabilisation techniques have been proposed through the years. In the early work of Marsten (1975), the values of a dual solution are constrained to be in "boxes" defined around the values of the previous dual solution. This is achieved by considering additional constraints of the form $l_j \leq \rho_j \leq u_j$ in the D-RMP where u_j and l_j represent the bounds of the box for dual variable ρ_j. This implies considering additional variables in the RMP. Note that to prove the optimality of the RMP, the dual variables need to be unconstrained, i.e., $\rho_j \in]l_j; u_j[$. In the RMP, it means the additional variables have value zero.

Stabilisation techniques based on adding constraints in the dual problem, the so-called *dual inequalities*, have also been proposed by Amor *et al.* (2006), Gschwind & Irnich (2016) and Haghani *et al.* (2022). Another technique entails "correcting" the values of a dual solution based on the values of the previous solution (see Wentges (1997), Neame (2000) and Pessoa *et al.* (2018)). Conversely, Du Merle *et al.* (1999), Briant *et al.* (2008), and Amor *et al.* (2009) proposed stabilisation techniques based on applying penalties in the dual objective function for the deviation from an incumbent dual solution. Recently, a technique based on adding redundant constraints in the primal problem was proposed by Costa *et al.* (2022).

Embedding such techniques in a column generation procedure is usually not trivial. Moreover, their effectiveness is sometimes unclear: Costa *et al.* (2022) used the

technique of Pessoa *et al.* (2018) without obtaining significant results.

2.4.9.2 Primal heuristics

In the execution of a B&B or a BPC algorithm, having an upper bound $\bar{z}$ of good quality helps in reducing the size of the tree (see `pruning` tests in lines 5 and 12 of Algorithm 4). Typically, upper bounds are detected quite deep in the branch-and-bound tree, when enough branching constraints allow to determine an integer solution. However, it would be beneficial to detect good quality upper bounds at the early levels of the tree. For this reason, primal heuristics, i.e., auxiliary procedures to quickly detect integer solutions for formulation [SC] may be applied. Such procedures are commonly called when a node is solved. A compromise between the time spent in the heuristic and the quality of the upper bound should be assessed with computational experiments.

In this section, we describe a simple and well-known heuristic: the *Restricted Master Heuristic* (RMH). We refer to Sadykov *et al.* (2019) for a thorough presentation on primal heuristics.

The RMH entails restricting formulation [SC] to the current subset of variables identified in the execution of the BPC algorithm and solve it as a monolithic integer program with a commercial solver. Although set covering/packing/partitioning formulations are well managed by commercial solver, a time limit on the execution of the solver should be imposed. In addition, the previous current best incumbent integer solution should be given as a warm-start to the solver to speed up the resolution. The RMH can be called each time a node of the branch-and-bound tree is processed (Archetti *et al.*, 2013). However, we can reduce the calls of such a heuristic by imposing that between two consecutive calls to the RMH, the BPC has to identify at least a given number of new columns.

2.4.9.3 Pricing heuristics

An efficient management of the pricing problem is a critical component in a BPC algorithm, in particular when the pricing problem is modelled as an ESPPRC.

In some applications, solving the ESPPRC directly by means of an exact algorithm may be time consuming even with the speed-up strategies described in Section 2.4.4 and the ng-paths relaxation presented in Section 2.4.5. Hence, more sophisticated solution schemes which employ heuristic algorithms to solve the ESPPRC may be considered to

solve the MP at each node of the branch-and-bound tree. Observe that an exact pricer is needed only to prove the optimality of the MP that is, to prove no negative reduced cost columns exist. Any procedure able to provide negative reduced cost columns can be employed as long as the exact pricer is called to prove the optimality or provide further negative reduced cost columns when the heuristic procedures fail to do so. In this section, first, we report two of these solution schemes. Then, we discuss three simple and generic heuristic algorithms that may be considered in such schemes.

We denote by $h_1, h_2, \ldots, h_H$ the algorithms to solve the ESPPRC in the order in which we want to apply them. Typically, the algorithms are ordered from the fastest to the slowest. We suppose that $h_1, \ldots, h_{H-1}$ are heuristics and h_H is an exact algorithm for the ESPPRC. Hence, being the exact prices the last one to be applied, we ensure that the MP is solved to optimality.

The first solution scheme to solve the MP at each node of the branch-and-bound tree consists in considering sequentially each algorithm $h_l, l = 1, \cdots, H$ to solve the ESPPRC over the iterations of the column generation. At the beginning, the ESPPRC is solved with the heuristic h_1. When, at a given iteration of the column generation procedure, solving the ESPPRC with heuristic h_l does not yield any negative reduced cost columns, the ESPPRC is then solved with heuristic h_{l+1}. Heuristic h_l will never be considered anymore to solve the ESPPRC in the next iterations of the column generation. Hence, in the last iterations, only the exact algorithm is applied to solve the ESPPRC.

Conversely, the second solution scheme works as follows: at each iteration of the column generation procedure, the ESPPRC is solved by algorithms $h_1, h_2, \ldots, h_H$ in this order until one of them provides negative reduced cost columns or exact algorithm h_H certifies that the current solution of the MP is optimal.

In the following, we report three simple and generic heuristic algorithms. The first algorithm entails imposing an early termination on the exact algorithm. Precisely, once the exact algorithm identifies a given number of negative reduced cost columns, its execution is stopped and such columns are inserted in the RMP. We refer to Pessoa *et al.* (2020) for an implementation of this algorithm. The second heuristic algorithm exploits the fact that the pricing problem entails finding paths in a graph. The algorithm follows the ideas presented by Toth & Vigo (2003) to reduce the size of the graph where the exact algorithm is applied. Specifically, each vertex $j \in \mathcal{J}$ is assigned with a neighbourhood of size g containing the closest g customers to j. A label associated

with vertex j can be extended only to vertices in the neighbourhood of j. This heuristic is implemented by preprocessing the arcs of graph $\mathcal{G}$: all the arcs $(j, i) \in \mathcal{A}$ where i is a vertex not in the neighbourhood of j are removed. The ESPPRC is solved considering a sequence of increasing neighbourhood sizes g. The value of g is incremented when the associated algorithm does not provide any negative reduced cost column. For example, we refer to Gschwind *et al.* (2019) and Petris *et al.* (2023) for more details about the procedure. Finally, the third algorithm exploits the fact that we solve the ESPPRC by means of a labelling algorithm. In such algorithms, the dominance checks are time consuming. A heuristic version of the dominance rules may be applied: the test on the unreachable vertices $\mathcal{U} \subseteq \mathcal{U}'$ is removed from (2.42) (see Desaulniers, 2010).

2.4.9.4 Strong branching

In this section, we present a more sophisticated selection strategy for the branching rules presented in Section 2.4.8, the so-called *strong branching* introduced in Applegate *et al.* (1995). Røpke (2012) embed such procedure in a BPC algorithm for vehicle routing problems. Such a selection strategy entails spending time to evaluate the branching candidates associated with a fractional solution of the MP at a node of the branch-and-bound tree. For example, given the branching rule on the visits to the vertices, a branching candidate is a vertex $j \in \mathcal{J}$ whose associated value v_j is fractional (see Section 2.4.8). Usually, there are several branching candidates with similar fractional values, the strong branching strategy aims to evaluate the potential good candidates before selecting one. The evaluation of a candidate exploits the column generation procedure. Precisely, each candidate gives rise to two child nodes c_1 and c_2 in the branch-and-bound tree. A few iterations of the column generation procedure are performed in both child nodes to obtain two values of the RMP, $\tilde{z}'_1$ and $\tilde{z}'_2$, respectively. Such values are used to assign a score $sc(c_1, c_2)$ to the candidate. Finally, the candidate with the best score is selected. Commonly, the score is computed according to the *product rule* proposed by Achterberg (2007):

$$sc(c_1, c_2) = \max\{\epsilon, \tilde{z}'_1 - \tilde{z}\} \times \max\{\epsilon, \tilde{z}'_2 - \tilde{z}\}$$

where $\tilde{z}$ is the value of the MP at the current node of the branch-and-bound tree. The idea of the score is to provide an estimation on the increase of the lower bound if the

candidate is selected to be solved. Indeed, for each children $i = 1, 2$, value $\tilde{z}'_i - \tilde{z}$ is an overestimation of the lower bound increase if child c_i is solved to optimality.

This procedure is rather costly in terms of computational time. Hence, it should be used only in the lowest levels of the branch-and-bound tree, i.e., in nodes of the tree close to the root. In addition, remark that taking good decisions in the lowest levels of the tree has a higher impact on the efficiency of the solution approach. A successful implementation of the strong branching rule appeared recently in Pessoa *et al.* (2020). Following Pessoa *et al.* (2020), we detail an example of a simple two-rounds strong branching procedure:

Round 1. At most d branching candidates are evaluated according to the score presented above, where the values of the RMPs associated with the child nodes are computed without generating any new column. The three candidates with the highest score move to round 2.

Round 2. The three candidates from round 1 are evaluated with more computational effort by applying a strategy similar to the previous round. The scores are computed in the same way, the difference is that some iterations of the column generation procedure are performed in order to have a better estimation of the lower bound increase if the candidate is selected. The candidate with the highest score is finally selected.

2.5 Final remarks

In this section, we provide some final remarks for an interested reader who wants to develop a BPC algorithm. The bottleneck of a BPC algorithm for a problem in the class we consider is usually the procedure to solve the pricing problem to optimality, in particular in the presence of the elementarity constraints for the paths. For this reason, the first step to devise an efficient BPC algorithm should be dedicated to the design and implementation of an efficient exact pricing algorithm. We stress that particular attention has to be given to the implementation details, such as the choice of data structures and the memory usage. Then, such an algorithm should be embedded in a column generation procedure to solve the linear relaxation of formulation [SC], i.e., the MP at the root node of the branch-and-bound tree. Here, the objective is to evaluate the quality of the lower bound at the root node. These values can be compared with

known upper bounds or optimal values from the literature if such exist. Otherwise, one might implement a primal heuristic at the root node like the Restricted Master Heuristic. In addition, the inclusion of valid inequalities can be assessed at this point by evaluating the improvement they bring to the lower bound. First, robust valid inequalities can be assessed since they are easy to manage in the pricing and will not deteriorate performance. Then, if the quality of the lower bound at the root node is not very good, non-robust inequalities can be added.

If the column generation convergence is slow, the following solutions can be applied.

1. Provide a good set of columns to the RMP at initialisation. By considering the values of the dual variables one should check they are not oscillating too much.

2. Design a set of pricing heuristics to quickly generate negative reduced cost columns during the first iterations of the column generation procedure.

3. Consider to add stabilisation techniques if the convergence is still slow.

Finally, the branching rules should be implemented in order to explore the branch-and-bound tree to provide an optimal solution. If many nodes are explored in the branch-and-bound tree, one might investigate adding more cuts and applying strong branching.

Chapter 3

A heuristic with a performance guarantee for the Commodity constrained Split Delivery Vehicle Routing Problem

Contents

The content of this chapter was presented at the following workshop: ROUTE 2022. This chapter corresponds to the paper "A heuristic with a performance guarantee for the Commodity constrained Split Delivery Vehicle Routing Problem" submitted to Networks An international journal on 28 November 2022 and received major revision on 18 May 2023.

Abstract: The Commodity constrained Split Delivery Vehicle Routing Problem (C-SDVRP) is a routing problem where customer demands are composed of multiple commodities. A fleet of capacitated vehicles must serve customer demands in a way that minimizes the total routing costs. Vehicles can transport any set of commodities and customers are allowed to be visited multiple times. However, the demand for a single commodity must be delivered by one vehicle only. In this work, we developed a heuristic with a performance guarantee to solve the C-SDVRP. The proposed heuristic is based on a set covering formulation, where the exponentially-many variables correspond to routes. First, a subset of the variables is obtained by solving the linear relaxation of the formulation by means of a column generation approach which embeds a new pricing heuristic aimed to reduce the computational time. Solving the linear relaxation gives a valid lower bound used as a performance guarantee for the heuristic. Then, we devise a restricted master heuristic to provide good upper bounds: the formulation is restricted to the subset of variables found so far and solved as an integer program with a commercial solver. A local search based on a mathematical programming operator is applied to improve the solution. We test the heuristic algorithm on benchmark instances from the literature. Several new (best-known) solutions are found in reasonable computational time. The comparison with the state of the art heuristics for solving C-SDVRP shows that our approach significantly improves the solution time, while keeping a comparable solution quality.

Keywords: Vehicle routing problems, Multiple commodities, Split delivery, Column generation, Matheuristic, Pricing heuristic.

3.1 Introduction

Splitting customer demands has proven to be beneficial in reducing the transportation costs and the number of vehicles (see Archetti *et al.*, 2006a, 2016). A first work in this direction is the article by Dror & Trudeau (1989). The authors introduced the *Split Delivery Vehicle Routing Problem* (SDVRP), where customer demands are composed of a single commodity and can be split among any number of vehicles. This problem and its variants have been widely studied and exact (e.g. Archetti *et al.*, 2014; Belenguer *et al.*, 2000; Desaulniers, 2010; Munari & Savelsbergh, 2022) and heuristic algorithms (e.g. Archetti *et al.*, 2006b; Bortfeldt & Yi, 2020; Chen *et al.*, 2016b; Silva *et al.*, 2015) were proposed. Among these, Chen *et al.* (2016a) studies a particular case of the SDVRP, where customer demands are discretised a priori.

Although this delivery policy brings remarkable cost savings when compared with the policy where no splits are allowed, it is hardly applicable from a practical point of view. Indeed, customers are usually not keen to accept an unconstrained split delivery Archetti *et al.* (2016). One step in the direction of making the split deliveries more adherent to real-world logistics is made in Gulczynski *et al.* (2010). The authors proposed a variant of the SDVRP where the quantity delivered to each customer has to be greater than a preset minimum amount.

Finally, under a multi-commodity setting, a delivery policy that might reduce customer inconvenience due to split deliveries is to allow demands to be split by commodity. Whenever a vehicle delivers a commodity to a customer, the entire quantity associated with the commodity has to be provided. This policy was firstly studied in the *Discrete Split Delivery Vehicle Routing Problem* (DSDVRP) proposed in Nakao & Nagamochi (2007) to deal with a real-life case study. The problem was formally introduced in the literature under the name of *Commodity constrained Split Delivery Vehicle Routing Problem* (C-SDVRP) in Archetti *et al.* (2016). In the C-SDVRP, a minimum cost set of routes have to be determined such that the customer demands, composed of multiple commodities, are met, and the capacity of the vehicles is respected. The authors showed that the C-SDVRP is a relaxation of the *Capacitated Vehicle Routing Problem* (CVRP) where all commodities of each customer are delivered with a single vehicle, and a restriction of the SDVRP. In addition, they proposed an in-depth analysis to assess the benefits of the C-SDVRP in terms of cost savings and applicability in comparison with the CVRP and the SDVRP. To do so, they introduced the first compact mathematical formulation and devised a branch-and-cut and a heuristic algorithm

to solve it. Finally, they introduced a first set of benchmark instances characterised by 15, 20, 40, 60, 80 or 100 customers requiring 2 or 3 commodities.

Despite its practical relevance, the literature on the C-SDVRP and its variants is quite limited. An exact approach for the C-SDVRP was proposed in Archetti *et al.* (2015). Specifically, the authors modelled the problem by means of a set covering formulation and devised a first branch-price-and-cut (BPC) algorithm. They formulate the pricing problem as an *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC) and solve the *ng-path* relaxation by means of a label setting dynamic programming technique. Gschwind *et al.* (2019) enhanced the performances of the BPC algorithm of Archetti *et al.* (2015) by embedding new procedures as the implicit bidirectional labelling search to solve the ESPPRC, the separation of non-robust valid inequalities to strengthen the lower bound, and the stabilization of the column generation procedure via dual-optimal inequalities. The authors extended the test-bed introduced in Archetti *et al.* (2016) with 336 new instances with 4, 5 and 6 commodities. The enhanced BPC algorithm outperformed the one of Archetti *et al.* (2015), being faster and providing several new optima and better lower bounds.

Conversely, Gu *et al.* (2019) focused on a heuristic algorithm for the C-SDVRP and proposed an adaptive large neighbourhood search (ALNS) that exploits the inherent characteristics of the problem. Specifically, several existing local search moves were adapted to better deal with the multi-commodity aspect, and a mathematical programming operator was developed to reassign commodities to routes. The authors assess the performance of their ALNS on the test-bed introduced in Archetti *et al.* (2016). The ALNS found the optimal value for 81 out of the 84 instances with 15 and 20 customers, and provided 344 new best-known solutions for the 380 instances with more than 40 customers. In Soleilhac (2022) the authors propose a small and large neighbourhood search (SLNS) which is capable of solving different variants of routing problems, among those the C-SDVRP. The SLNS is compared with the ALNS proposed in Gu *et al.* (2019) on 320 instances with 100 customers and 2 and 3 commodities. The SLNS found 155 new best-known solutions, while the computational time is on average three times the one of the ALNS.

Finally, variants of the C-SDVRP (or of the DSDVRP) have been studied, see e.g. Alinaghian & Shokouhi (2018); Guo *et al.* (2021); Mirzaei & Wøhlk (2019); Qiu *et al.* (2018); Salani & Vacca (2011); Wang *et al.* (2015); Zbib & Laporte (2020). In Salani & Vacca (2011), customer demands are composed of multiple items grouped in

orders. Each order can be seen as a commodity required by a customer in the C-SDVRP. In addition to the C-SDVRP, this variant includes time windows for the customers and considers service times that depend on the order delivered. The authors proposed a branch-and-price approach. An extension of the aforementioned problem in a pickup and delivery context is the *Vehicle Routing Problem with Discrete Split Deliveries and Pickups* proposed by Qiu *et al.* (2018). Mirzaei & Wøhlk (2019) study the effect of splitting customer demands by commodity in a multi-compartment vehicle routing problem. The authors propose a branch-and-price to solve this multi-compartment C-SDVRP. Zbib & Laporte (2020) addressed another multi-compartment C-SDVRP in the context of a capacitated arc routing problem arising in the collection of recyclable waste. Vehicles with multiple compartments may make multiple visits to the same household to collect different recyclables, however, the amount of a single recyclable cannot be split.

In this paper, we consider a set covering formulation for the C-SDVRP, where the exponential number of variables are related to routes. Generating all such variables is intractable. Hence, we propose a *restricted master heuristic* (Sadykov *et al.*, 2019) to solve the problem. This heuristic scheme consists in solving the formulation restricted to a subset of variables as a static integer program. Similar approaches have been successfully applied to deal with vehicle routing problems (see, e.g., Briant *et al.*, 2020; Muter *et al.*, 2010; Parragh & Schmid, 2013; Taillard, 1999). The main difference in the methodologies proposed in these works is the way the subset of variables is generated. Taillard (1999) and Muter *et al.* (2010) developed a tabu search heuristic to populate a subset of variables for the solution of a routing problem with a heterogeneous fleet of vehicles and the *Vehicle Routing Problem with Time Windows*, respectively. In Parragh & Schmid (2013), the authors devised a restricted master heuristic for the dial-a-ride problem: variables are generated by means of a hybrid column generation procedure where a variable neighbourhood search heuristic is employed to identify negative reduced cost columns. Finally, Briant *et al.* (2020) deals with the *Joint Order Batching and Picker Routing Problem* (JOBPRP). The authors proposed formulation with exponentially many variables and solve its linear relaxation by means of column generation. The objective is twofold: determining a subset of variables to use in a restricted master heuristic and calculating a lower bound on the optimal solution value.

The approach proposed in the current paper follows the strategy used in Briant *et al.* (2020). Unlike the existing literature on the C-SDVRP, our approach is a heuris-

tic that provides lower and upper bounds even for large-scale instances of the problem within reasonable computation times. The lower bound serves as a performance guarantee for our heuristic. In the column generation step, the pricing problem reduces to solve an ESPPRC. Efficient handling of the pricing problem is essential in a column generation procedure. Therefore, heuristics are commonly used to address the pricing problem before solving it exactly. In this respect, we devise a new pricing heuristic that exploits the multi-commodity aspect of the problem. More precisely, the heuristic articulates in two phases: **Phase 1** computes a set of promising customer sequences by solving the ESPPRC on a modified version of the pricing graph; **Phase 2** is called for each customer sequence produced by the first phase and determines all the negative reduced cost routes arising from the sequence by solving the *Shortest Path Problem with Resource Constraints* (SPPRC) on acyclic graphs. It is noteworthy that the first phase of our heuristic also provides a valid lower bound on the value of the pricing problem. After the column generation procedure, upper bounds are identified by the restricted master heuristic. Finally, a local search step is applied to improve the upper bound. This step uses the mathematical programming operator proposed in Gu *et al.* (2019) to reassign commodities to the routes. Computational experiments proved that our approach successfully provides upper bounds of good quality in shorter computational time than the state-of-the-art heuristic approaches. More precisely, it is capable of solving large size instances with four, five, and six commodities and improves few best-known solution values from the literature. When compared against the state-of-the-art heuristics of Gu *et al.* (2019) and Soleilhac (2022), our approach improves the solution time with an average speedup ratio of 17.0, while keeping the percentage gap with respect to the upper bounds to 0.55% on average.

The reminder of the paper is organized as follows. In Section 3.2, we give a formal description of the C-SDVRP and introduce the notation. In Section 3.3, we present a set covering formulation for the problem. In Section 3.4 we describe the main components of the restricted master heuristic we devised to solve it. Finally, the computational results obtained on the benchmark instances are reported and discussed in Section 3.5.

3.2 Problem description

In the *Commodity constrained Split Delivery Vehicle Routing Problem* (C-SDVRP) the commodities of a set $\mathcal{K} = \{1, \ldots, \kappa\}$ have to be delivered from a depot 0 to a set of

customers $\mathcal{N} = \{1, \ldots, n\}$. The request of a customer $j \in \mathcal{N}$ may be composed of multiple commodities and is identified by set $\mathcal{K}_j = \{k \in \mathcal{K} \colon D_{jk} > 0\}$, where $D_{jk} \geq 0$ is the demand of commodity $k \in \mathcal{K}$ to be delivered to customer j. An unlimited fleet performs the distribution of the commodities to the customers. Each vehicle has a capacity Q and is initially based at the depot. The vehicles can transport any subset of commodities provided that their capacity is not exceeded. We suppose without loss of generality that $Q \geq \max\{D_{jk} : j \in \mathcal{N}, k \in \mathcal{K}_j\}$. When a vehicle visits a customer $j \in \mathcal{N}$, a non-empty subset of commodities in $\mathcal{K}_j$ is delivered to j. Hence, a customer request may be split, and a customer may be visited multiple times. However, when a vehicle visits customer j, the amount of each commodity k delivered by the vehicle to j must be equal to D_{jk}. In other words, the demand for a single commodity cannot be split.

The C-SDVRP can be defined on a directed weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. The vertex set $\mathcal{V} = \{0\} \cup \mathcal{N}$ contains a vertex 0 representing the depot, and the set $\mathcal{N}$ of vertices representing the customers. The arc set $\mathcal{A} = \{(i, j) : i, j \in \mathcal{V}, i \neq j\}$ contains arcs modelling each possible vehicle travel between two distinct vertices. Each arc $(i, j) \in \mathcal{A}$ is associated with a non-negative cost C_{ij} which corresponds to the cost of traversing arc (i, j). We suppose that the arc costs satisfy the triangular inequality. A route in graph $\mathcal{G}$ is a non-empty circuit starting and ending at the depot. A route is *feasible* if the total amount of commodities delivered to the customers visited along the route does not exceed the vehicle capacity Q. The set of feasible routes is denoted by $\mathcal{R}$. The cost of a route r is $C_r = \sum_{(i,j)\in\mathcal{A}(r)} C_{ij}$, where $\mathcal{A}(r)$ is the set of arcs traversed by the route.

The C-SDVRP aims to find a least-cost set of feasible routes such that all the customer requests are served.

3.3 Problem formulation

We consider the set covering formulation proposed in Archetti *et al.* (2015). For each feasible route $r \in \mathcal{R}$, we introduce a binary coefficient a^r_{jk} with value one if commodity $k \in \mathcal{K}$ is delivered to customer $j \in \mathcal{N}$ by route r and zero otherwise. Then, for $r \in \mathcal{R}$, we introduce a binary variable λ_r taking value one if the route is selected in the solution and zero otherwise. Last, we define an auxiliary variable v to count the number of vehicles in the solution.

The Set Covering formulation [SC] reads as follows:

$$[\text{SC}] \quad \min \sum_{r \in \mathcal{R}} C_r \lambda_r \tag{3.1}$$

$$\text{s.t.} \sum_{r \in \mathcal{R}} a_{jk}^r \lambda_r \geq 1 \qquad \forall j \in \mathcal{N}, \forall k \in \mathcal{K}_j \tag{3.2}$$

$$\sum_{r \in \mathcal{R}} \lambda_r = v \tag{3.3}$$

$$\underline{v} \leq v \leq \bar{v} \tag{3.4}$$

$$\lambda_r \in \{0, 1\} \qquad \forall r \in \mathcal{R} \tag{3.5}$$

Objective function (3.1) minimizes the total routing costs. Constraints (3.2), which we refer to as *covering constraints*, ensure that the customer demands are met. Constraints (3.3) and (3.4) define an auxiliary variable v and impose a lower bound and an upper bound on it, i.e., on the number of vehicles used in the solution. Finally, constraints (3.5) define variables λ_r as binary.

Bounds in constraints (3.4) can be obtained by solving an instance of the *Bin Packing Problem* (BPP), where bins represent vehicles associated with their capacity, and objects, with the respective weights, are the customer demands. The BPP is formulated as an integer program and solved with a commercial solver within a time limit. Values $\underline{d}$ and $\bar{d}$ denote the obtained lower and upper bounds, respectively. Hence, the number of vehicles is bounded from below by the ceil function of $\underline{d}$ ($\underline{v} \coloneqq \lceil \underline{d} \rceil$) and from above by the minimum between twice value $\bar{d}$ (see Federgruen & Simchi-Levi, 1995) and the number of customers ($\bar{v} \coloneqq \min\{2\bar{d}, |\mathcal{N}|\}$).

3.4 A restricted master heuristic

This section describes the main components of the restricted master heuristic we designed to tackle [SC]. The heuristic scheme articulates in three steps.

In the first step, the Master Problem (MP), i.e., the linear relaxation of the formulation [SC] is solved using a column generation procedure (see, e.g., Desrosiers & Lübbecke (2005)) to obtain a subset of variables $\mathcal{R}' \subseteq \mathcal{R}$ and a valid lower bound. Afterwards, if the solution of the MP is fractional, valid inequalities are possibly included to strengthen the lower bound and enrich the set $\mathcal{R}'$. The procedure is then repeated. In the second step, an upper bound is obtained by solving formulation [SC] defined on

the variables of $\mathcal{R}'$ generated in the first step. Specifically, [SC] restricted to $\mathcal{R}'$ is solved as a static integer program with a commercial solver run within a time limit. Note that the set $\mathcal{R}'$ is preprocessed before solving [SC] to repair all the routes whose total amount of delivered commodities is not tight with respect to the vehicle capacity. For each of these routes, we randomly select commodities to be delivered to the customers they visit in order to generate new routes that fill the vehicle capacity. Finally, in the third step, the mathematical programming operator proposed in Gu *et al.* (2019) to reassign commodities to the routes is used to improve the upper bound determined in the second step.

3.4.1 Column generation

As mentioned above, we developed a column generation algorithm to solve the MP and populate a subset of variables $\mathcal{R}' \subseteq \mathcal{R}$. The restriction of the MP to $\mathcal{R}'$ is referred to as Restricted Master Problem (RMP). At each iteration of the procedure, the RMP and the pricing problem are solved sequentially. The pricing problem aims to either identify negative reduced cost variables (columns) to add to $\mathcal{R}'$ or to produce a certificate of optimality for the solution of the MP. We consider some heuristic approaches to quickly identify negative reduced cost variables when solving the pricing problem. Among others, we devised a novel pricing algorithm which exploits the multi-commodity aspect of the C-SDVRP. When the heuristic column generators do not yield negative reduced cost variables, we solve the pricing problem using an exact algorithm to produce a certificate of optimality. In addition, it allows us to compute the Lagrangian bound, a valid lower bound on the value of [SC] which we use to provide an optimality gap for the solution of the restricted master heuristic.

3.4.2 Pricing problem

In this section, we use the terms path and route interchangeably.

As in Archetti *et al.* (2015), at each iteration of the column generation procedure, we price out routing variables λ_r, $r \in \mathcal{R}$. The reduced cost of λ_r is given by:

$$\bar{C}_r = C_r - \sum_{j \in \mathcal{N}} \sum_{k \in \mathcal{K}_j} a^r_{jk} \rho_{jk} - \tau,$$

where ρ_{jk} and τ are the dual prices associated with constraints (3.2) and (3.3), respectively.

Archetti *et al.* (2015) showed that the pricing problem reduces to an *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC), where the resource is associated with the vehicle capacity. Following to some extent Gschwind *et al.* (2019), we formulate the ESPPRC on a directed multi-graph $\mathcal{G}' = (\mathcal{V}', \mathcal{A}')$ defined over the original graph $\mathcal{G}$ as follows. We include in vertex set $\mathcal{V}'$ two replica i' and i'' of each vertex $i \in \mathcal{V}$. In arc set $\mathcal{A}'$, we include an arc (i'', j') for each arc $(i, j) \in \mathcal{A}$ to model the trip of a vehicle from vertex i to vertex j. Finally, for all non-empty subsets of commodities $\mathcal{M}_j \subseteq \mathcal{K}_j$, we introduce an arc $(j', j'')^{\mathcal{M}_j}$ to model the delivery of $\mathcal{M}_j$ to customer j. The resource consumption $\bar{D}$ is set to zero ($\bar{D}_{i''j'} \coloneqq 0$) on arcs (i'', j'), whereas on arcs $(j', j'')^{\mathcal{M}_j}$, it is equal to the demand associated with commodity subset $\mathcal{M}_j$, i.e., $\bar{D}^{\mathcal{M}_j}_{j'j''} \coloneqq \sum_{k \in \mathcal{M}_j} D_{jk}$. Finally, the cost on arcs (i'', j') is $\bar{C}_{i''j'} \coloneqq C_{ij}$ if $i'' \neq 0''$ and $j' \neq 0'$, and $\bar{C}_{i''j'} \coloneqq C_{ij} - \tau/2$, otherwise. The cost on arcs $(j', j'')^{\mathcal{M}_j}$ considers the dual prices of constraints (3.2) associated with customer $j \in \mathcal{N}$ and the commodities $k \in \mathcal{M}_j$, that is $\bar{C}^{\mathcal{M}_j}_{j'j''} \coloneqq -\sum_{k \in \mathcal{M}_j} \rho_{jk}$.

Solving the pricing problem results in searching for a negative reduced cost elementary path in $\mathcal{G}'$ from $0''$ to $0'$ such that the resource consumption does not exceed the vehicle capacity Q.

3.4.3 Solution of the pricing problem

Negative reduced cost paths are retrieved in the multi-graph $\mathcal{G}'$ by solving the ESPPRC by means of a label setting dynamic programming algorithm (see Feillet *et al.*, 2004). More precisely, labels identify partial paths in $\mathcal{G}'$ starting at $0''$ and are characterised by the following resources: reduced cost, accumulated demand, set of visited customers, and for each customer, the subset of commodities delivered. The starting point of the procedure is a label associated with vertex $0''$ with resources set to zero or empty. Then, labels are propagated from a vertex to another while satisfying the elementarity and capacity constraints: each customer is visited at most once along a partial path , and the accumulated demand cannot exceed the vehicle capacity Q. Dominance rules are applied to prune unpromising labels.

In order to accelerate the solution of the pricing problem, we implemented some state-of-the-art procedures. Specifically, the first one is the *ng-path* relaxation (Baldacci *et al.*, 2011) which partially relaxes the elementarity constraint of the paths: a

neighbourhood is pre-assigned to each customer, and cycles are allowed only if the customer visited more than once in a path is not in the neighbourhood of its predecessor in that path. In addition, we incorporate an implicit version of the bidirectional labelling search algorithm proposed in Righini & Salani (2006). The labels are extended from vertex $0''$ to the other vertices of $\mathcal{G}'$ up to a value of the accumulated demand equal to $Q/2$. Then, the generated labels are merged to obtain complete paths (see Bode & Irnich, 2012, for more details).

However, even by embedding the two techniques mentioned above, the exact resolution of the pricing problem might be time consuming. We therefore proceed as follows. First, the pricing problem is solved with a new heuristic coupled with two reduced graph heuristics similar to those presented in Gschwind *et al.* (2019). Then, the same reduced graph heuristics are also applied on the multi-graph $\mathcal{G}'$. Finally, the exact pricing method is invoked. We switch from one pricing algorithm to the next one when the first fails to produce negative reduced cost paths.

In the following, we give a detailed description the heuristics mentioned above, together with the description of the preprocessing phase to reduce the size of the multi-graph $\mathcal{G}'$.

3.4.3.1 Preprocessing phase

We perform the preprocessing procedure proposed in Archetti *et al.* (2015) and Gschwind *et al.* (2019) to reduce the size of the multi-graph $\mathcal{G}'$. At each iteration of the column generation procedure, we only consider in $\mathcal{G}'$ arcs of type $(j', j'')^{\mathcal{M}_j}$ whose associated pair demand-cost $(\bar{D}^{\mathcal{M}_j}_{j'j''}, \bar{C}^{\mathcal{M}_j}_{j'j''})$ is Pareto-optimal. Since the number of commodities is small in the benchmark instances of the C-SDVRP ($|\mathcal{K}| \leq 6$), the Pareto-optimal commodity subsets can be computed by enumeration. The reader may refer to Gschwind *et al.* (2019) for a general procedure, based on the solution of the *Shortest Path Problem with Resource Constraints* (SPPRC) on acyclic graphs, to determine such subsets when the enumeration strategy is not applicable.

3.4.3.2 A new two-phase pricing heuristic

The heuristic we propose to solve the pricing problem consists of two phases. Phase 1 aims to compute a set of promising customer sequences. To do so, we solve the ESPPRC on a modified graph of reduced size compared with the multi-graph $\mathcal{G}'$. Solving the ESPPRC on such a graph is not only faster than solving it on $\mathcal{G}'$, but Phase

1 also permits to derive a valid lower bound on the pricing problem value. Phase 2 aims to determine all negative reduced cost paths arising from each of the customer sequences provided by Phase 1. We solve the SPPRC on an acyclic graph for each customer sequence. The topology and size of graph $\mathcal{G}'$ allow to perform such operation in negligible time.

The graph used in Phase 1, denoted by $\mathcal{G}'' = (\mathcal{V}', \mathcal{A}'')$, differs from the multi-graph $\mathcal{G}'$ in the arcs modelling the deliveries to the customers: in $\mathcal{G}''$ a unique subset of commodities can be delivered to each customer. Hence, $\mathcal{G}''$ is obtained from $\mathcal{G}'$ by removing, for each customer $j \in \mathcal{N}$, all the arcs of type $(j', j'')^{\mathcal{M}_j}$ but one, which we denote by (j', j''). The demand and cost are set on these arcs so that whenever a customer $j \in \mathcal{N}$ is visited the least consuming commodity is delivered and all profitable dual prices related to j are collected. Hence, they are defined as $\bar{D}_{j'j''} \coloneqq \min\{D_{jk} : k \in \mathcal{K}_j\}$ and $\bar{C}_{j'j''} \coloneqq -\sum_{k \in \mathcal{K}_j} \rho_{jk}$, respectively.

This definition of demand and cost permits to derive the following properties.

Proposition 3.1. *All feasible solutions of the ESPPRC on multi-graph $\mathcal{G}'$ are feasible solutions for the ESPPRC on graph $\mathcal{G}''$.*

Proposition 3.2. *The optimal solution of the ESPPRC on graph $\mathcal{G}''$ provides a lower bound on the optimal value of the ESPPRC on multi-graph $\mathcal{G}'$.*

Corollary 3.3. *If the optimal value of the ESPPRC on the reduced graph $\mathcal{G}''$ is positive then the optimal value of the ESPPRC on multi-graph $\mathcal{G}'$ is positive as well.*

In Phase 2, we determine all negative reduced cost routes arising from each customer sequence (path) generated in Phase 1. We do this by solving the SPPRC on an acyclic multi-graph for each path. Specifically, let $p = (j''_0 = 0'', j'_1, j''_1, \ldots, j'_{l(p)-1}, j''_{l(p)-1}, j'_{l(p)} = 0')$ be a path produced in Phase 1, where $l(p)$ denotes the length of p. The acyclic multi-graph $\mathcal{G}'(p) = (\mathcal{V}'(p), \mathcal{A}'(p))$ associated with p is defined as follows. $\mathcal{V}'(p)$ is the vertex st that includes only vertices visited along p, i.e., $\mathcal{V}'(p) = \{j''_0 = 0'', j'_1, j''_1, \ldots, j'_{l(p)-1}, j''_{l(p)-1}, j'_{l(p)} = 0'\}$. $\mathcal{A}'(p)$ contains the arcs of $\mathcal{G}'$ connecting each vertex in $\mathcal{V}'(p)$ to its successor in p, i.e. (i) arcs (j''_h, j'_{h+1}), $h = 0, \ldots, l(p) - 1$ to model the travel from j_h to j_{h+1}, and (ii) arcs $(j'_h, j''_h)^{\mathcal{M}_{j_h}}$, $h = 1, \ldots, l(p) - 1$, $\mathcal{M}_{j_h} \subseteq \mathcal{K}_j$ to model the deliveries of subsets of commodities $\mathcal{M}_{j_h}$ to customer j_h.

The negative reduced cost routes arising from path p correspond to the negative cost paths in $\mathcal{G}'(p)$ from $j_0 = 0''$ to $j_{l(p)} = 0'$, which satisfy the capacity constraint. These paths are determined by solving the SPPRC on the multi-graph $\mathcal{G}'(p)$. Although

solving the SPPRC on acyclic graphs is NP-hard (see Di Puglia Pugliese & Guerriero (2013)), the size and particular topology of multi-graphs $\mathcal{G}'(p)$ allow to do this operation very efficiently in terms of computational time.

In the following, we provide an example to illustrate how the proposed pricing heuristic works.

Example 3.4. *We consider a C-SDVRP instance with three customers* $\mathcal{N} = \{1, 2, 3\}$ *and three commodities* $\mathcal{K} = \{1, 2, 3\}$*: customer* 1 *requires the commodity of* $\mathcal{K}_1 = \{1\}$*, with* $D_{11} = 2$*; customer* 2 *requires the commodities of* $\mathcal{K}_2 = \{1, 2, 3\}$*, with* $D_{21} = 2$*,* $D_{22} = 4$ *and* $D_{23} = 3$ *; customer* 3 *requires the commodities of* $\mathcal{K}_3 = \{1, 3\}$*, with* $D_{31} = 2$ *and* $D_{33} = 1$*. We assume the travelling cost from the depot to the customers and between customers to be unitary. The vehicle capacity is set to* 10*.*

Figure 3.1 shows the pricing multi-graph $\mathcal{G}' = (\mathcal{V}', \mathcal{A}')$ *arising from such instance at a certain iteration of the column generation procedure. The consumption and cost on arcs of type* $(i'', j') \in \mathcal{A}'$ *modelling the movement of the vehicle from one vertex to another are* $(\bar{D}_{i''j'}, \bar{C}_{i''j'}) = (0, 1)$*. Differently, the consumption and cost on arcs of type* $(j', j'')^{\mathcal{M}_j} \in \mathcal{A}'$ *modelling the delivery to the customers are reported in the figure with the following notation:* $\mathcal{M}_j$*:*$(\bar{D}^{\mathcal{M}_j}_{j'j''}, \bar{C}^{\mathcal{M}_j}_{j'j''})$*. We only consider the Pareto-optimal deliveries to the customers.*

The graph $\mathcal{G}'' = (\mathcal{V}'', \mathcal{A}'')$ *built at Phase 1 is shown in Figure 3.2a. The consumption and cost on arcs of type* $(i'', j') \in \mathcal{A}'$ *are as in* $\mathcal{G}'$ *and those on arcs* $(j', j'') \in \mathcal{A}'$ *are displayed in the figure with the same convention as in Figure 3.1. In the first phase,we solve the ESPPRC on the graph* $\mathcal{G}''$ *to obtain all non-dominated negative cost paths (customer sequences) that respect the vehicle capacity. The second phase of the heuristic identifies the negative reduced cost routes arising from each of these sequences by solving the ESPPRC on acyclic multi-graphs. As an example, we show how this is done on the most negative path found in phase one (in red in Figure 3.2a), i.e., on* $p = (0'', 2', 2'', 3', 3'', 0')$ *with consumption and cost equal to* 3 *and* -13*, respectively. The acyclic multi-graph* $\mathcal{G}'(p)$ *associated with path* p *is shown in Figure 3.2b. In* $\mathcal{G}'(p)$*, all the possible deliveries to customers* 2 *and* 3 *are restored. Finally, the SPPRC is solved on* $\mathcal{G}'(p)$ *to obtain all non-dominated feasible routes with negative reduced costs. We obtain six routes that visit customers* 2 *and* 3 *in the order imposed by path* p *and deliver either subset of commodities* $\{1\}$ *or* $\{1, 2\}$ *to customer* 2*, both combined with all the possible deliveries to customer* 3*. The route with the most negative reduced cost (in red in Figure 3.2b) delivers* $\{1, 2\}$ *to customer* 2 *and* $\{1, 3\}$ *to customer* 3*. Its consumption and reduced cost are* 9 *and* -11*, respectively.*

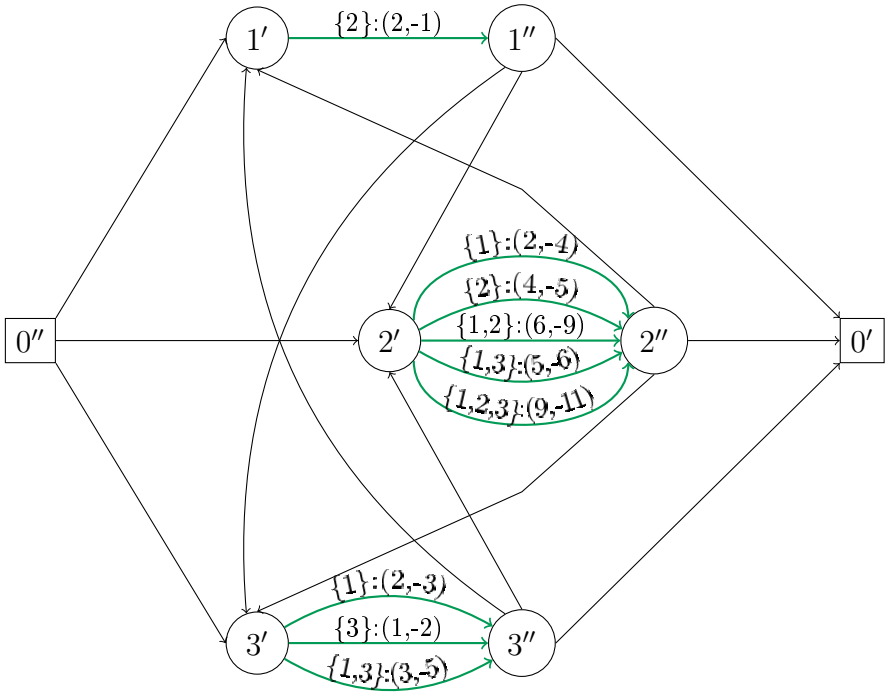


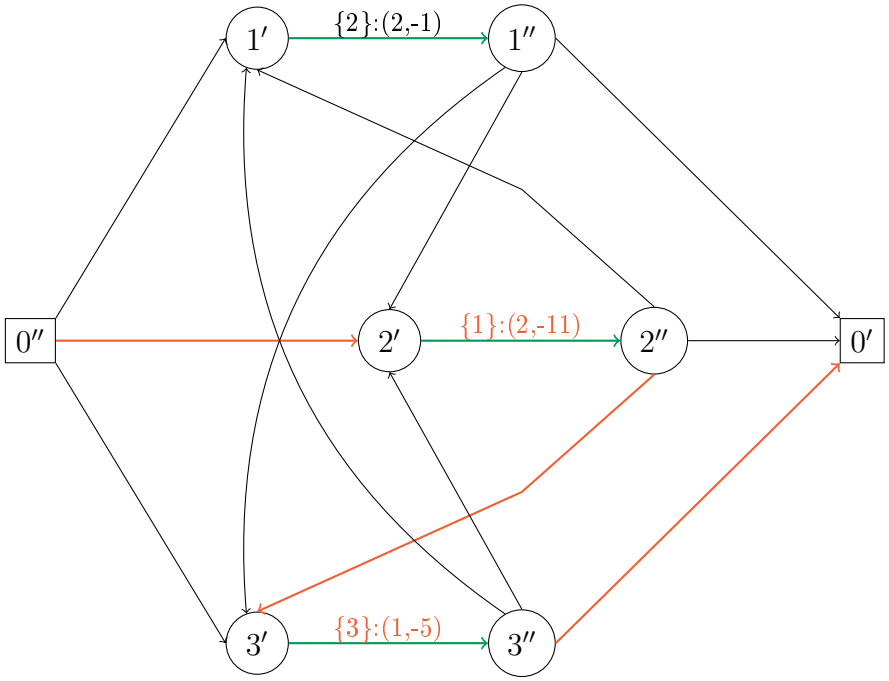
Figure 3.1: Pricing multi-graph $\mathcal{G}'$ for the C-SDVRP instance defined in Example 3.4.

3.4.3.3 Reduced graph heuristics

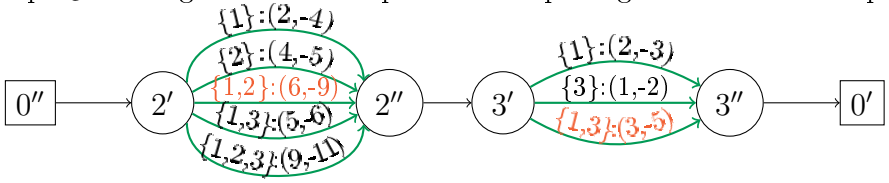
In this section, we present two classical heuristics based on the reduction of the size of the pricing graph. They are applied to the multi-graph $\mathcal{G}'$ and to the graph $\mathcal{G}''$ of the first phase of heuristic, we just described. We discuss them for the case of $\mathcal{G}'$, knowing that the case of $\mathcal{G}''$ can be treated similarly.

The first heuristic is inspired by Toth & Vigo (2003) and limits the possibilities of moving between customers. Specifically, a neighbourhood containing the g closest customers is built for each customer $j \in \mathcal{N}$. A partial path ending in j can only be extended to customers belonging to its neighbourhood. This is implemented by removing from $\mathcal{G}'$ all arcs of type (j'', l') such that l does not belong to the neighborhood of j. The pricing problem is solved considering a sequence of increasing neighbourhood sizes: $g = 3, 6, 10, |\mathcal{N}|$. The value of g is incremented when the associated pricing problem produces no negative reduced cost path.

The second heuristic is specifically designed to handle the multi-commodity aspect of the C-SDVRP. Indeed, it aims at reducing the delivery possibilities to customers. Specifically, we impose an upper bound b on the number of customers whose demand can be split per path. This strategy is motivated by the analysis carried out in Archetti



(a) Graph $\mathcal{G}''$ arising from the first phase of the pricing heuristic in Example 3.4.



(b) Acyclic multi-graph $\mathcal{G}'(p)$ where $p = (0'', 2', 2'', 3', 3'', 0')$ arising from the second phase of the pricing heuristic in Example 3.4.

Figure 3.2: Graphs of the first and second phase of the novel pricing heuristic built in Example 3.4.

et al. (2015) on optimal solutions of the C-SDVRP. They note that in most of them the number of split deliveries is less than three. To count the number of split customers in a path, we introduce an integer resource s in the label definition. The value of s is initially set to zero and is incremented by one unit along arcs of type $(j', j'')^{\mathcal{M}_j}$, if $\mathcal{M}_j$ does not correspond to the full delivery to j ($\mathcal{M}_j \subsetneqq \mathcal{K}_j$). On the other arcs, the value of s is simply propagated. Once s reaches the bound b, all the following customers visited in the path are delivered only with subset $\mathcal{K}_j$, i.e., only arcs $(j', j'')^{\mathcal{K}_j}$ are considered. As for the previous heuristic, we consider an incremental procedure relying on a sequence of increasing upper bounds: $b = 0, 1, 2, 3, \infty$.

3.4.4 Valid inequalities

We consider a family of robust valid inequalities, the so-called *capacity cuts*:

$$\sum_{r \in \mathcal{R}} \left(\sum_{(i,j) \in \delta^-(\mathcal{S})} b_{ij}^r \right) \lambda_r \geq \left\lceil \frac{\sum_{j \in \mathcal{S}} \sum_{k \in \mathcal{K}_j} D_{jk}}{Q} \right\rceil \qquad \forall \mathcal{S} \subseteq \mathcal{N}, \tag{3.6}$$

where $\delta^-(\mathcal{S}) = \{(i,j) \in \mathcal{A} : i \notin \mathcal{S}, j \in \mathcal{S}\}$ is the set of arcs of graph $\mathcal{G}$ having their final extremity in $\mathcal{S}$ and b_{ij}^r is a binary coefficient taking value one if route $r \in \mathcal{R}$ traverses the arc $(i,j) \in \mathcal{A}$.

If the solution of the MP is fractional, we separate the capacity cuts (3.6). Since the separation of these inequalities is NP-hard, we do so by means of the heuristic algorithms presented in Ralphs *et al.* (2003), namely the *extended shrinking heuristic* and the *greedy shrinking heuristic*. The violated cuts are included in the RMP, and the associated dual prices $\pi_{\mathcal{S}}$ are incorporated in the definition of the reduced cost of the variables λ_r, and then considered in the pricing problem solution.

3.4.5 Initialization of the set $\mathcal{R}'$

We initialize the set of routes $\mathcal{R}'$ to avoid starting the column generation procedure with large dual prices, which usually slows down the pricing problem resolution. Specifically, for each customer $j \in \mathcal{N}$, we include a round trip (0-j-0) delivering the commodities of each subset $\mathcal{M}_j \subseteq \mathcal{K}_j$ requested by j, feasible with respect to the capacity Q. Moreover, we insert in $\mathcal{R}'$ the routes obtained by applying a variant of the Clarke-Wright algorithm (CW) Clarke & Wright (1964). Precisely, we modified the randomized CW algorithm

proposed in Battarra *et al.* (2008) to take into account the multi-commodity aspect of the C-SDVRP. We set a limit of 20 runs.

3.4.6 Local search

In this section, we present the local search we implement to improve the C-SDVRP solution provided by the restricted master heuristic. Specifically, we consider the Mathematical Programming Operator (MPO) proposed in Gu *et al.* (2019) to reassign the commodities of a specific customer $j \in \mathcal{N}$ to the routes of the solution. We iteratively call the MPO for each customer in $\mathcal{N}$.

Let $j \in \mathcal{N}$ be a customer. We introduce the following notation. We denote by $\bar{\mathcal{R}}^{-j}$ the set of routes in the current C-SDVRP solution where all the visits to customer j are removed. More precisely, $\bar{\mathcal{R}}^{-j}$ contains the routes of the current C-SDVRP solution which do not visit j and, for the ones that visit j, $\bar{\mathcal{R}}^{-j}$ contains a copy of those routes where j has been removed from the sequence of visited customers. For each $r \in \bar{\mathcal{R}}^{-j}$, Q_r^j denotes the residual capacity in route r. Finally, we indicate by C_r^j the cost of the cheapest insertion of customer j in route $r \in \bar{\mathcal{R}}^{-j}$.

The MPO consists in solving a *Capacitated Facility Location Problem* (CFLP) (see Mirchandani & Francis, 1990) where all commodities $k \in \mathcal{K}_j$ of customer j have to be assigned to the routes (facilities) of $\bar{\mathcal{R}}^{-j}$ at minimum insertion costs and such that residual capacities of the routes are not exceeded.

The integer program on which the MPO is based makes use of the following decision variables. For each $k \in \mathcal{K}_j$ and each $r \in \bar{\mathcal{R}}^{-j}$, we introduce a binary variable

$$y_{kr}^j = \begin{cases} 1 & \text{if commodity } k \text{ is delivered to customer } j \text{ by route } r \\ 0 & \text{otherwise.} \end{cases}$$

In addition, for each $r \in \bar{\mathcal{R}}^{-j}$ we include a binary variable z_r^j defined as follows

$$z_r^j = \begin{cases} 1 & \text{if route } r \text{ delivers to customer } j \text{ at least one commodity} \\ 0 & \text{otherwise.} \end{cases}$$

Note that j refers to a specific customer and is not used as an index for the variables.

The integer program is for customer j as follows:

$$\min \sum_{r \in \bar{\mathcal{R}}^{-j}} C_r^j z_r^j \tag{3.7}$$

$$\text{s.t.} \sum_{r \in \bar{\mathcal{R}}^{-j}} y_{kr}^j = 1 \qquad \forall k \in \mathcal{K}_j \tag{3.8}$$

$$\sum_{k \in \mathcal{K}_j} D_{jk} y_{kr}^j \leq Q_r^j z_r^j \qquad \forall r \in \bar{\mathcal{R}}^{-j} \tag{3.9}$$

$$y_{kr}^j \in \{0,1\} \qquad \forall k \in \mathcal{K}_j, \forall r \in \bar{\mathcal{R}}^{-j} \tag{3.10}$$

$$z_r^j \in \{0,1\} \qquad \forall r \in \bar{\mathcal{R}}^{-j} \tag{3.11}$$

The objective function (3.7) minimizes the total insertion cost. Constraints (3.8) guarantee that all commodities of customer j are covered by exactly one route. Constraints (3.9) ensure that if some commodities of $\mathcal{K}_j$ are added to a route, their demand do not exceed the remaining capacity of the route. Finally, Constraints (3.10) and (3.11) are the binary requirements.

3.5 Computational experiments

Our algorithm is implemented in C++ and compiled in release mode under a 64-bit version of MS Visual Studio 2019. CPLEX 12.9.0 (64-bit version) is used to solve the RMP in the column generation procedure and the restricted version of formulation [SC]. All experiments are carried out on a 64-bit Windows machine equipped with a Intel(R) Xeon(R) Silver 4214 processor with 24 cores hyper-threaded to 48 virtual cores, with a base clock frequency of 2.2 GHz, and 96 GB of RAM. A time limit of one hour and a single thread are imposed for each run of the algorithm.

In the following, we denote by LB and UB, respectively, the lower and upper bounds returned by our restricted master heuristic. The percentage optimality gap is defined as $100((UB - LB)/LB)$. The percentage gap with respect to a best-known solution value UB^{bk} from the literature is computed as $100((UB - UB^{bk})/UB^{bk})$. More precisely, values UB^{bk} are retrieved from Archetti *et al.* (2016); Gschwind *et al.* (2019); Gu *et al.* (2019) or Soleilhac (2022). Finally, all the solution times are expressed in seconds.

In this section, we first describe the benchmark instances, then, we measure the impact of the new two-phase pricing heuristic on the performance of the overall solution

algorithm. Later, we present the results on the whole testbed. Finally, we compare our algorithm with the two existing heuristic approaches from the literature providing the majority of the best-known solution values.

3.5.1 Benchmark instances

We tested our restricted master heuristic on the benchmark instances for the C-SDVRP proposed by Archetti *et al.* (2016) and Gschwind *et al.* (2019). The instances are divided in three groups: small ($|\mathcal{N}| = 15$), mid-size ($|\mathcal{N}| = 20, 40, 60, 80$) and large ($|\mathcal{N}| = 100$). In each small and mid-size instance, customers' locations are taken from the C101 and R101 Solomon's instances (Solomon, 1987), whereas, in each large instance, locations are taken from the Solomon's RC101 instances. In addition, the following parameters define the instances: (i) number of commodities $|\mathcal{K}|$; (ii) probability p that a customer requires a commodity with a non-zero demand; (iii) interval Δ to select the non-zero demand of a commodity required by a customer, expressed as a percentage of vehicle capacity; (iv) percentage α of vehicle capacity with respect to the maximum demand ($Q = \alpha \max\{\sum_{k \in \mathcal{K}_j} D_{jk} : j \in \mathcal{N}\}$). Table 3.1 summarises the values of these parameters which characterise each group of instances.

Table 3.1: Characteristics of the small, mid-size and large instances.

Group	Number of instances	Values of the parameters					
		$\lvert\mathcal{N}\rvert$	customers' locations	$\lvert\mathcal{K}\rvert$	p	Δ	α
small	160	15	C101, R101	2, 3, 4, 5, 6	0.6, 1.0	[40, 60], [1, 100]	1.1, 1.5, 2.0, 2.5
mid-size	320	20, 40, 60, 80	C101, R101	3, 4, 5, 6	0.6, 1.0	[1, 100]	1.5
large	900	100	C101, R101, RC101	2, 3, 4, 5, 6	0.6, 1.0	[40, 60], [1, 100]	1.1, 1.5, 2.0, 2.5

Finally, we report an issue related to some of the benchmark instances' name with 100 customers, which are called C101, whereas their correct customer's location is the one corresponding to Solomon's instance RC101. This has been corrected in the new database made available at `https://hal.inria.fr/hal-03836982`. In addition, we mention that 40 instances with 100 customers, customer's location RC101, and 2 and 3 commodities are available at `https://logistik.bwl.uni-mainz.de/forschung/benchmarks/`, but have not been tested in any of the former papers dealing with the problem, i.e, in Archetti *et al.* (2016); Gschwind *et al.* (2019); Gu *et al.* (2019) and Soleilhac (2022). Thus, we exclude this subset of instances when showing the comparison with benchmark approaches.

3.5.2 Impact of the novel pricing heuristic

In this section, we measure the impact of the novel pricing heuristic presented in Section 3.4.3.2 on the performance of the overall solution algorithm. To do so, we define `RMH-2P` (`RMH-N2P`) to be the variant of the resticted master heuristic presented in Section 3.4 where we enable (disable) the two-phase pricing heuristic. We test the two variants on a subset of 180 instances characterised by 100 customers and 4 commodities, which is representative for the whole testbed.

The results obtained by comparing `RMH-2P` against `RMH-N2P` are shown in Table 3.2. We group the considered instances by value of Δ, i.e., by the interval where the commodity demands are selected. Hence, for each group of instances, we have two columns associated with `RMH-2P` and `RMH-N2P`, respectively. The rows of the table are: *avg. exact pricing it.*: average number of iterations of the exact pricing algorithm; *avg. UB*: average upper bound value; *avg. t[s]*: average solution time in seconds.

Table 3.2: Impact of the two-phase pricing heuristic on the instances with $|\mathcal{N}| = 100$ and $|\mathcal{K}| = 4$.

	$\Delta = [40, 60]$		$\Delta = [1, 100]$	
	`RMH-2P`	`RMH-N2P`	`RMH-2P`	`RMH-N2P`
avg. exact pricing it.	69.37	111.20	128.39	168.42
avg. UB	3 170.03	3 175.56	2 359.43	2 353.60
avg. t[s]	530.17	692.17	1900.58	1942.55

First, we report that both variants are capable of providing a lower bound for all instances. Variant `RMH-2P` yields better results on the instances characterised by $\Delta = [40, 60]$. Indeed, it allows to reduce the iterations of the exact pricing algorithm by 38% and, consequently, the solution time of on average by 23%. In addition, `RMH-2P` provides upper bounds of slightly better quality (see row *avg. UB*). Differently, both variants behave in a similar manner on the instances with $\Delta = [1, 100]$. Although `RMH-2P` shows a good reduction of 24% of the iterations of the exact pricing algorithm, the reduction of the solution time of `RMH-2P` w.r.t. the one of `RMH-N2P` is rather slim (2%). This means that the additional time spent by the two-phase heuristic in `RMH-2P` is not effective enough in lightening the burden on the reduced graph heuristics and on the exact pricing algorithm.

From these results, we infer that the performance of `RMH-2P` heavily depends on the interval of commodity demands Δ. The reason lies in how Phase 1 of the two-phase

pricing heuristic is designed. The main idea of Phase 1 is to reduce the combinatorics in the solution of the ESPPRC due to the multi-commodity aspect of the C-SDVRP. Indeed, each customer is delivered with its least consuming commodity and all the profitable dual prices associated with the customer are collected. Clearly, the quality of such approximation is highly affected by the variability of the commodity demands. If $\Delta = [40, 60]$, Phase 1 provides a reasonable approximation of the benefit of serving a customer, whereas this might not be the case if $\Delta = [1, 100]$. Indeed, in the latter case, when a customer is visited, we might collect all the profitable dual prices against a very small consumption of the resource associated with capacity.

In light of the analysis conducted in this section, we apply variant `RMH-2P` to obtain results on the whole testbed. In the following, such variant will be simply denoted by `RMH`.

3.5.3 Results on the whole testbed

In this section, we discuss the results obtained by the restricted master heuristic (`RMH`) on the 1380 benchmark instances. Due to the large number of instances, the results are presented in an aggregated form. The detailed instance-wise version can be found at https://hal.inria.fr/hal-03836982. We compare the results with best-known solution values from the state-of-the-art exact and heuristic methods for the C-SDVRP available in the literature, namely Archetti *et al.* (2016); Gschwind *et al.* (2019); Gu *et al.* (2019) and Soleilhac (2022). Note that in Section 3.5.4, we compare in more details `RMH` against Gu *et al.* (2019) and Soleilhac (2022).

Table 3.3 shows the results obtained by `RMH` on the small and mid-size instances. Each row of the table corresponds to a subset of instances with the same number of customers and commodities. The first four columns report some information regarding the instance subsets: $|\mathcal{N}|$: number of customers; $|\mathcal{K}|$: number of commodities; *avg.*$\#CC$: average number of customer-commodities ($\sum_{j \in \mathcal{N}} |\mathcal{K}_j|$) per instance; $\#$: number of instances in the subsets. The remaining eight columns of the table summarise the results of the `RMH`: $\#LB$: number of instances for which a LB is found; *opt. gap[%] avg./min./max.*: average/minimal/maximal optimality gap expressed as a percentage; *avg. t[s]*: average solution time in seconds; $\#opt$: number of optima identified by `RMH` with respect to the ones identified by Gschwind *et al.* (2019); $\#equal$: number of times `RMH` returned the best-known solution values from the literature; $\#impr.$: number of times `RMH` improved the best-known solution values from the literature (considering

also the *new solutions*, i.e., the cases where no solution was available in the literature); *avg. gap[%].*: average percentage gap with respect to the best-known solution values; if no best-know solution values are available, we write symbol -.

Table 3.3: Results on the small and mid-size instances.

Instances				RMH results								
					opt. gap[%]					best known		
$\|\mathcal{N}\|$	$\|\mathcal{K}\|$	avg.#CC	#inst.	#LB	avg.	min.	max.	avg.t[s]	#opt	#equal	#impr.	avg. gap[%]
15	2	26.00	32	32	0.29	0.00	2.16	0.32	30	30	0	0.02
	3	36.50	32	32	0.68	0.00	5.89	3.98	22	22	0	0.23
	4	48.56	32	32	0.43	0.00	2.19	14.59	29	29	0	0.04
	5	60.13	32	32	0.84	0.00	6.93	57.30	18	18	0	0.41
	6	72.66	32	32	0.63	0.00	2.51	87.25	20	20	1	0.12
20	3	48.70	20	20	0.81	0.00	2.45	4.52	10	10	0	0.19
	4	64.10	20	20	0.94	0.00	2.70	33.54	9	9	1	0.24
	5	80.50	20	20	0.97	0.00	4.70	137.27	11	12	2	0.12
	6	96.10	20	20	0.98	0.04	2.76	467.27	11	11	2	0.28
40	3	98.10	20	20	1.69	0.24	2.79	58.76	2	2	0	0.41
	4	129.10	20	20	1.90	0.79	4.16	243.24	0	0	9	0.76
	5	160.40	20	20	2.35	0.58	5.27	749.61	1	1	14	0.71
	6	192.45	20	20	2.74	0.73	5.91	1627.37	0	0	17	0.27
60	3	145.00	20	20	2.27	1.14	3.38	170.76	0	0	0	0.76
	4	193.70	20	20	2.66	0.62	4.89	556.42	0	0	18	0.51
	5	240.10	20	20	3.46	0.92	6.21	1848.11	0	0	20	-
	6	287.70	20	20	4.66	1.71	7.70	2467.48	0	0	20	-
80	3	195.20	20	20	2.75	1.47	5.02	354.51	0	0	2	0.98
	4	256.40	20	20	3.74	1.52	8.34	1045.47	0	0	20	-
	5	320.80	20	20	4.31	2.31	7.46	2268.34	0	0	20	-
	6	386.75	20	20	6.16	2.37	13.28	3044.60	0	0	20	-

Note that RMH provides a lower bound for all the small and mid-size instances.

For the instances with $|\mathcal{N}| = 15$ and $|\mathcal{N}| = 20$, the branch-price-and-cut algorithm of Gschwind *et al.* (2019) provides 158 and 71 optima over 160 and 80 instances, respectively. RMH is able to identify 119 of them if $|\mathcal{N}| = 15$ and 41 of them if $|\mathcal{N}| = 20$. In respectively 50 and 6 of such cases, RMH proves the optimality of the obtained solutions (opt. gap = 0). Overall solutions are good, being the optimality gap on average equal to 0.69% and larger than 2.5% only in nine instances out of 240. In addition, RMH provides good solution values since the average gap with respect to the best-known solution is 0.18%. Six new best solutions are found (see column *#impr.*) and the optimality gap referred to these six new best solution is on average 2.09%.

For the instances with $|\mathcal{N}| = 40, 60, 80$, the number of commodities has a great impact on the measure of quality of the solutions found by RMH: the greater the number of commodities, the larger the average optimality gap (see columns *opt. gap[%]*). However, the number of instances for which the optimality gap exceeds 5% is limited

(35 out of 240) and most of them correspond to instances with 80 customers and a number of commodities larger than four. For these instances, no further insight can be drawn in terms of solution quality, indeed, no solution is available in the literature. RMH equals three best-known solutions and improves 160 of them (see columns *#equal.* and *#impr.*). Note that, out of those 160, RMH provides a new solution for 39, 58 and 60 instances with 40, 60 and 80 customers, respectively. The average optimality gap of such solutions is 2.72%, 3.68% and 4.73%. The gap w.r.t. the best-known solution values is on average equal to 0.70% and larger than 2.5% only in four cases.

In general, RMH runs within relatively short computational times on the small and mid-size instances. Also, Table 3.3 shows that the average solution time grows with the number of commodities and the number of customers.

Table 3.4 shows the results obtained on the large instances ($|\mathcal{N}| = 100$) which are grouped by number of commodities $|\mathcal{K}|$, interval Δ of customer demand and probability p that a customer requires a commodity. The remaining columns reporting the instance details are the same as in Table 3.3. The columns which summarise the results of the RMH are: *#LB*: number of instances for which a LB is found; *opt. gap[%] avg./min./max.*: average/minimal/maximal optimality gap expressed in percentage; *avg.t[s]*: average solution time in seconds; *#impr.*: number of times RMH improved the best-known solution values from the literature (counting also the new solutions); *gap[%] avg./min./max.*: average/minimal/maximal percentage gap with respect to the best-known solution values; if no best-know solution values are available, we write symbol -.

First, we note that RMH manages to provide a lower bound for 874 of the 900 instances. The 26 instances where it fails to do so are characterised by $|\mathcal{K}| = 5, 6$ and 25 of these also by $p = 1$ and $\Delta = [1, 100]$, i.e., they count 500 or 600 non-zero customer demands. When a lower bound is found, the optimality gap mirrors the behaviour observed in Table 3.3: it deteriorates as the number of commodities increases. More precisely, it is usually higher when $p = 1$, and especially when also $\Delta = [1, 100]$. In addition, the optimality gap is larger than 5% in 241 instances and, among those, 191 are characterised by five and six commodities. For the instances with fewer commodities, the average optimality gap is 2.71%, meaning that RMH yields good performance guarantee. For the ones with $|\mathcal{K}| = 2, 3$, RMH shows a behaviour comparable with the approaches from the literature yielding the best-known solution values, namely Archetti *et al.* (2016), Gu *et al.* (2019) and Soleilhac (2022). Indeed,

Table 3.4: Results on the large instances.

Instances					RMH results						best known		
						opt. gap[%]					gap[%]		
$\|\mathcal{K}\|$	Δ	p	avg.#CC	#inst.	#LB	avg.	min.	max.	avg.t[s]	#impr.	avg.	min.	max.
2	[40,60]	0.6	136.40	40	40	1.36	0.03	3.39	30.49	0	0.43	0.00	1.74
		1	200.00	50	50	1.39	0.09	3.12	68.28	11	0.39	-0.05	1.10
	[1,100]	0.6	136.40	40	40	2.40	1.06	5.24	183.20	0	0.95	0.04	2.87
		1	200.00	50	50	1.94	0.54	3.95	250.62	16	0.48	-0.23	2.27
3	[40,60]	0.6	188.40	40	40	1.77	0.70	3.22	155.42	1	0.45	-0.15	1.79
		1	300.00	50	50	2.46	0.74	7.22	247.15	12	1.09	-0.60	5.17
	[1,100]	0.6	188.40	40	40	2.46	0.67	5.54	459.32	2	0.89	-0.11	3.49
		1	300.00	50	50	3.51	1.50	6.71	995.53	13	1.62	-0.45	4.48
4	[40,60]	0.6	242.95	40	40	2.42	1.00	5.00	379.13	40	-	-	-
		1	400.00	50	50	3.91	0.93	9.11	651.01	50	-	-	-
	[1,100]	0.6	243.30	40	40	3.29	0.80	7.33	1100.49	40	-	-	-
		1	400.00	50	50	5.10	2.05	9.61	2540.64	50	-	-	-
5	[40,60]	0.6	299.13	40	40	3.66	1.65	6.19	957.39	40	-	-	-
		1	500.00	50	50	5.19	2.54	10.08	1609.24	50	-	-	-
	[1,100]	0.6	301.48	40	40	4.82	1.66	13.01	2196.31	40	-	-	-
		1	500.00	50	45	5.85	2.99	11.50	3479.58	50	-	-	-
6	[40,60]	0.6	359.65	40	40	4.78	2.03	8.05	1764.76	40	-	-	-
		1	600.00	50	50	6.43	2.95	11.98	2641.67	50	-	-	-
	[1,100]	0.6	357.73	40	39	6.04	3.14	15.24	2972.40	40	-	-	-
		1	600.00	50	30	7.37	4.99	11.89	3622.95	50	-	-	-

the gap against the best-known solution values is on average 0.79% and it is zero for five instances with two commodities. Our approach provides a new solution for the 540 instances with $|\mathcal{K}| = 4, 5, 6$. In addition, `RMH` manages to improve the value of 55 instances with $|\mathcal{K}| = 2, 3$, 40 of these are new solutions. More insights about the comparison against the existing heuristic methods in the literature are drawn in Section 3.5.4.

We conclude this section with a discussion regarding the price to pay to have a performance guarantee, i.e., to compute good lower bounds. In the case of `RMH`, providing lower bounds entails solving the exact pricing algorithm within the column generation procedure and the separation of the capacity cuts (3.6).

To analyse the impact of computing the lower bound, we store the upper bounds and computational time before the first iteration of the exact pricing algorithm. Note that such upper bounds are obtained by solving formulation [SC] restricted to the subset of columns found so far, and applying the mathematical programming operator. Such results are labelled in Table 3.5 with `RMH-NG`. The rows of the table represent subset of instances which share the same number of customers. The columns are: $|\mathcal{N}|$: number

of customers; *#inst.*: number of instances; *avg.UB*: average upper bounds of RMH-NG and RMH; *avg.gap[%]*: average percentage gap between the upper bounds of RMH with respect to the ones of RMH-NG; *avg.t[s]*: average solution time in seconds of RMH-NG and RMH; *avg.ratio*: ratio between the solution time of RMH with respect to the one of RMH-NG;

As expected, the price to compute a performance guarantee is paid in terms of computational time: the one of RMH is on average 3.9 times the one of RMH-NG. The additional time spent in RMH serves not only to compute good lower bounds, but also to improve the quality of the upper bounds. Indeed, several more columns are generated in RMH and the improvement of the upper bounds of RMH with respect to the ones of RMH-NG is on average 0.6%. These results shows that RMH can be easily adapted according to the needs: in case a solution is needed within a short computing time, RMH-NG can be used without sacrificing solution quality to a large extent.

Table 3.5: Impact of the performance guarantee.

Instances		RMH-NG vs. RMH					
		avg. UB			avg.t[s]		
$\|\mathcal{N}\|$	#inst.	RMH-NG	RMH	avg. gap[%]	RMH-NG	RMH	avg. ratio
15	160	389.35	386.69	-0.65	9.92	32.69	3.93
20	80	703.78	693.76	-1.41	41.47	160.65	4.07
40	80	1152.61	1139.68	-1.14	135.62	669.74	5.67
60	80	1659.90	1644.11	-0.96	254.19	1260.69	4.72
80	80	2126.85	2113.77	-0.66	361.21	1678.23	4.45
100	900	2793.71	2780.10	-0.38	382.29	1348.10	3.64

3.5.4 Comparison with Gu *et al.* (2019) and Soleilhac (2022)

In this section, we analyse how RMH performs against the two existing heuristic algorithms available in the literature providing the majority of the best-known solution values, namely Gu *et al.* (2019) and Soleilhac (2022). Remark that, differently from Gu *et al.* (2019) and Soleilhac (2022), our approach also provides a performance guarantee on the solution values. The authors of Gu *et al.* (2019) consider instances with two and three commodities, given that the ones with more commodities were not yet available. Among the instances used in Gu *et al.* (2019), Soleilhac (2022) considers only the ones with 100 customers. Hence, in this section, the testbed is restricted accordingly. The

experiments in Gu *et al.* (2019) and Soleilhac (2022) are carried on machines with similar characteristics to our when run with a single thread.

Tables 3.6 and 3.7 report the comparison against Gu *et al.* (2019) and Soleilhac (2022), respectively. The rows of the tables correspond to subset of instances with the same number of customers and commodities. The first three columns report the number of customers ($|\mathcal{N}|$), commodities ($|\mathcal{K}|$) and instances (*#inst.*) in each subset. The next five columns compare the upper bounds reporting the number of instances where `RMH` matches (*#equal*) or improves (*#impr.*) the upper bounds from the competitor and the average/minimal/maximal gap (*gap[%] avg./min./max.*). The final three columns show the average solution times (*avg.t[s]*) of `RMH`, the one of the compared heuristic and the average ratio between the solution times of the compared heuristic w.r.t. the ones of `RMH` (*avg.ratio*).

Table 3.6: Comparison with Gu *et al.* (2019) on the instances with $|\mathcal{K}| = 2, 3$ and customers' locations from C101 and R101.

Instances			`RMH` vs. Gu *et al.* (2019)							
			UB							
					gap[%]			avg.t[s]		
$\|\mathcal{N}\|$	$\|\mathcal{K}\|$	#inst.	#equal	#impr.	avg.	min.	max.	`RMH`	Gu *et al.* (2019)	avg.ratio
15	2	32	30	0	0.02	0.00	0.28	0.32	8.91	35.12
15	3	32	22	0	0.17	0.00	2.47	3.98	15.63	21.05
20	3	20	12	0	0.18	0.00	0.98	4.52	56.78	14.28
40	3	20	2	2	0.36	-0.09	1.01	58.76	117.14	2.72
60	3	20	0	0	0.70	0.05	2.15	170.76	281.83	1.94
80	3	20	0	2	0.98	-0.26	2.83	354.51	511.00	1.47
100	2	160	5	42	0.34	-0.67	2.66	136.07	445.09	9.46
	3	160	0	24	0.91	-0.96	5.17	481.80	822.60	4.02

The `RMH` runs much faster than the other heuristics: the speedup ratio is on average 9.4 and 28.0 against Gu *et al.* (2019) and Soleilhac (2022), respectively. Although the good decrease in the solution time, the quality of the solutions provided by `RMH` stays comparable with the one of the competitors. Indeed, when compared with Gu *et al.* (2019), `RMH` matches 71 upper bounds, improves 70 of them of on average 0.26% and the percentage gap of the remaining ones is on average 0.83%. The results show a similar trend in the comparison with Soleilhac (2022): `RMH` matches five upper bounds, improves 36 of them of on average 0.35% and the percentage gap of the remaining ones

Table 3.7: Comparison with Soleilhac (2022) on the instances with $|\mathcal{N}| = 100$ and $|\mathcal{K}| = 2, 3$ and customers' locations from C101 and R101.

Instances			RMH vs. Soleilhac (2022)							
			UB							
					gap[%]			avg.t[s]		
$\|\mathcal{N}\|$	$\|\mathcal{K}\|$	#inst.	#equal	#impr.	avg.	min.	max.	RMH	Soleilhac (2022)	avg.ratio
100	2	160	5	12	0.49	-0.34	2.84	136.07	1800	44.62
	3	160	0	24	0.63	-1.45	4.03	481.80	1800	11.47

is on average 0.69%. Recall that our heuristic provides also a performance guarantee contrary to Gu *et al.* (2019) and Soleilhac (2022).

3.6 Conclusions

In this paper, we considered the Commodity constrained Split Delivery Vehicle Routing Problem (C-SDVRP), a routing problem where customer demands may be composed of multiple commodities and split deliveries are allowed as long as the demand of a single commodity is delivered all in once. We presented a heuristic with a performance guarantee to solve the problem. Our heuristic is based on a column generation approach which embeds a new pricing heuristic that exploits the multi-commodity aspect of the problem. Such contribution allowed to reduce the computational time on instances where the variability of the customer demands is not large. We performed a thorough computational analysis on the 1380 benchmark instances available in the literature. We provide an upper bound for all the considered instances, the majority of those are guaranteed to be of good quality (optimality gap less than 5%). Some new best-known solutions are found. Finally, our approach outperforms the state-of-the-art metaheuristic for the C-SDVRP (Gu *et al.* (2019) and Soleilhac (2022)) in terms of computational time while maintaining the quality of the upper bounds comparable.

Future research may be devoted to the inclusion of additional families of valid inequalities (possibly robust) to improve both the lower and upper bounds. Finally, our approach may be adapted to solve variants of the problem where the additional constraints can be easily handled in the pricing problem (e.g. a multi-compartment C-SDVRP).

Chapter 4

A Branch-Price-and-Cut algorithm for the Multi-Commodity two-echelon Distribution Problem

Contents

The content of this chapter was presented at the following conferences: ROADEF 2022, Odysseus 2021, VeRoLog 2022 and TRISTAN XI 2022. This chapter corresponds to the paper "A Branch-Price-and-Cut algorithm for the Multi-Commodity two-echelon Distribution Problem" submitted to Computers & Industrial Engineering on 17 July 2023.

Abstract: In the Multi-Commodity two-echelon Distribution Problem (MC2DP), multiple commodities are distributed in a two-echelon distribution system involving suppliers, distribution centres and customers. Each supplier may provide different commodities and each customer may request several commodities as well. In the first echelon, capacitated vehicles perform direct trips to transport the commodities from the suppliers to the distribution centres for consolidation purposes. In the second echelon, each distribution centre owns a fleet of capacitated vehicles to deliver the commodities to the customers through multi-stop routes. Commodities are compatible, i.e., they can be mixed in the vehicles. Finally, customer requests can be split by commodities, that is, a customer can be visited by several vehicles, but the total amount of each commodity has to be delivered by a single vehicle. The aim of the MC2DP is to minimise the total transportation cost to satisfy customer demands.

We propose a set covering formulation for the MC2DP where the exponential number of variables relates to the routes in the delivery echelon. We develop a Branch-Price-and-Cut algorithm (BPC) to solve the problem. The pricing problem results in solving an Elementary Shortest Path Problem with Resource Constraints (ESPPRC) per distribution centre. We tackle the ESPPRC with a label setting dynamic programming algorithm which incorporates ng-path relaxation and a bidirectional labelling search. Pricing heuristics are invoked to speed up the procedure. In addition, the formulation is strengthened by integrating capacity cuts and two families of valid inequalities specific for the multiple commodities aspect of the problem.

Our approach solves to optimality 439 over the 736 benchmark instances from the literature. The optimality gap of the unsolved instances is 2.1%, on average.

Keywords: Two echelon routing problems, Multiple commodities, Split delivery, Branch-Price-and-Cut.

4.1 Introduction

In a two-echelon distribution system, goods are transferred from origins (depots, suppliers) to destinations (customers) via intermediate facilities (satellites, distribution centres) (see Guastaroba *et al.*, 2016). In the *collection echelon*, large vehicles bring goods from the origins to the intermediate facilities where consolidation operations are performed. Whereas, in the *delivery echelon*, smaller vehicles are in charge of distributing the goods to the final customers. Routing decisions are usually required in both echelons. Two-echelon systems take advantage of consolidating goods at intermediate facilities and using different fleets within each echelon to reduce overall transportation costs. An example of this delivery strategy can be encountered in city logistics (Cattaruzza *et al.*, 2017; Crainic *et al.*, 2023) where the aim is also to grant the access in urban areas only to environmental-friendly vehicles that usually have a small capacity.

In this article, we consider a two-echelon distribution problem arising in the short and local fresh food supply chains, namely the *Multi-Commodity two-echelon Distribution Problem* (MC2DP) introduced in Gu *et al.* (2022). In this context, origins, intermediate facilities and destinations are referred to as *suppliers*, *distribution centres* and *customers*, respectively. There are few vehicle routing problems which explicitly deal with multiple commodities within a two-echelon distribution system. To the best of our knowledge, among these problems, the MC2DP is the only one considering a *many-to-many* setting. In fact, in the MC2DP, the commodity requested by a customer is not pre-assigned to a specific supplier, so it can be collected at any supplier or subset of suppliers where it is available. The amount of the commodities available at the suppliers is limited. In contrast with the usual setting in the literature, the MC2DP requires routing decisions only in the delivery echelon. Indeed, commodities are collected from the suppliers and brought to the distribution centres via direct round trips. In the delivery echelon, a fleet of vehicles performing routes starting and ending at the same distribution centre is used to deliver the commodities to the customers. All vehicles involved in the distribution system are capacitated and commodities are *compatible*, i.e., they can be mixed inside all vehicles. Finally, as in the *Commodity constrained Split Delivery Vehicle Routing Problem* (C-SDVRP) Archetti *et al.* (2016), customers can be visited by multiple vehicles as long as the demand of a single commodity is served by a single vehicle. The aim of the MC2DP is to determine a distribution plan to satisfy customer demands while respecting the capacity of the vehicles and not exceeding the

commodity availabilities at the suppliers and such that the total transportation cost is minimised.

The authors in Gu *et al.* (2022) proposed a compact Mixed Integer Linear Programming (MILP) formulation and a sequential heuristic for the MC2DP. The authors decompose the MC2DP in two subproblems: one for the collection from suppliers, and the other one for the delivery to customers. The collection subproblem is modeled as a MILP and solved with a commercial solver while the delivery subproblem is solved by an Adaptive Large Neighbourhood Search (ALNS) algorithm.

The contribution of this paper is to present an exended model and to propose the first ever exact approach based on a Branch-Price-and-Cut (BPC) algorithm to solve the MC2DP. Similar exact approaches have recently been proposed to deal with two-echelon vehicle routing problems (see e.g. Li *et al.*, 2022; Marques *et al.*, 2020, 2022; Mhamedi *et al.*, 2022). However, our BPC algorithm is designed to take into account explicitly the multi-commodity dimension. Specifically, our algorithm relies on a set covering formulation for the MC2DP where the exponentially-many number of variables correspond to the routes in the delivery echelon starting and ending at each distribution centre. We also strengthen the formulation by the insertion of capacity cuts, valid inequalities arising from the set covering polytope (Balas & Ng, 1989) and a new family of valid inequalities based on the number partitioning problem polytope. While capacity cuts are classical inequalities derived for the *Capacitated Vehicle Routing Problem* (CVRP) (see Laporte *et al.*, 1985), the other two families of inequalities tackle the multi-commodity aspect of the problem. Finally, several state-of-art speed-up techniques are also incorporated in our BPC algorithm

The remainder of the paper is organized as follows. Section 4.2 provides a literature review. In Section 4.3, a formal description of the MC2DP is provided. In Section 4.4, a set covering formulation is presented along with different families of valid inequalities. Our Branch-Price-and-Cut algorithm is described in Section 4.5. Finally, in Section 4.6 we analyse the results obtained by the proposed algorithm on the benchmark instances introduced in Gu *et al.* (2022) to assess its effectiveness.

4.2 Literature review

In this section, we review the existing literature on the two-echelon distribution problems, with particular attention to the ones dealing with multiple commodities. The first

two-echelon routing problem introduced by Jacobsen & Madsen (1980) was motivated by a specific application. Newspapers have to be distributed from a printing office to sales points possibly passing through some transfer points whose locations are to be decided. Crainic *et al.* (2004) and Crainic *et al.* (2009) proposed a formal description of a rich class of two-echelon routing problems along with some economic insights. However, the seminal problem in this class, namely the *two-echelon Capacitated Vehicle Routing Problem* (2E-CVRP), was introduced in the literature and studied for the first time in Perboli *et al.* (2011). In the 2E-CVRP, a single commodity has to be transferred from a single origin to several destinations through some intermediate facilities. Two fleets of capacitated vehicles perform routes in the two echelons to transport the commodity from the origin to the intermediate facilities and from the intermediate facilities to the destinations. The objective of the 2E-CVRP is to minimise the total transportation cost of the distribution system. The authors proposed two math-heuristics to solve the problem, a diving and a sub-MIP heuristic.

The 2E-CVRP and related problems have received increasing attention in recent years and many variants have been addressed, e.g., 2E-CVRP with (i) time windows (Mhamedi *et al.*, 2022); (ii) mobile satellites (Li *et al.*, 2020); (iii) synchronization (Grangier *et al.*, 2016) and bi-synchronization (Li *et al.*, 2021b); (iv) simultaneous pickup and delivery (Li *et al.*, 2022); (v) electric vehicles (Breunig *et al.*, 2019) and battery swapping stations (Jie *et al.*, 2019); (vi) real-time transshipment capacity varying (Li *et al.*, 2018); (vii) covering options (Enthoven *et al.*, 2020); (viii) delivery options (Zhou *et al.*, 2018); (ix) stochastic demands (Sluijk *et al.*, 2021). The interested reader may refer to Cuda *et al.* (2015); Li *et al.* (2021a) and Sluijk *et al.* (2023) for recent surveys on the subject.

According to the existing literature, the vast majority of the two-echelon routing problems deal with the single commodity case. Apart from the MC2DP, which is addressed in this paper, only a few works integrate multiple commodities in a two-echelon routing problem (e.g. Dellaert *et al.*, 2021; Gu *et al.*, 2022; Jia *et al.*, 2023). In Dellaert *et al.* (2021), the authors extended the 2E-CVRP by introducing multiple origins and multiple commodities. In addition, hard time windows are imposed for the delivery at the destinations. In their problem, customers have a commodity demand from a specific origin, i.e., there is a *one-to-one* setting. Several mathematical formulations are proposed and a BPC algorithm is devised to solve the problem. In Jia *et al.* (2023),

the problem setting is similar to the one of Dellaert *et al.* (2021). However, the multi-commodity aspect is handled with more restrictions: only two origins are considered and each destination requires one commodity per origin (one-to-one setting). The authors developed an ALNS algorithm to solve large-scale instances of the problem. The MC2DP introduced in Gu *et al.* (2022) differs from Dellaert *et al.* (2021) and Jia *et al.* (2023) for three reasons:(i) there is a many-to-many setting for the commodities, i.e. any commodity requested by a customer can be served from any supplier; (ii) suppliers provide commodities in limited amounts; (iii) routing decisions are not required in the collection echelon.

4.3 Problem description

In the *Multi-Commodity two-Echelon Distribution Problem* (MC2DP), a set of commodities $\mathcal{K}$ is distributed in a system involving a set of *suppliers* (origins) $\mathcal{S}$, a set of *distribution centres* (intermediate facilities) $\mathcal{D}$ and a set of *customers* (destinations) $\mathcal{C}$. The system is split in two echelons: the collection echelon where the commodities are collected at the suppliers and brought to the distribution centres, and the delivery echelon where the commodities at the distribution centres are delivered to the customers. More precisely, in the collection echelon, each supplier $i \in \mathcal{S}$ provides a maximal amount $P_{ik} \geq 0$ for each commodity $k \in \mathcal{K}$. Note that a supplier $i \in \mathcal{S}$ might not supply a commodity $k \in \mathcal{K}$, and in that case, P_{ik} takes value 0. An unlimited fleet of homogeneous vehicles of capacity Q^S performs direct round trips from the distribution centres to collect the commodities from the suppliers. The vehicles can transport any subset of commodities. Due to the limited capacity of the vehicles, direct round trips between a distribution centre $o \in \mathcal{D}$ and a supplier $i \in \mathcal{S}$ may be performed by several vehicles. The problem associated with the collection operations can be modeled as a *Multi-commodity Capacitated fixed-charge Network Design Problem* (MCNDP, Magnanti & Wong, 1984) with a specific cost structure: there is a step-wise cost function defined by a unitary cost associated with each vehicle used between a distribution centre and a supplier.

Differently, the problem of distributing the commodities from the distribution centres to the customers is a multi-depot version of the *Commodity constrained Split Delivery Vehicle Routing Problem* (C-SDVRP). Each customer $j \in \mathcal{C}$ has a demand $D_{jk} \geq 0$ for all commodities $k \in \mathcal{K}$. The request of customer j is identified by set

$\mathcal{K}_j = \{k \in \mathcal{K} : D_{jk} > 0\}$. Each distribution centre owns an unlimited fleet of homogeneous and capacitated vehicles of capacity Q^D which performs routes to deliver the commodities to the customers. Each vehicle has to end its route at its starting distribution centre. As in the collection echelon, a vehicle can be loaded with any commodities. Without loss of generality, we suppose $Q^D \geq \max\{\sum_{k\in\mathcal{K}_j} D_{jk} : j \in \mathcal{C}\}$. Furthermore, customer requests can be split, i.e., different vehicles can serve the same customer. However, the demand of a single commodity cannot be split: it has to be delivered by a single vehicle. Note that direct trips from suppliers to customers and inter-connections between distribution centres are not allowed.

Finally, the collection and delivery operations taking place in the two echelons are coordinated at the distribution centres by means of the so-called *load synchronization* strategy Drexl (2012): the total amount of each commodity collected at the suppliers by each distribution centre must be sufficient to serve the customer demands of that commodity delivered by a vehicle of that distribution centre.

We formulate the MC2DP on a directed weighed graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. Set $\mathcal{V} = \mathcal{S} \cup \mathcal{D} \cup \mathcal{C}$ contains a vertex for each supplier, distribution centre and customer. Arc set $\mathcal{A} = \mathcal{A}_S \cup \mathcal{A}_D$ is defined as the union of two sets of arcs which model the possible vehicle travels in the two echelons. Specifically, set $\mathcal{A}_S = (\mathcal{S} \times \mathcal{D}) \cup (\mathcal{D} \times \mathcal{S})$ includes the arcs modelling the direct trips from suppliers to distribution centres in the collection echelon, whereas $\mathcal{A}_D = (\mathcal{D} \times \mathcal{C}) \cup (\mathcal{C} \times \mathcal{D}) \cup (\mathcal{C} \times \mathcal{C})$ contains all arcs between customers and between distribution centres and customers. Each arc $(i, j) \in \mathcal{A}$ is assigned with a non-negative cost C_{ij} which represent the transportation cost of a vehicle traversing (i, j). The arc costs are symmetric and satisfy the triangular inequality. In graph $\mathcal{G}$, a route in the delivery echelon is a non-empty circuit starting and ending at a distribution centre $o \in \mathcal{D}$. A route is *feasible* if the total amount of the delivered commodities to the customers visited along the route does not exceed vehicle capacity Q^D. The cost of any feasible route r is $C_r = \sum_{(i,j)\in\mathcal{A}(r)} C_{ij}$, where $\mathcal{A}(r)$ is the set of arcs traversed by the route. Finally, the total transportation cost of the distribution system arising from the MC2DP is the sum of the cost of the direct round trips in the collection echelon and the routing costs in the delivery echelon.

The aim of the MC2DP is to determine a distribution plan, i.e., the direct round trips in the collection echelon and the routes in the delivery echelon, which satisfies the customer requests, does not exceed the commodity availabilities at the suppliers,

satisfies the vehicle capacities in both echelons and respects the load synchronization constraints while minimising the total transportation cost.

4.4 Problem formulation

We model the MC2DP by means of a set covering formulation, where the exponentially-many variables are associated with the routes in the delivery echelon.

For each distribution centre $o \in \mathcal{D}$, we define $\mathcal{R}_o$ as the set of all feasible routes starting and ending at o. The set of all feasible routes is denoted by $\mathcal{R} = \bigcup_{o \in \mathcal{D}} \mathcal{R}_o$. We define a binary coefficient a^r_{jk} with value one if commodity $k \in \mathcal{K}$ is delivered to customer $j \in \mathcal{N}$ by route $r \in \mathcal{R}$ and zero otherwise.

For each supplier $i \in \mathcal{S}$ and each distribution centre $o \in \mathcal{D}$, we introduce an integer variable x_{io} to represent the number of vehicles traversing arc $(i, o) \in \mathcal{A}_{\mathcal{S}}$. For each $i \in \mathcal{S}$, $o \in \mathcal{D}$ and $k \in \mathcal{K}$, we define a non-negative continuous variable q^k_{io} that represents the amount of commodity k collected at supplier i by distribution centre o. Finally, for each route $r \in \mathcal{R}$, we introduce a binary variable λ_r taking value one if r is selected in the solution and zero otherwise.

The Set Covering formulation [SC] for the MC2DP reads as follows:

$$[\text{SC}] \quad \min \sum_{(i,o) \in \mathcal{A}_{\mathcal{S}}} 2C_{io}x_{io} + \sum_{r \in \mathcal{R}} C_r \lambda_r \tag{4.1}$$

$$\text{s.t.} \sum_{o \in \mathcal{D}} q^k_{io} \leq P_{ik} \qquad \forall i \in \mathcal{S}, \forall k \in \mathcal{K} \tag{4.2}$$

$$\sum_{k \in \mathcal{K}} q^k_{io} \leq Q^S x_{io} \qquad \forall i \in \mathcal{S}, \forall o \in \mathcal{D} \tag{4.3}$$

$$\sum_{r \in \mathcal{R}} a^r_{jk} \lambda_r \geq 1 \qquad \forall j \in \mathcal{C}, \forall k \in \mathcal{K}_j \tag{4.4}$$

$$\sum_{i \in \mathcal{S}} q^k_{io} \geq \sum_{r \in \mathcal{R}_o} \sum_{j \in \mathcal{C}} a^r_{jk} D_{jk} \lambda_r \qquad \forall o \in \mathcal{D}, \forall k \in \mathcal{K} \tag{4.5}$$

$$x_{io} \in \mathbb{Z}_{\geq 0} \qquad \forall i \in \mathcal{S}, \forall o \in \mathcal{D} \tag{4.6}$$

$$q^k_{io} \in \mathbb{R}_{\geq 0} \qquad \forall i \in \mathcal{S}, \forall o \in \mathcal{D}, \forall k \in \mathcal{K} \tag{4.7}$$

$$\lambda_r \in \{0, 1\} \qquad \forall r \in \mathcal{R} \tag{4.8}$$

Objective function (4.1) minimises the total transportation cost. Constraints (4.2) ensure that the commodity availabilities at each supplier are respected. Constraints (4.3)

guarantee that a sufficient number of vehicles perform the collection operations and that the capacity of these vehicles is not exceeded. Covering Constraints (4.4) impose that each commodity required by a customer is served by at least one route. In addition, the load synchronization constraint linking the collection and delivery echelons is expressed in constraints (4.5): the quantity of each commodity collected by each distribution centre has to be large enough to satisfy the demand for that commodity delivered by a route of that distribution centre. Finally, Constraints (4.6), (4.7) and (4.8) define variable domains.

4.4.1 Valid inequalities

In this section, we introduce four families of valid inequalities considered to strengthen formulation [SC]. Two of these inequalities are known in the context of vehicle routing problems, while the other two are tailored to deal with the multi-commodity aspect of the MC2DP. Note that such inequalities are valid for the C-SDVRP, hence for the MC2DP.

In what follows, given a subset of customers $\mathcal{C}' \subseteq \mathcal{C}$, we define $D(\mathcal{C}') = \sum_{j \in \mathcal{C}'} \sum_{k \in \mathcal{K}_j} D_{jk}$ to be the total demand requested by the customers in $\mathcal{C}'$. In addition, we introduce a binary coefficient b^r_{ij} with value one if route $r \in \mathcal{R}$ traverses arc $(i,j) \in \mathcal{A}_\mathcal{D}$ and zero otherwise. Finally, we define $e^r_{j\mathcal{M}} = \prod_{k \in \mathcal{M}} a^r_{jk}$ to be a binary coefficient equal to one if route r delivers all the commodities of subset $\mathcal{M} \subseteq \mathcal{K}_j$ to customer $j \in \mathcal{C}$ and zero otherwise.

Bounds on the number of vehicles

The following inequalities set bounds on the number of vehicles in the collection and delivery echelons (see Marques *et al.*, 2020):

$$\sum_{(i,o) \in \mathcal{A}_\mathcal{S}} x_{io} \geq \left\lceil \frac{D(\mathcal{C})}{Q^S} \right\rceil \tag{4.9}$$

and

$$\sum_{r \in \mathcal{R}} \lambda_r \geq \lceil \underline{v} \rceil \tag{4.10a}$$

$$\sum_{r \in \mathcal{R}} \lambda_r \leq \min\{|\mathcal{C}|, 2\bar{v}\}. \tag{4.10b}$$

In inequalities (4.10a) and (4.10b), values $\underline{v}$ and $\bar{v}$ are obtained by solving an instance of the *Bin Packing Problem* (BPP), where bins have size equal to the vehicle capacity Q^D, and each customer demand has a corresponding item to be packed with size D_{jk}. Precisely, we solve an integer program for the BPP on such an instance with a commercial solver within a short time limit: $\underline{v}$ and $\bar{v}$ are the obtained lower and upper bounds. If the instance is solved to optimality within the time limit, $\underline{v} = \bar{v}$ holds. The right hand-side of (4.10b) is the minimum between twice value $\bar{v}$ (see Federgruen & Simchi-Levi, 1995) and the number of customers.

Capacity cuts

Laporte *et al.* (1985) introduced the capacity cuts to deal with the *Capacitated Vehicle Routing Problem*:

$$\sum_{r \in \mathcal{R}} \left(\sum_{(i,j) \in \delta^-(\mathcal{C}')} b_{ij}^r \right) \lambda_r \geq \left\lceil \frac{D(\mathcal{C}')}{Q^D} \right\rceil \qquad \forall \mathcal{C}' \subseteq \mathcal{C}, \tag{4.11}$$

where $\delta^-(\mathcal{C}') = \{(i,j) \in \mathcal{A}_\mathcal{D} : i \notin \mathcal{C}', j \in \mathcal{C}'\}$ is the set of arcs of graph $\mathcal{G}$ reaching a vertex in $\mathcal{C}'$. Given a subset of customers $\mathcal{C}'$, inequality (4.11) states that at least $\lceil D(\mathcal{C}')/Q^D \rceil$ vehicles of the delivery echelon are required to cover the requests of the customers in $\mathcal{C}'$.

Set covering polytope

We present a family of valid inequalities inspired by the facet-defining inequalities proposed in Balas & Ng (1989) for the set covering polytope. Although these inequalities were proposed several years ago, to the best of our knowledge, they have not been used in BPC algorithms before. We adapted them to deal with the multi-commodity aspect of the problem.

Let us first briefly present a formulation for the set covering problem. Let $\mathcal{I}$ bet a set of elements to be covered, and $\mathcal{J}$ be a set of subsets of $\mathcal{I}$. We denote by c_j the cost associated to subset $j \in \mathcal{J}$, and d_{ij} a binary parameter that takes value one if element $i \in \mathcal{I}$ is in subset $j \in \mathcal{J}$, and zero otherwise. Let x_j be a binary decision variable taking value one if subset $j \in \mathcal{J}$ is selected, zero otherwise. An integer programming

formulation for the set covering problem is

$$\begin{aligned} \min & \sum_{j \in \mathcal{J}} c_j x_j & \\ \text{s.t.} & \sum_{j \in \mathcal{J}} d_{ij} x_j \geq 1 & \forall i \in \mathcal{I} \\ & x_j \in \{0,1\} & \forall j \in \mathcal{J} \end{aligned}$$

Given a subset $\mathcal{I}' \subseteq \mathcal{I}$, the inequalities introduced in Balas & Ng (1989) reads as follows:

$$2 \sum_{j \in \mathcal{J}^{\mathcal{I}'}} x_j + \sum_{j \in \bar{\mathcal{J}}^{\mathcal{I}'}} x_j \geq 2,$$

where $\mathcal{J}^{\mathcal{I}'} = \{j \in \mathcal{J} : d_{ij} = 1, \forall i \in \mathcal{I}'\}$ is the set of the elements of $\mathcal{J}$ which cover $\mathcal{I}'$ and $\bar{\mathcal{J}}^{\mathcal{I}'} = \{j \in \mathcal{J} : \sum_{i \in \mathcal{I}'} d_{ij} \geq 1 \wedge \prod_{i \in \mathcal{I}'} d_{ij} = 0\}$ is the set of the elements of $\mathcal{J}$ which contain some, but not all, the elements in $\mathcal{I}'$. The inequalities express how subset $\mathcal{I}'$ may be covered: either it suffices to select a unique element in $\mathcal{J}$ that covers $\mathcal{I}'$, i.e., an element in $\mathcal{J}^{\mathcal{I}'}$, or at least two elements in $\mathcal{J}$ that partially cover $\mathcal{I}'$ have to be selected, i.e., at least two elements in $\bar{\mathcal{J}}^{\mathcal{I}'}$. Under specific conditions, these constraints are facet defining for the set covering polytope.

In what follows, we adapt these inequalities to the MC2DP to express how the subsets of commodities required by a given customer may be covered. For the ease of readability, we introduce the following notation. Let $j \in \mathcal{C}$ be a customer and $\mathcal{M}_j \subseteq \mathcal{K}_j$ be a subset of the commodities requested by j. We denote by $\mathcal{R}_j^{\mathcal{M}_j} \subseteq \mathcal{R}$ the subset of routes delivering all commodities in $\mathcal{M}_j$ to j, i.e., $\mathcal{R}_j^{\mathcal{M}_j} = \{r \in \mathcal{R} : e^r_{j\mathcal{M}_j} = 1\}$.

In addition, we write $\bar{\mathcal{R}}_j^{\mathcal{M}_j} \subseteq \mathcal{R}$ for the subset of routes which deliver some of the commodities in $\mathcal{M}_j$ to j, but not all of them, i.e., $\bar{\mathcal{R}}_j^{\mathcal{M}_j} = \{r \in \mathcal{R} : \sum_{k \in \mathcal{M}_j} a^r_{jk} \geq 1 \wedge e^r_{j\mathcal{M}_j} = 0\}$.

The set covering polytope inequalities for the MC2DP are defined as follows:

$$2 \sum_{r \in \mathcal{R}_j^{\mathcal{M}_j}} \lambda_r + \sum_{r \in \bar{\mathcal{R}}_j^{\mathcal{M}_j}} \lambda_r \geq 2 \qquad \forall j \in \mathcal{C}, \forall \mathcal{M}_j \subseteq \mathcal{K}_j. \tag{4.12}$$

Inequalities (4.12) state that subset of commodities $\mathcal{M}_j \subseteq \mathcal{K}_j$ of customer $j \in \mathcal{C}$ can be covered either by a single route in $\mathcal{R}_j^{\mathcal{M}_j}$ or by at least two routes in $\bar{\mathcal{R}}_j^{\mathcal{M}_j}$. Note that these inequalities are meaningful only if $|\mathcal{M}_j| \geq 3$. Indeed, if $|\mathcal{M}_j| = 2$, they can be retrieved as an aggregation of covering constraints (4.4).

Number partitioning polytope

We propose a novel family of valid inequalities which exploits the multi-commodity aspect of the MC2DP. More precisely, given a customer $j \in \mathcal{C}$, these inequalities specify the possible combinations of routes to deliver the set of commodities $\mathcal{K}_j$ required by customer j.

For each customer $j \in \mathcal{C}$, we denote by $\mathcal{R}_j^l$ the subset of routes which deliver exactly $l = 1, \ldots, |\mathcal{K}_j|$ commodities to j, i.e., $\mathcal{R}_j^l = \{r \in \mathcal{R} : \sum_{k \in \mathcal{K}_j} a_{jk}^r = l\}$.

Equalities

$$\sum_{l=1}^{|\mathcal{K}_j|} l \sum_{r \in \mathcal{R}_j^l} \lambda_r = |\mathcal{K}_j| \qquad \forall j \in \mathcal{C} \tag{4.13}$$

ensure that the selected routes that serve customer j will exactly bring $|\mathcal{K}_j|$ commodities to customer j. As an example, let $\bar{j} \in \mathcal{C}$ be a customer having a demand for three commodities, i.e., $|\mathcal{K}_{\bar{j}}| = 3$. Equality (4.13) for customer $\bar{j}$ states that the commodities of $\mathcal{K}_{\bar{j}}$ can be covered by (i) a single route of $\mathcal{R}_{\bar{j}}^3$ or (ii) one route of $\mathcal{R}_{\bar{j}}^2$ and a route of $\mathcal{R}_{\bar{j}}^1$ or (iii) three routes of $\mathcal{R}_{\bar{j}}^1$.

Proposition 4.1. *Inequalities* $\sum_{l=1}^{|\mathcal{K}_j|} l \sum_{r \in \mathcal{R}_j^l} \lambda_r \geq |\mathcal{K}_j|$, $\forall j \in \mathcal{C}$, *are implied by set Covering Constraints* (4.4) *and inequalities*

$$\sum_{l=1}^{|\mathcal{K}_j|} l \sum_{r \in \mathcal{R}_j^l} \lambda_r \leq |\mathcal{K}_j| \qquad \forall j \in \mathcal{C} \tag{4.14}$$

are valid for the MC2DP.

Proof. It is straightforward that equalities (4.13) are valid for the MC2DP. Hence, we only need to show the first statement of the proposition. Let $j \in \mathcal{C}$ be a customer. By summing up the Covering Constraints (4.4) associated with j and swapping the summation order, we obtain

$$\sum_{r \in \mathcal{R}} \sum_{k \in \mathcal{K}_j} a_{jk}^r \lambda_r \geq |\mathcal{K}_j|.$$

Let $\mathcal{M}_j^r$ denote the subset of commodities delivered to customer j by route r. We have $\sum_{k \in \mathcal{K}_j} a_{jk}^r = |\mathcal{M}_j^r|$. The proof follows from partitioning the set of routes as $\mathcal{R} = \bigcup_{l=0}^{|\mathcal{K}_j|} \mathcal{R}_j^l$,

where we denoted by $\mathcal{R}_j^0$ the subset of routes which do not visit j. Indeed, it holds

$$|\mathcal{K}_j| \leq \sum_{l=0}^{|\mathcal{K}_j|} \sum_{r \in \mathcal{R}_j^l} |\mathcal{M}_j^r| \lambda_r = \sum_{l=1}^{|\mathcal{K}_j|} l \sum_{r \in \mathcal{R}_j^l} \lambda_r.$$

□

Remark that if we model the MC2DP by means of a set partitioning formulation, i.e., we impose the equality in Constraints (4.4), Equalities (4.13) become trivial. Indeed, they can be retrieved as an aggregation of the partitioning constraints.

Given a customer $j \in \mathcal{C}$ and $l = 1, \ldots, |\mathcal{K}_j|$, we introduce an auxiliary variable $y_j^l \in \mathbb{Z}_{\geq 0}$ defined as $y_j^l \coloneqq \sum_{r \in \mathcal{R}_j^l} \lambda_r$. Now, let

$$\mathcal{F}_j \coloneqq \{y_j \in \mathbb{Z}_{\geq 0}^{|\mathcal{K}_j|} : \sum_{l=1}^{|\mathcal{K}_j|} l y_j^l \leq |\mathcal{K}_j|\}$$

be the set of the integer points which satisfy inequality (4.14), rewritten in terms of y_j^l variables.

Proposition 4.2. *The inequalities defining the convex hull of $\mathcal{F}_j$, $j \in \mathcal{C}$, are valid for the MC2DP.*

Determining the external description of a convex set is not an easy task, in particular in large dimensions. In the following, we propose a general procedure to determine the external description of the convex hull of sets $\mathcal{F}_j$ for customers $j \in \mathcal{C}$ with an arbitrary (and reasonable) number of commodities. However, given that customers require at most three commodities in the benchmark instances of Gu *et al.* (2022) for the MC2DP, we explicitly derive the external description of the convex hull of sets $\mathcal{F}_j \subseteq \mathbb{Z}^3$, $j \in \mathcal{C}$. The procedure reads as follows:

1. determine the extreme vertices of the convex hull of $\mathcal{F}_j$; by definition of $\mathcal{F}_j$, this operation reduces to determine the integer partitions of number $|\mathcal{K}_j|$ which are pair-wise linear independent;

2. enumerate all the half-spaces containing all such extreme vertices, and originating from hyperplanes generated by the extreme vertices of $\mathcal{F}_j$.

Note that inequalities (4.14) are meaningful only for customers $j \in \mathcal{C}$ who require at least three commodities, i.e., $|\mathcal{K}_j| \geq 3$. The external description of the convex hull of sets $\mathcal{F}_j$, $j \in \mathcal{C}$ such that $|\mathcal{K}_j| = 3$ reads as follows:

$$\begin{cases} y_j^1 + 2y_j^2 + 3y_j^3 \leq 3 & \text{(4.15a)} \\ y_j^1 - y_j^2 \geq 0 & \text{(4.15b)} \\ y_j^2 \geq 0 & \text{(4.15c)} \\ y_j^3 \geq 0. & \text{(4.15d)} \end{cases}$$

Inequalities (4.15c) and (4.15d) are trivial, indeed, they are implied by the definition of variables y_j^l. Therefore, inequalities (4.15a) and (4.15b) are the only meaningful ones in the case of a customer $j \in \mathcal{C}$ requiring three commodities ($|\mathcal{K}_j| = 3$); in terms of λ variables, they are expressed respectively as

$$\sum_{l=1}^{|\mathcal{K}_j|} l \sum_{r \in \mathcal{R}_j^l} \lambda_r \leq |\mathcal{K}_j| \qquad \forall j \in \mathcal{C} : |\mathcal{K}_j| = 3 \qquad \text{(4.16)}$$

$$\sum_{r \in \mathcal{R}_j^1} \lambda_r - \sum_{r \in \mathcal{R}_j^2} \lambda_r \geq 0 \qquad \forall j \in \mathcal{C} : |\mathcal{K}_j| = 3. \qquad \text{(4.17)}$$

In conclusion, the number partitioning polytope valid inequalities we consider are (4.16) and (4.17).

4.5 Branch-Price-and-Cut algorithm

We solve formulation [SC] by means of a Branch-Price-and-Cut (BPC) algorithm (Barnhart *et al.*, 1998), i.e., a variant of the branch-and-bound algorithm which deals with integer programming model with exponentially-many variables. Specifically, at each node of the branch-and-bound tree, the Master Problem (MP), that is the linear relaxation of formulation [SC], is solved by a column generation procedure (Desrosiers & Lübbecke, 2005). If the solution of the MP is fractional, violated valid inequalities of Section 4.4.1 may be inserted and the column generation procedure is repeated while some valid inequalities are violated. Finally, branching rules are applied to ensure the integrality of the solution. We impose a time limit as a termination criterion for our BPC algorithm.

In this section, we describe the main components of our BPC algorithm. Specifically, in Section 4.5.1 we present the column generation scheme applied in our BPC algorithm. In Section 4.5.2, we detail the management of the valid inequalities, and their impact on the pricing problem. Branching strategies and accelerating techniques are presented in Sections 4.5.3 and 4.5.4, respectively.

4.5.1 Column generation

At each node of the branch-and-bound tree, a column generation procedure solves the MP defined on the exponentially-many variables λ_r, $r \in \mathcal{R}$, which correspond to the routes in the delivery echelon. The starting point is the Restricted Master Problem (RMP). The column generation procedure iteratively solves a Restricted Master Problem (RMP), i.e., the MP restricted to a subset of variables λ_r. At each iteration of the procedure, after the RMP is solved, a subproblem, named *pricing problem* is solved. The aim of the pricing problem is to identify a variable (column) with the smallest reduced cost. If such a column has a negative reduced cost, it is added to the RMP in order to decrease (in a minimization problem) the current value of the solution, and the column generation procedure iterates. The procedure ends when the solution of the pricing problem is a non negative reduced cost column, proving the optimality of the MP.

More precisely, the pricing problem is

$$[\text{PP}] \ \min\{\bar{C}_r : r \in \mathcal{R}\}$$

where $\bar{C}_r$ denotes the reduced cost of λ_r variable. Note that set of routes $\mathcal{R}$ can be partitioned per distribution centre, i.e., $\mathcal{R} = \bigcup_{o \in \mathcal{D}} \mathcal{R}_o$ where $\mathcal{R}_o$ is the set of routes starting and ending at o. Hence, solving [PP] can be done by solving sequentially $|\mathcal{D}|$ independent problems with the same structure:

$$[\text{PP}(o)] \ \min\{\bar{C}_r : r \in \mathcal{R}_o\}, \quad o \in \mathcal{D}.$$

Specifically, the aim of problem [PP(o)] is to determine the most negative reduced cost λ_r, $r \in \mathcal{R}_o$, or to detect that none of them exists. The column generation procedure terminates once all problems [PP(o)], $o \in \mathcal{D}$ do not yield any negative reduced cost variable.

In the following, we detail how a problem [PP(o)] for $o \in \mathcal{D}$ is formulated and solved. By denoting $\rho_{jk} \geq 0,\ \forall j \in \mathcal{C}, k \in \mathcal{K}_j$ and $\sigma_{ok} \geq 0,\ \forall o \in \mathcal{D}, k \in \mathcal{K}$ as the optimal dual prices associated with Constraints (4.4) and (4.5), respectively, the reduced cost of a λ_r, $r \in \mathcal{R}_o$ variable is defined as follows:

$$\bar{C}_r = C_r - \sum_{j \in \mathcal{C}} \sum_{k \in \mathcal{K}_j} a^r_{jk}(\rho_{jk} - D_{jk}\sigma_{ok}). \tag{4.18}$$

As mentioned in Section 4.3, the delivery echelon is a multi-depot version of the C-SDVRP. Hence, the problem [PP(o)] is the pricing problem arising in Branch-Price-and-Cut approaches for the C-SDVRP (see Archetti *et al.*, 2015; Gschwind *et al.*, 2019) and is formulated as an *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC) on a graph $\mathcal{G}(o) = (\mathcal{V}(o), \mathcal{A}(o))$. Such graph is analogous to the one presented in Gschwind *et al.* (2019) to formulate the ESPPRC in the context of the C-SDVRP. Vertex set $\mathcal{V}(o)$ contains two copies o' and o'' of distribution centre o and two copies j' and j'' of each customer $j \in \mathcal{C}$. Each arc of set $\mathcal{A}(o)$ is associated with two resources: demand $\bar{D}$ and cost $\bar{C}$. Arc set $\mathcal{A}(o)$ contains:

1. an arc (i'', j') for each arc $(i, j) \in \mathcal{A}$ to model the movement of a vehicle from vertex i to vertex j; the demand and cost are set to $\bar{D}_{i''j'} \coloneqq 0$ and $\bar{C}_{i''j'} \coloneqq C_{ij}$, respectively.

2. an arc $(j', j'')^{\mathcal{M}_j}$ for each customer $j \in \mathcal{C}$ and each subset $\mathcal{M}_j \subseteq \mathcal{K}_j$ to model the delivery of the commodities of $\mathcal{M}_j$ to j; the demand and cost are set to $\bar{D}^{\mathcal{M}_j}_{j'j''} \coloneqq \sum_{k \in \mathcal{M}_j} D_{jk}$ and $\bar{C}^{\mathcal{M}_j}_{j'j''} \coloneqq -\sum_{k \in \mathcal{M}_j}(\rho_{jk} - D_{jk}\sigma_{ok})$, respectively.

Solving problem [PP(o)] results in searching for negative reduced cost elementary paths in $\mathcal{G}(o)$ from o'' to o' such that the resource consumption (demand) does not exceed the vehicle capacity Q^D.

To do so, we adopt a two phase procedure:

Phase 1 computes the Pareto-optimal (demand, cost) pairs $(\bar{D}^{\mathcal{M}_j}_{j'j''}, \bar{C}^{\mathcal{M}_j}_{j'j''})$ for each customer $j \in \mathcal{C}$.

Phase 2 solves the ESPPRC on graph $\mathcal{G}(o)$ which includes all arcs of type (i'', j'), and only the Pareto-optimal arcs of type $(j', j'')^{\mathcal{M}_j}$ that have been computed in phase 1. Precisely, the ESPPRC is solved by means of a label setting dynamic programming technique Feillet *et al.* (2004) which works with an implicit version

of the bidirectional labelling search (see Bode & Irnich, 2012; Righini & Salani, 2006). The elementarity constraints are the bottleneck of such procedure, hence, we partially relax it by solving the *ng-path* relaxation (Baldacci *et al.*, 2011) of the ESPPRC. For each customer $j \in \mathcal{C}$, we consider a fixed size *ng-neighbourhood* which includes the 10 closest customers to j and j itself. Remark that such relaxation allows a route to serve the same commodity to the same customer multiple times. Hence, the coefficients of the constraints and valid inequalities need to be update accordingly: e.g., in the covering constraints 4.4, a^r_{jk} becomes an integer coefficient expressing the number of times customer $j \in \mathcal{C}$ is delivered with commodity $k \in \mathcal{K}_j$ by route $r \in \mathcal{R}$.

The reader may refer to Archetti *et al.* (2015) and Gschwind *et al.* (2019) for further details. The resolution of the ESPPRCs is the bottleneck of our algorithm, hence, we heuristically solve the ESPPRC with the objective of quickly finding a negative reduced cost column. When all the heuristics fail to identify such a column, we solve the ESPPRC exactly.

Among others, we apply the two-phase heuristic proposed in (Petris *et al.*, 2023) which exploits the multi-commodity aspect of our problem.

4.5.2 Management of the valid inequalities

In this section, we first describe how the valid inequalities presented in Section 4.4.1 are considered in the pricing problem. Then, we present the cutting strategy adopted in our BPC algorithm.

Impact of the valid inequalities on the pricing problem

First, note that inequality (4.9) imposes a lower bound on the number of vehicles used in the collection echelon. Therefore, it has no impact on the pricing problem. The other inequalities presented in Section 4.4.1 are all robust, i.e. they do not change the structure of the pricing problem, and their associated dual prices have to be integrated into the objective function of pricing problems [PP(o)], $o \in \mathcal{D}$, i.e. on the cost of arcs in graph $\mathcal{G}(o)$.

The arc costs in graph $\mathcal{G}(o)$ are modified in the following way:

Inequalities (4.10a) and (4.10b). Let $\tau^+ \geq 0$ and $\tau^- \leq 0$ be the optimal dual prices associated with valid inequalities (4.10a) and (4.10b) respectively. The value

$\tau^+/2 + \tau^-/2$ is subtracted from the cost of arcs of type (i'', j'), if vertices i or j represent distribution centre o.

Inequalities (4.11). Let $\xi_{\mathcal{C}'} \geq 0$ be the optimal dual prices associated with the capacity cut (4.11) defined over the subset of customers $\mathcal{C}' \subseteq \mathcal{C}$. Let $\delta^-(\mathcal{C}')$ be the subset of arcs in graph $\mathcal{G}$ entering in vertices of $\mathcal{C}'$. The value $\xi_{\mathcal{C}'}$ is subtracted from the cost of arcs (i'', j'), for all $(i, j) \in \delta^-(\mathcal{C}')$.

Inequalities (4.12). Let $\gamma_{j\mathcal{M}_j} \geq 0$ be the optimal dual prices associated with the inequality (4.12) identified by customer $j \in \mathcal{C}$ and commodity subset $\mathcal{M}_j \subseteq \mathcal{K}_j$. The value $2\gamma_{j\mathcal{M}_j}$ is subtracted from the cost of arcs $(j', j'')^{\mathcal{M}'_j}$, for all $\mathcal{M}'_j \subseteq \mathcal{K}_j$ that contain at least all the commodities of $\mathcal{M}_j$, i.e., $\mathcal{M}_j \subseteq \mathcal{M}'_j$. The value $\gamma_{j\mathcal{M}_j}$ is subtracted from the cost of arcs $(j', j'')^{\mathcal{M}'_j}$ for all $\mathcal{M}'_j$ that contain some, but not all, commodities of $\mathcal{M}_j$, i.e. $\mathcal{M}'_j \cap \mathcal{M}_j \neq \emptyset$ and $\mathcal{M}'_j \cap \mathcal{M}_j \neq \mathcal{M}_j$.

Inequalities (4.16) and (4.17). Let $j \in \mathcal{C}$ be a customer requiring exactly three commodities ($|\mathcal{K}_j| = 3$) and let $\psi \leq 0$ and $\chi \geq 0$ be the optimal dual prices associated with inequalities (4.16) and (4.17) defined on j. For all $\mathcal{M}_j \subseteq \mathcal{K}_j$, the cost of arc $(j', j'')^{\mathcal{M}_j}$ is modified as follows: value $|\mathcal{M}_j|\psi$ is subtracted, value χ is added if $|\mathcal{M}_j| = 2$, and value χ is subtracted if $|\mathcal{M}_j| = 1$.

Management of the valid inequalities in the RMP

Valid inequalities on vehicle bounds, namely (4.9), (4.10a) and (4.10b), are included in the formulation from the beginning of the solution procedure. Differently, a cut generation procedure manages the insertion of violated inequalities (4.11), (4.12), (4.16) and (4.17) in the RMP. Such a procedure is called at each node of the branch-and-bound tree of level at most equal to 5, if the associated solution of the RMP is fractional. Specifically, it separates the inequalities hierarchically according to the sequence : (4.11), (4.12), (4.16), and (4.17). When the separation of a given inequality fails, we separate the next one in the above order. The separation of inequalities (4.11) is done using the heuristic algorithms presented in Ralphs *et al.* (2003), namely the *extended shrinking heuristic* and the *greedy shrinking heuristic*. Then, although, inequalities (4.12) are exponentially-many, the size of the problem instances allows the separation by enumeration. The same separation strategy is applied for inequalities (4.16) and (4.17), whose number is linear in the number of customers $|\mathcal{C}|$. Finally, we limit

the number of inequalities (4.11) to 100 in each cut generation round. For the other inequalities, we include all the violated inequalities.

4.5.3 Branching strategies

Let $(\bar{x}, \bar{q}, \bar{\lambda})$ be a fractional optimal solution of the MP at a certain node of the branch-and-bound tree. We consider eight branching rules that are hierarchically applied. Rules 1 and 3 are specific for the MC2DP, while the other ones are used to solve the C-SDVRP by Branch-and-Price. The interested reader can refer to Gschwind *et al.* (2019) for more details about these latest branching rules.

Rule 1 is on the number of vehicles traversing an arc in the collection echelon, i.e., on value $\bar{x}_{io}$, $i \in \mathcal{S}$, $o \in \mathcal{D}$. Since λ_r variables are not concerned by this rule, there is no impact on the pricing problem.

Rule 2 is on the number of vehicles used at each distribution centre $o \in \mathcal{D}$ in the delivery echelon, i.e., on value $\sum_{r \in \mathcal{R}_o} \bar{\lambda}_r$.

Rule 3 forces the assignment of a delivery to a distribution centre. Specifically, given a distribution centre $o \in \mathcal{D}$, a customer $j \in \mathcal{C}$ and a commodity $k \in \mathcal{K}_j$, we branch on value $p^o_{jk} := \sum_{r \in \mathcal{R}_o} a^r_{jk} \bar{\lambda}_r$. The branching decisions related to this rule can be expressed as follows: commodity k required by customer j is either delivered from distribution centre o, i.e. $\sum_{o' \in \mathcal{D} \setminus \{o\}} \sum_{r \in \mathcal{R}_{o'}} a^r_{jk} \lambda_r = 0$; or not delivered from o, i.e. $\sum_{r \in \mathcal{R}_o} a^r_{jk} \lambda_r = 0$. Note that both decisions entail modifications in the pricing problem. As an example, if the first decision is imposed, then we prevent the pricing problem from generating routes starting and ending at distribution centres $o' \in \mathcal{D} \setminus \{o\}$ and delivering commodity k to customer j. Arcs of type $(j', j'')^{\mathcal{M}_j}$, with $\mathcal{M}_j \subseteq \mathcal{K}_j$ and $k \in \mathcal{M}_j$, are removed from all graphs $\mathcal{G}(o')$, $o' \in \mathcal{D} \setminus \{o\}$.

Rule 4 is on the number of visits to each customer $j \in \mathcal{C}$ from a distribution centre $o \in \mathcal{D}$, i.e., on value $w^o_j := \sum_{r \in \mathcal{R}_o} d^r_j \bar{\lambda}_r$, where d^r_j is a binary parameter with value one if route r visits customer j and zero otherwise. The branching decisions arising from this rule may enforce modifications in the pricing problem. As an example, if customer $j \in \mathcal{C}$ cannot be visited from distribution centre $o \in \mathcal{D}$, then we eliminate vertices j' and j'' from graph $\mathcal{G}(o)$.

Rule 5 considers the flow on the edges in the delivery echelon coming from a specific distribution centre. Specifically, given a distribution centre $o \in \mathcal{D}$ and an arc $(i,j) \in \mathcal{A}^2$, we branch on value $z^o_{ij} := \sum_{r \in \mathcal{R}_o} (b^r_{ij} + b^r_{ji})\bar{\lambda}_r$. If a zero-flow is imposed on edge (i,j) from distribution centre o, then the arcs in $\mathcal{G}(o)$ associated to (i,j) and (j,i) are removed.

Rules 6 and 7 implement the Ryan and Foster branching rules (Ryan & Foster, 1981). More precisely, given two customers $i, j \in \mathcal{C}$ and two commodities $h \in \mathcal{K}_i$ and $k \in \mathcal{K}_j$, we branch on value $f^{jk}_{ih} := \sum_{r \in \mathcal{R}} a^r_{ih} a^r_{jk} \lambda_r$. The branching decisions related to these rules, namely the *separate* and *together* branches, force the two customer requests to be served by different routes or by the same route, respectively. Rule 6 concerns the case where i and j represent the same customer, i.e., $i = j$. Here, in the separate branch we remove arcs $(j', j'')^{\mathcal{M}}$, $\mathcal{M} \subseteq \mathcal{K}_j$, such that $h, k \in \mathcal{M}$, from all graphs $\mathcal{G}(o)$, $o \in \mathcal{D}$. Symmetrically, in the together branch, we remove arcs $(j', j'')^{\mathcal{M}}$, $\mathcal{M} \subseteq \mathcal{K}_j$, which model the delivery of only one commodity between h and k, i.e., such that $h \in \mathcal{M}$ and $k \notin \mathcal{M}$ (or vice versa). Conversely, Rule 7 concerns the case where i and j represent different customers, i.e., $i \neq j$. Here, both branching decisions entails non-robust constraints in the MP which enforce the introduction of binary resources in the solution of the ESPPRC in the pricing problem. We refer to Gschwind *et al.* (2019) for further details.

Rule 8 is the separation of *strong-degree* inequalities (Contardo *et al.*, 2014).

The branch-and-bound tree is explored according to a best-bound first strategy to favour the improvement of the dual bound. The strategies to select the branching decisions are presented in the following. For rule 2, we branch on the fractional value closest to 0.5. For rules 6 and 7, we branch on the first fractional value f^{jk}_{ih} that is found. For all the other rules, we consider a two-round strong branching procedure Røpke (2012) similar to the one presented in Pessoa *et al.* (2020). In the first round, we evaluate at most 100 branching candidates according to the *product rule* (Achterberg, 2007). More precisely, each candidate gives rise to two branching decisions d_1 and d_2 and is evaluated by applying such decisions to the RMP and by solving it without generating columns. Then, each candidate is assigned with a score $sc(d_1, d_2) = \max\{\epsilon, \Delta LB_1\} \times \max\{\epsilon, \Delta LB_2\}$, where $\epsilon = 10^{-6}$ and ΔLB_i is the increase of the lower bound obtained by applying decision d_i to the RMP. The three candidates

with the highest scores are sent to the second round, where the same evaluation criterion is used to select the winning candidate. Differently from the first round, the RMP is solved with an iteration of the column generation procedure where the pricing problem is solved heuristically.

The strong branching procedure is employed in nodes of the branch-and-bound tree of level at most 5. In the other levels, we evaluate the branching candidates based on the fractional value closest to 0.5 for all the rules.

4.5.4 Accelerating strategies

The BPC algorithm incorporates the following accelerating strategies:

Initialization of set $\mathcal{R}$. We initialize the set of routes $\mathcal{R}$ to avoid very large dual prices at the first iterations of the column generation procedure which may slow down the pricing solution (Desaulniers, 2010). Specifically, for each distribution centre $o \in \mathcal{D}$, we include round-trips (0-j-0) to each customer $j \in \mathcal{N}$, which deliver the commodities of each subset $\mathcal{M}_j \subseteq \mathcal{K}_j$ requested by j. In addition, we modified the randomised Clarke-Wright algorithm (CW) (Clarke & Wright, 1964) proposed in Battarra *et al.* (2008) to take into account the multi-commodity aspect of our problem. The algorithm is run 10 times per distribution centre and the obtained routes are inserted into $\mathcal{R}$.

Heuristic column generators. Before solving the pricing problem to optimality, we consider heuristic column generators to speed up the solution of problems [PP(o)], $o \in \mathcal{D}$.As mentioned in Section 4.5.1, each problem [PP(o)] is the pricing problem arising in a BPC algorithm for the C-SDVRP. Hence, we apply the same heuristic scheme used in Petris *et al.* (2023) which proved to be effective in accelerating such pricing problems. This scheme considers two reduced graph heuristics and the two-phase heuristic introduced in Petris *et al.* (2023) which proved to be effective in dealing with the multi-commodity aspect of the C-SDVRP. The two reduced graph heuristics reduce the size of graphs $\mathcal{G}(o)$, $o \in \mathcal{D}$ by limiting both the possibilities of travelling between customers and of deliveries to customers. In the two-phase heuristic, the aim of the first phase is to compute a set of promising customer sequences by solving the ESPPRC on a modified version of graphs $\mathcal{G}(o)$ where only one delivery per customer is allowed. Specifically, when visiting a customer, the least consuming commodity is delivered and all the profitable dual

prices are collected. In the second phase, for each of the customer sequences generated by the first phase, we solve the ESPPRC on the associated acyclic graphs to obtain all negative reduced cost routes arising from the sequence. We refer to Petris *et al.* (2023) for more details.

Restricted master heuristic. We invoke a restricted master heuristic, which consists in solving the formulation [SC] restricted to the subset of variables generated so far, to obtain good upper bounds. Such a technique helps to reduce the integrality gap (see Archetti *et al.*, 2013). Note that variables λ_r are then binary. We call the restricted master heuristic every 1000 explored nodes in the branch-and-bound tree as well as when the time limit of the algorithm is reached. When the restricted master heuristic is called during the tree exploration a time limit of 3 seconds is imposed, while the time limit is 30 seconds when the algorithm terminates.

4.6 Computational experiments

We implemented the BPC algorithm in C++ and compiled it in release mode under a 64-bit version of MS Visual Studio 2019. IBM CPLEX 12.9.0 (64-bit version) is used as a solver. We performed the experiments on a 64-bit Windows machine equipped with a Intel(R) Xeon(R) Silver 4214 processor with 24 cores hyper-threaded to 48 virtual cores, with a base clock frequency of 2.2 GHz, and 96 GB of RAM. For each run of the algorithm, we impose one hour time limit and allow a single thread.

In this section, first, we describe the characteristics of the benchmark instances for the MC2DP introduced in Gu *et al.* (2022). Then, we discuss the impact of valid inequalities (4.12), (4.16) and (4.17). Finally, we evaluate the effectiveness of the BPC algorithm against solving the compact formulation for the MC2DP presented in Gu *et al.* (2022) with a commercial solver and we present the results obtained by the BPC algorithm on the benchmark instances.

4.6.1 Benchmark instances

Gu *et al.* (2022) introduced artificial instances as well as instances arising from a real-world case study in the context of a short and local fresh food supply chain. In the following computational experiments, we only consider the artificial instances. Indeed,

the sizes of the instances based on the case study are too large to be tackled efficiently with the BPC algorithm.

First, Gu *et al.* (2022) generated a base set of artificial instances $\mathscr{S}$ and, then, they produced 12 additional sets by applying modifications to one of the characteristics of the base set. In Table 4.1, we summarise the main characteristics of all sets of artificial instances, the generation of which is detailed below. Each row in the table represents a set of instances. The columns of the table report: *set*: the name of the set of instances; #: the number of instances in the set; $|\mathcal{S}|$: the number of suppliers; $|\mathcal{C}|$: the number of customers; $|\mathcal{K}|$: the number of commodities; *description*: a brief description of the main characteristic of the set. In all sets of instances, the number of distribution centres is fixed to two.

Table 4.1: Characteristics of the sets of instances.

		Characteristics			
set	#	$\|\mathcal{S}\|$	$\|\mathcal{C}\|$	$\|\mathcal{K}\|$	description
$\mathscr{S}$	64	8	30	2, 3	base set
$\mathscr{S}_1^{\mathcal{S}}$	64	8	30	2, 3	unbalanced supplier locations (6-2)
$\mathscr{S}_2^{\mathcal{S}}$	64	8	30	2, 3	unbalanced supplier locations (8-0)
$\mathscr{S}_1^{\mathcal{C}}$	64	8	30	2, 3	unbalanced customer locations (5-10, with $\delta = -5, 30$)
$\mathscr{S}_2^{\mathcal{C}}$	64	8	30	2, 3	unbalanced customer locations (5-10, with $\delta = 10, 30$)
$\mathscr{S}_3^{\mathcal{C}}$	64	8	30	2, 3	unbalanced customer locations (10-5, with $\delta = -5, 30$)
$\mathscr{S}_4^{\mathcal{C}}$	64	8	30	2, 3	unbalanced customer locations (10-5, with $\delta = 10, 30$)
$\mathscr{S}^{O}$	32	8	30	2	unbalanced available amounts at the suppliers
$\mathscr{S}_1^{\mathcal{S}_{add}}$	64	10	30	2, 3	increased number of supplier to 10
$\mathscr{S}_2^{\mathcal{S}_{add}}$	64	12	30	2, 3	increased number of supplier to 12
$\mathscr{S}_1^{\mathcal{C}_{add}}$	64	8	50	2, 3	increased number of customer to 50
$\mathscr{S}_2^{\mathcal{C}_{add}}$	64	8	70	2, 3	increased number of customer to 70
`small`	36	4, 6	10, 15, 20, 25	2, 3	small instances

First, we describe how Gu *et al.* (2022) generated base set $\mathscr{S}$. The 64 instances of $\mathscr{S}$ involve two distribution centres ($|\mathcal{D}| = 2$), eight suppliers ($|\mathcal{S}| = 8$) and 30 customers ($|\mathcal{C}| = 30$). The features of the delivery echelon are based on the 64 small instances proposed in Archetti *et al.* (2016) for the C-SDVRP. Such instances are characterised by: (i) 15 customers; (ii) customers location from the C101 or R101 instances for the *Vehicle Routing Problem with Time Windows* (Solomon, 1987); (iii) number of commodities $|\mathcal{K}|$ equal to two or three; (iv) probability p that a customer requires a commodity with a non-zero demand equal to 0.6 or 1; (v) interval Δ to select the non-zero commodity demands of a customer equal to $[1, 100]$ or $[40, 60]$; (vi) parameter α to

determine the vehicle capacity ($Q^D = \alpha \max\{\sum_{k\in\mathcal{K}_j} D_{jk} : j \in \mathcal{N}\}$) in $\{1.1, 1.5, 2, 2.5\}$. Each C-SDVRP instance gives rise to a MC2DP instance where the locations of one distribution centre and 15 customers are the ones of the C-SDVRP instance. Customer demands are also as in the C-SDVRP instance. Such distribution centre and 15 customers are duplicated and their locations are modified. Specifically, a translation of $\delta = 30$ is applied to both locations' coordinates. In addition, customer locations are rotated of 180 degrees around the modified location of the distribution centre. Then, four suppliers are randomly located around each distribution centre within a circle of radius equal to 30. The availability of commodity $k \in \mathcal{K}$ at supplier $i \in \mathcal{S}$ is set to $P_{ik} = \left\lceil \zeta \frac{\sum_{j\in\mathcal{C}} D_{jk}}{|\mathcal{S}|} \right\rceil$, where $\zeta = 1.2$.

In the following, we report how Gu *et al.* (2022) obtained the remaining 12 sets of instances from base set $\mathscr{S}$.

Sets $\mathscr{S}_1^{\mathcal{S}}$ and $\mathscr{S}_2^{\mathcal{S}}$ (second and third rows in Table 4.1) are obtained by unbalancing the number of suppliers around the distribution centres. Specifically, in set $\mathscr{S}_1^{\mathcal{S}}$, the two distribution centres are surrounded by respectively, six and two suppliers. In set $\mathscr{S}_2^{\mathcal{S}}$ all the eight suppliers are located around one distribution centre.

Sets $\mathscr{S}_1^{\mathcal{C}}$, $\mathscr{S}_2^{\mathcal{C}}$, $\mathscr{S}_3^{\mathcal{C}}$ and $\mathscr{S}_4^{\mathcal{C}}$ (fourth to seventh rows in Table 4.1) are obtained by modifying the locations of some of the 15 duplicated customers. Precisely, in sets $\mathscr{S}_1^{\mathcal{C}}$ and $\mathscr{S}_2^{\mathcal{C}}$ the translation parameter δ used to compute the locations of the first five duplicated customers is set to $\delta = -5$ and $\delta = 10$, respectively. The same parameter values are used to compute the locations of the first 10 duplicated customers in sets $\mathscr{S}_3^{\mathcal{C}}$ and $\mathscr{S}_4^{\mathcal{C}}$. The locations of the remaining customers is determined as for the instances in base set $\mathscr{S}$, i.e., with $\delta = 30$.

The instances of set $\mathscr{S}^O$ (eighth row of Table 4.1) are characterised by unbalanced amounts of commodities available at the suppliers. This set counts 32 instances, as in Gu *et al.* (2022) only the instances in base set $\mathscr{S}$ with two commodities were modified. The amounts of commodities 1 and 2 available at the four suppliers around one of the two distribution centres are set to $P_{i1} = \left\lceil \zeta \frac{1.8\sum_{j\in\mathcal{C}} D_{j1}}{|\mathcal{S}|} \right\rceil$ and $P_{i2} = \left\lceil \zeta \frac{0.2\sum_{j\in\mathcal{C}} D_{j2}}{|\mathcal{S}|} \right\rceil$, respectively. The availability of commodities 1 and 2 is reversed for the suppliers around the other distribution centre.

In sets $\mathscr{S}_1^{\mathcal{S}_{add}}$ and $\mathscr{S}_2^{\mathcal{S}_{add}}$ (ninth and tenth rows of Table 4.1) the number of suppliers is increased to 10 and 12, respectively. As in set $\mathscr{S}$, each distribution centre is surrounded by half of the suppliers.

Similarly, in sets $\mathscr{S}_1^{\mathcal{C}_{add}}$ and $\mathscr{S}_2^{\mathcal{C}_{add}}$ (eleventh and twelfth rows of Table 4.1) the number of customers is increased to 50 and 70, respectively. The additional customers are copies of customers in the corresponding instance of the base set. Their location is modified by applying a translation of parameter randomly chosen in $[-20, 20]$.

Finally, the 36 instances in set `small` are characterised by a reduced number of customers and suppliers (see the last row of Table 4.1). Differently from the other sets, the instances in `small` are obtained from instances of $\mathscr{S}$, $\mathscr{S}_1^{\mathcal{S}}$ and $\mathscr{S}_2^{\mathcal{S}}$. Gu *et al.* (2022) generated such instances to compare the quality of the solutions obtained by their heuristic approach against the solutions obtained by solving a compact formulation for the MC2DP with a commercial solver.

4.6.2 Impact of valid inequalties

In this section, we assess the impact of valid inequalities. To do so, we consider the 32 instances of base set $\mathscr{S}$ having three commodities. Indeed, as mentioned in Section 4.4.1, if the number of commodities is equal to two, inequalities (4.12), (4.16) and (4.17) can be retrieved as an aggregation of covering constraints (4.4).

We examine the following four variants of the BPC algorithm. `BPC`: valid inequalities on bounds on the number of vehicles are inserted, and no valid inequalities is separated in the course of the algorithm; `BPC+CC`: only capacity cuts (valid inequalities (4.11)) are separated; `BPC+SC+NP`: only the inequalities arising from the set covering polytope (`SC`), i.e., inequalities (4.12), and the ones arising from the number partitioning polytope (`NP`) are separated, i.e., inequalities (4.16) and (4.17), are separated; `BPC+CC+SC+NP`: all valid inequalities are separated.

Each row of Table 4.2 corresponds to a BPC variant. The first two columns report the average lower bound (*avg.LB*) and time (*avg.t[s]*) at the root node of the branch-and-bound-tree. The next four columns show the results at the end of the execution of the corresponding BPC variant: the average number of nodes of the branch-and-bound tree (*avg.#nodes*), the average lower bound at termination (*avg.LB*) the average time (*avg.t[s]*) and the number of instances solved to optimality (*#opt./#inst.*) over the 32 instances.

Table 4.2: Comparison of four variant of the BPC algorithm

BPC variant	Root node		BPC results			
	avg.LB	avg.t[s]	avg.#nodes	avg.LB	avg.t[s]	#opt./#inst.
BPC	983.00	25.30	3235.72	1028.08	3145.03	6/32
BPC+CC	1000.35	54.51	1721.53	1039.61	2484.34	14/32
BPC+SC+NP	985.36	46.19	2970.78	1029.42	2992.08	7/32
BPC+CC+SC+NP	1001.31	76.40	1623.84	1039.61	2454.58	14/32

As expected, BPC yields the worst results solving only six instances out of the 32 considered. Variant BPC+SC+NP solves an additional instance w.r.t. BPC, however, the improvement of the lower bound at the root node is mediocre. The best results are obtained when the well-established capacity cuts are separated, namely with variants BPC+CC and BPC+CC+SC+NP. Both variants solve the same 14 instances to optimality and yield the best lower bounds at the root node, being on average equal to 1000.35 and 1001.31 in BPC+CC and BPC+CC+SC+NP, respectively. The same remark applies to the lower bounds at termination which is on average equal to 1039.61 in BPC+CC and BPC+CC+SC+NP. In both cases, lower bounds at the root node and at termination improve significantly with respect to BPC. We also observe that the addition of inequalities (4.12), (4.16) and (4.17) in BPC+CC+SC+NP slightly improves the results with respect to BPC+CC in terms of lower bounds at the root node, number of explored branch-and-bound nodes and solution time. Hence, we choose BPC+CC+SC+NP as the configuration for our BPC algorithm.

4.6.3 Evaluation of the BPC algorithm

The aim of this section is to evaluate the effectiveness of the BPC algorithm. To do so, we compare the results obtained by the BPC algorithm on the instances of set small with the ones obtained by solving a compact formulation for the MC2DP on the same instances with CPLEX 12.8. The latter results are retrieved from Gu *et al.* (2022) and were obtained on a machine with Intel (R) Core(TM) i7-4600U processor with a base clock frequency of 2.10GHz and with 16 GB of RAM. The performances of such machine are comparable with the ones of our machine given that we allow a single thread in the experiments. A time limit of one hour is imposed on both methods.

Table 4.3 presents the results of the comparison. Each row of the table corresponds to an instance in set `small`. The first five columns report some characteristics of the instance (see Section 4.6.1).The following five columns report the results of the BPC algorithm: *#nodes*: number of nodes of the branch-and-bound tree; LB: lower bound at termination; UB: value of the best solution found; *gap[%]*: percentage optimality gap ($100((UB - LB)/LB)$) if the instance in not solved to optimality, *opt* otherwise; *t[s]*: total computational time in seconds of the BPC algorithm. Last, in the last two columns, we report the optimality gap (*gap[%]*) and computational time (*t[s]*) obtained by Gu *et al.* (2022) when solving the compact formulation. In column *gap[%]*, a '-' indicates that CPLEX was not able to provide a feasible solution.

The results of Table 4.3 show that the BPC algorithm proved to be a effective as it provides 34 optima over 36 instances. The two unsolved instances are with 20 and 25 customers and are left with an optimality gap of 0.74% and 3.57%, respectively. Conversely, the performances of the compact formulation deteriorates as the size of the instances grows. The formulation provides only eight optima, all obtained for instances with 10 customers, and it fails to return a feasible solution for two instances with 25 customers. The average optimality gap of the remaining 26 instances is 31.73%. Finally, when both approaches prove the optimality of a solution, the BPC algorithm is generally faster than the compact formulation by at least one order of magnitude.

4.6.4 Results on the whole testbed

In this section, we present a summary of the results obtained by the BPC algorithm in Tables 4.4 and 4.5 on the 12 sets of benchmark instances with one-hour time limit. The instance-by-instance results can be found at `https://hal.inria.fr/hal-03946477v1`.

In Table 4.4, we report results only for the instances solved to optimality and table 4.5 summarises the results for the remaining instances. Each row of both tables corresponds to a subset of instances from the same set and with the same number of commodities. The first three columns of the tables report some information about the instance subset (see Section 4.6.1). The column headings of Table 4.4 are defined as follows: *#opt.*: number of considered instances (solved to optimality); *#nodes avg./min./max.*: average/minimum/maximum number of nodes of the branch-and-bound tree; $avg.gap^{root}$*[%]*: average optimality gap at the root node expressed as a percentage, i.e., $100((OPT - LB^{root})/LB^{root})$, where OPT is the value of the optimal

Table 4.3: Results on set `small`.

Instances					BPC					CPLEX	
$\|\mathcal{S}\|$	$\|\mathcal{C}\|$	$\|\mathcal{K}\|$	p	set	#nodes	LB	UB	gap[%]	t[s]	gap[%]	t[s]
		2	0.6	$\mathscr{S}$	161	394.655	394.655	opt	5.26	opt	240.00
		2	1	$\mathscr{S}$	17	579.522	579.522	opt	0.98	5.55	3600.00
		3	0.6	$\mathscr{S}$	7787	470.77	470.77	opt	1446.16	opt	378.00
		2	0.6	$\mathscr{S}_1^S$	101	406.52	406.52	opt	4.28	opt	108.00
4	10	2	1	$\mathscr{S}_1^S$	11	562.34	562.34	opt	1.19	opt	1441.00
		3	0.6	$\mathscr{S}_1^S$	227	437.98	437.98	opt	14.10	opt	486.00
		2	0.6	$\mathscr{S}_2^S$	23	406.52	406.52	opt	1.11	opt	57.00
		2	1	$\mathscr{S}_2^S$	13	663.52	663.52	opt	0.66	opt	2261.00
		3	0.6	$\mathscr{S}_2^S$	19	463.58	463.58	opt	0.80	opt	41.00
		2	0.6	$\mathscr{S}$	181	510.88	510.88	opt	10.75	8.45	3600.00
		2	1	$\mathscr{S}$	15	742.71	742.71	opt	1.37	17.89	3600.00
		3	0.6	$\mathscr{S}$	13785	551.87	551.87	opt	3469.76	6.87	3600.00
		2	0.6	$\mathscr{S}_1^S$	195	533.43	533.43	opt	13.98	13.82	3600.00
4	15	2	1	$\mathscr{S}_1^S$	3185	784.05	784.05	opt	349.18	15.79	3600.00
		3	0.6	$\mathscr{S}_1^S$	3829	553.49	553.49	opt	624.91	20.42	3600.00
		2	0.6	$\mathscr{S}_2^S$	33	590.55	590.55	opt	2.32	7.25	3600.00
		2	1	$\mathscr{S}_2^S$	17	893.09	893.09	opt	1.67	20.62	3600.00
		3	0.6	$\mathscr{S}_2^S$	117	590.71	590.71	opt	11.41	15.26	3600.00
		2	0.6	$\mathscr{S}$	45	636.71	636.71	opt	7.64	22.09	3600.00
		2	1	$\mathscr{S}$	233	1007.04	1007.04	opt	24.61	31.75	3600.00
		3	0.6	$\mathscr{S}$	63	708.62	708.62	opt	24.26	28.15	3600.00
		2	0.6	$\mathscr{S}_1^S$	139	659.37	659.37	opt	35.82	37.59	3600.00
4	20	2	1	$\mathscr{S}_1^S$	13848	1069.54	1077.43	0.74	3631.00	30.18	3600.00
		3	0.6	$\mathscr{S}_1^S$	1209	768.57	768.57	opt	528.33	37.92	3600.00
		2	0.6	$\mathscr{S}_2^S$	43	713.16	713.16	opt	7.29	27.87	3600.00
		2	1	$\mathscr{S}_2^S$	5	1177.46	1177.46	opt	0.98	43.37	3600.00
		3	0.6	$\mathscr{S}_2^S$	157	835.00	835.00	opt	74.17	37.47	3600.00
		2	0.6	$\mathscr{S}$	209	815.15	815.15	opt	70.15	-	3600.00
		2	1	$\mathscr{S}$	661	1184.62	1184.62	opt	227.48	31.32	3600.00
		3	0.6	$\mathscr{S}$	555	826.12	826.12	opt	334.55	58.7	3600.00
		2	0.6	$\mathscr{S}_1^S$	287	784.11	784.11	opt	70.08	37.76	3600.00
6	25	2	1	$\mathscr{S}_1^S$	1111	1258.91	1258.91	opt	310.47	54.77	3600.00
		3	0.6	$\mathscr{S}_1^S$	4073	876.72	908.00	3.57	3632.66	87.9	3600.00
		2	0.6	$\mathscr{S}_2^S$	35	881.02	881.02	opt	9.95	-	3600.00
		2	1	$\mathscr{S}_2^S$	99	1367.61	1367.61	opt	29.11	60.19	3600.00
		3	0.6	$\mathscr{S}_2^S$	15	939.52	939.52	opt	18.84	65.95	3600.00

solution found by the BPC algorithm and LB^{root} is the lower bound at the root node after the valid inequalities have been inserted; *avg.t[s]*: average computational time; *dev. Gu et al. (2022) avg./min./max.*: average/minimum/maximum deviation from the best solution value UB reported in Gu *et al.* (2022), i.e., $100((OPT - UB)/UB)$.

Table 4.4: Aggregated results on the instances solved to optimality by the BPC algorithm

Instances			BPC results						dev. Gu *et al.* (2022)		
				#nodes							
set	$\|\mathcal{K}\|$	#	#opt.	avg.	min.	max.	avg.gap^{root}[%]	avg.t[s]	avg.	min.	max.
$\mathscr{S}$	2	32	31	791.19	11	7037	3.50	241.78	-0.19	-2.21	0.00
	3	32	14	751.79	23	2197	4.24	921.14	-0.95	-2.61	0.00
$\mathscr{S}_1^{\mathcal{S}}$	2	32	30	2126.47	47	21125	4.90	621.02	-0.19	-1.69	0.00
	3	32	15	872.00	29	2674	4.51	837.63	-1.02	-3.08	0.00
$\mathscr{S}_2^{\mathcal{S}}$	2	32	26	3350.61	51	18367	4.93	768.70	-1.65	-5.31	0.00
	3	32	20	1932.25	47	12259	4.95	1417.92	-2.11	-6.71	0.00
$\mathscr{S}_1^{\mathcal{C}}$	2	32	29	667.83	13	2669	3.82	430.16	-0.61	-2.94	0.00
	3	32	15	600.13	39	1547	3.48	872.74	-1.13	-4.04	0.00
$\mathscr{S}_2^{\mathcal{C}}$	2	32	27	674.85	5	3213	4.08	449.58	-0.66	-3.00	0.00
	3	32	15	780.53	81	1855	3.98	1058.08	-1.01	-3.63	0.00
$\mathscr{S}_3^{\mathcal{C}}$	2	32	24	2627.50	25	41105	4.58	747.43	-0.65	-3.64	0.00
	3	32	15	1040.20	141	4269	4.81	1125.06	-0.91	-3.49	0.00
$\mathscr{S}_4^{\mathcal{C}}$	2	32	26	958.85	15	11217	4.21	471.72	-0.88	-3.70	0.00
	3	32	13	978.23	219	2109	4.06	909.64	-0.83	-3.09	0.00
$\mathscr{S}^O$	2	32	14	5925.57	435	20135	6.44	1360.41	-2.34	-4.24	0.00
$\mathscr{S}_1^{\mathcal{S}_{add}}$	2	32	30	1339.33	79	12299	5.34	444.41	-0.35	-1.99	0.00
	3	32	17	727.18	87	2247	4.57	904.53	-0.16	-2.55	0.00
$\mathscr{S}_2^{\mathcal{S}_{add}}$	2	32	32	1160.37	1	17735	6.43	284.21	-0.12	-1.77	0.00
	3	32	23	622.13	65	2651	7.07	826.92	-0.09	-1.64	0.00
$\mathscr{S}_1^{\mathcal{C}_{add}}$	2	32	16	1900.87	43	5481	3.08	1292.03	-0.23	-1.12	0.00
	3	32	4	404.25	27	728	2.53	875.52	-0.04	-0.16	0.00
$\mathscr{S}_2^{\mathcal{C}_{add}}$	2	32	3	2884.33	663	5855	1.68	1159.56	-0.71	-0.97	-0.36
	3	32	0	-	-	-	-	-	-	-	-

Table 4.4 shows that the BPC algorithm identifies 439 optima over the 736 instances.

The number of nodes of the branch-and-bound tree varies widely: it ranges from 1 to 41105 and its average is 1458 while its standard deviation is 3237. Note that we found no correlation between the number of nodes of the branch-and-bound tree and the gap at the root node. The average time needed to prove the optimality of a solution is 720 seconds. Among the 439 optima provided by the BPC algorithm, 416 are obtained on the 10 sets of instances with 30 customers (first ten sets in Table 4.4).

We observe that, except for set $\mathscr{S}^{\mathcal{O}}$, the BPC algorithm behaves homogeneously on instances with 30 customers, i.e., on $\mathscr{S}$, $\mathscr{S}_1^{\mathcal{S}}$, $\mathscr{S}_2^{\mathcal{S}}$, $\mathscr{S}_1^{\mathcal{C}}$, $\mathscr{S}_2^{\mathcal{C}}$, $\mathscr{S}_3^{\mathcal{C}}$, $\mathscr{S}_4^{\mathcal{C}}$, $\mathscr{S}_1^{\mathcal{S}_{add}}$ and $\mathscr{S}_2^{\mathcal{S}_{add}}$. Indeed, the BPC algorithm solves to optimality at least 60.9% of the 64 instances belonging to each set. This percentage increases to 73.4% and 85.9% for the two sets of instances with a larger number of suppliers (see $\mathscr{S}_1^{\mathcal{S}_{add}}$ and $\mathscr{S}_2^{\mathcal{S}_{add}}$ in Table 4.4). Increasing the number of suppliers seems to make the instances easier to solve. In addition, in each of the sets with 30 customers, the BPC algorithm proves the optimality of almost all the instances with two commodities (at least 24 out of 32) and of around half of the instances with three commodities (on average 16 out of 32). Hence, we can conclude that the BPC algorithm seems rather insensitive with respect to the distinctive characteristics of the sets of instances with 30 customers, i.e., unbalanced customer/supplier locations and an increased number of suppliers. Conversely, increasing the number of customers has a major impact on the performance of the BPC algorithm. Indeed, when the number of customers increases to 50 and 70 (see sets $\mathscr{S}_1^{\mathcal{C}_{add}}$ and $\mathscr{S}_2^{\mathcal{C}_{add}}$ in Table 4.4), the number of optima decreases to 20 and 3, respectively.

Finally, we note that the sequential heuristic of Gu *et al.* (2022) was able to identify 220 out of 439 optima. For the remaining instances, the BPC algorithm improves the solution values found by Gu *et al.* (2022) by 1.46% on average (see the last three columns of Table 4.4).

Table 4.5 reports the results on the instances not solved to proven optimality by the BPC algorithm. The meaning of the rows and columns in Table 4.5 is similar to the ones of Table 4.4. The differences are: the column *#opt.* is replaced with the column *#notOpt.* which indicates the number of instances not solved to proven optimality, and the column *avg.t[s]* is replaced by the columns *gap[%] avg./min./max.* reporting the average, minimum, and maximum optimality gap computed as $100((UB - LB)/LB)$, where UB is the value of the best solution found by the BPC algorithm and LB is the lower bound when the time limit is reached. Similarly, the average gap at the root

node $avg.gap^{root}[\%]$:] and the average deviation from the best solution value reported in Gu *et al.* (2022) are computed by replacing the optimal value OPT with UB.

Table 4.5: Aggregated results on the instances not solved to proven optimality by the BPC algorithm

Instances			BPC results									
				#nodes			gap[%]				dev. Gu *et al.* (2022)	
set	$\|\mathcal{K}\|$	#	#notOpt.	avg.	min.	max.	avg.	min.	max.	avg.gap^{root}[%]	avg.	min.	max.
$\mathscr{S}$	2	32	1	653.00	653	653	3.31	3.31	3.31	14.08	2.07	2.07	2.07
	3	32	18	2319.17	29	13135	2.44	0.01	6.46	6.80	0.92	-1.95	5.85
$\mathscr{S}_1^{\mathtt{S}}$	2	32	2	3880.00	940	6820	3.38	0.41	6.35	11.97	1.01	-0.06	2.09
	3	32	17	3057.65	19	16126	3.01	0.20	10.01	8.82	0.91	-2.67	5.10
$\mathscr{S}_2^{\mathtt{S}}$	2	32	6	12069.00	941	46006	0.37	0.06	1.03	6.92	-1.15	-3.61	0.07
	3	32	12	3212.42	40	13254	1.87	0.01	4.71	6.69	-1.30	-5.89	3.67
$\mathscr{S}_1^{\mathtt{C}}$	2	32	3	29380.33	12463	40874	0.98	0.03	1.77	4.09	-0.58	-0.90	-0.29
	3	32	17	1913.18	1	10229	2.61	0.34	7.11	6.68	0.16	-1.78	3.36
$\mathscr{S}_2^{\mathtt{C}}$	2	32	5	18695.80	7	40567	0.78	0.03	2.82	3.55	-0.43	-1.00	0.02
	3	32	17	1856.53	23	9311	2.58	0.21	6.09	7.00	0.08	-1.67	2.86
$\mathscr{S}_3^{\mathtt{C}}$	2	32	8	4371.88	380	12552	1.09	0.12	2.25	5.34	0.17	-0.92	1.94
	3	32	17	2070.71	29	12984	2.72	0.03	10.89	7.65	0.85	-1.31	10.69
$\mathscr{S}_4^{\mathtt{C}}$	2	32	6	4876.50	342	12698	1.09	0.10	1.94	5.78	0.11	-1.13	1.62
	3	32	19	1463.68	10	12005	2.47	0.18	5.96	7.23	0.41	-1.68	2.26
$\mathscr{S}^O$	2	32	18	9788.83	1033	50109	2.24	0.13	9.17	8.15	0.05	-2.98	4.85
$\mathscr{S}_1^{\mathtt{S}_{add}}$	2	32	2	774.50	763	786	2.37	1.05	3.69	18.86	1.32	0.69	1.96
	3	32	15	2484.00	37	12241	2.60	0.00	8.76	10.65	0.65	-1.77	2.40
$\mathscr{S}_2^{\mathtt{S}_{add}}$	2	32	0	-	-	-	-	-	-	-	-	-	-
	3	32	9	2990.11	16	14444	1.99	0.07	6.03	9.21	0.75	0.00	4.05
$\mathscr{S}_1^{\mathtt{C}_{add}}$	2	32	16	4300.00	180	32352	1.15	0.00	3.52	6.27	0.25	-0.84	2.11
	3	32	28	1344.14	1	6585	2.60	0.08	10.19	6.06	0.65	-1.20	4.63
$\mathscr{S}_2^{\mathtt{C}_{add}}$	2	32	29	2141.07	337	14413	1.15	0.04	3.78	3.64	-0.18	-1.83	2.98
	3	32	32	684.22	1	2820	2.45	0.04	9.31	3.97	0.26	-2.32	2.93

The BPC algorithm cannot prove the optimality for 297 instances. For these instances, the average optimality gap at the root node is 6.63%. However, the exploration of the branch-and-bound tree allows the optimality gap to be reduced to 2.1% on av-

erage. The final optimality gap is larger than 5% for only 29 instances. The number of nodes of the branch-and-bound tree follows the trend observed in Table 4.4: it varies greatly, with an average of 3429 nodes, while the standard deviation is 6809. The comparison with Gu *et al.* (2022) (last three columns of Table 4.5) shows mixed results. The BPC algorithm finds larger upper bounds for 161 instances. On these instances, the average deviation is 1.23%. For 24 instances, the BPC algorithm finds the same value as the one reported by Gu *et al.* (2022). Finally, for the remaining 112 instances, the BPC algorithm provides a lower value with an average improvement of 1.02%.

4.7 Conclusions

In this paper, we presented a Branch-Price-and-Cut (BPC) algorithm to solve the Multi-Commodity two-echelon Distribution Problem (MC2DP), a two-echelon routing problem where multiple commodities are sent from suppliers to customers via distribution centres. The collection operations are done by capacitated vehicles performing direct round trips between the distribution centres and the suppliers. The delivery operations are also performed by capacitated vehicles. Each delivery vehicle performs a route starting and ending at the same distribution centre. Customers are allowed to be visited multiple times, provided that the amount of a single commodity is delivered at once by a single vehicle. Commodities can be mixed inside all vehicles. The objective is to minimise the transportation costs of the distribution system.

The BPC algorithm incorporates several state-of-the-art accelerating techniques and three families of robust valid inequalities: capacity cuts, valid inequalities arising from the set covering polytope, and a new family of valid inequalities based on the number partitioning polytope. The inequalities improve the lower bound at the root node and reduce the number of nodes of the branch-and-bound tree and the computational time. The BPC algorithm is able to solve to optimality nearly 60% of the benchmark instances introduced in Gu *et al.* (2022) within one-hour time limit. The final optimality gap is reasonable for the remaining instances, with an average value of 2.1%. Finally, we identified 331 new best-known solutions compared to the results of Gu *et al.* (2022).

The main issue with the instances left unsolved by the BPC algorithm is the large optimality gap at the root node. To overcome this difficulty, future research should be devoted to the inclusion of new dedicated valid inequalities. Adding non-robust valid

inequalities known for routing problems is also an interesting perspective. However, it would lead to more difficult pricing problems to solve. In addition, Gu *et al.* (2022) proposed a sequential heuristic for the MC2DP. Therefore, another line of research could be the development of heuristic algorithms that address the problem from an integrated point of view.

Chapter 5

Solving the Kidney Exchange Problem with long cycles and chains via a Branch-Price-and-Cut algorithm

Contents

Abstract: In this paper, we study a Kidney Exchange Problem (KEP) with altruistic donors and incompatible patient-donor pairs. Kidney exchanges can be modelled in a directed graph as circuits starting and ending in an incompatible pair or as paths starting at an altruistic donor. For medical reasons, both circuits and paths are of limited length and are associated with a medical benefit which evaluates the quality of the transplants. The aim of the KEP is to determine a set of disjoint kidney exchanges of maximal medical benefit or of maximal cardinality.

We consider an extended set packing formulation for the KEP where the exponentially-many variables correspond to the circuits and paths, and develop a Branch-Price-and-Cut algorithm (BPC) to solve it. We show that the pricing problem can be decomposed into a subproblem to price out the variables associated with paths and several subproblems, one per each incompatible pair, to price out the variables associated with circuits. The subproblem to price out paths can be formulated as a variant of the Elementary Shortest Path Problem with Resource Constraints (ESPPRC), that is NP-hard in the strong sense, and it is solved by means of a label correcting dynamic programming algorithm. Conversely, the subproblems to price out circuits can be formulated as a variant of the Shortest Path Problem with Resource Constraint (SPPRC) that can be solved in polynomial time by a variant of the Bellman-Ford algorithm. In our BPC algorithm, we separate two families of non-robust valid inequalities, namely, the subset-row and the odd-hole inequalities.

We perform extensive computational experiments to assess the performances of the BPC algorithm on three sets of instances from the literature. On each set of instances, an algorithm for the KEP from the literature is used as a reference method. On the set with the easiest instances, the BPC algorithm yields comparable results with the literature, and it is able to outperform the results on the two other sets.

Keywords: Kidney exchange, altruistic donors, elementary paths, elementary circuits, Branch-Price-and-Cut.

5.1 Introduction

Kidneys are essential organs for the survival of the human body: (i) they produce the urine by filtering the blood to expel wastes and toxins; (ii) they play a role in maintaining the homeostasis of the body by regulating the acid-base balance, the concentrations

of electrolyte; (iii) they secrete several hormones responsible, for example, of the maturation of the red blood cells or of regulating the blood pressure. Unfortunately, kidneys may suffer from chronic diseases or failures which prevent them to perform their usual tasks. According to Kovesdy (2022), more than 10% of the worldwide population was affected by such diseases in 2022.

Chronic kidney diseases or failures are often treated by dialysis or transplantation (Levey & Coresh, 2012). Transplantation is the most preferable treatment: it affects less the quality of life of the patients, it offers a longer expectancy of life and it is more cost efficient (see, e.g., Axelrod *et al.*, 2018; Yoo *et al.*, 2016). However, such operations have to be performed with extreme care in order to minimise the risk of rejection. For this reason, several medical requirements must be met for a patient and a donor to be considered eligible for a transplant. Precisely, patient and donor must be *compatible* according to several indicators such as blood type and presence of specific antibodies (Kälble *et al.*, 2005). In addition, when different patients and donors are compatible between each other, each possible transplant is assigned with a *medical benefit* which quantifies the medical interest of performing the transplant.

Kidneys are harvested from either deceased or living donors. Usually, the former case constitutes the majority of the transplants. In 2021, more than the 60% of the 92 532 worldwide transplants involved a deceased donor*. However, patients must register to a waiting list and might wait several years before (possibly) receiving a kidney from a compatible deceased donor (Lentine *et al.*, 2023). Transplants from living donors may help patients in receiving a kidney more promptly (Nemati *et al.*, 2014). Such transplants are possible because one functioning kidney is enough for a human being to conduct a healthy life. The organisation of transplants from living donors is more complex than the one from deceased donors. Indeed, patients need to find a willing and compatible donor in their circle of acquaintances that is often not easy. Most of the times, a patient is capable of finding a willing donor with whom it is unfortunately not compatible. However, together, they can form an *incompatible patient-donor pair* and join a *kidney exchange programme* (see Rapaport (1986) and Roth *et al.* (2004)). Such programmes aim to determine the best (in terms of medical benefit) set of kidney exchanges in a pool of incompatible patient-donor pairs. An *exchange* takes place when a donor of a pair gives a kidney to a patient of another pair with whom he/she is compatible. For obvious reasons, in this context, a donor is willing to be in an exchange

*https://www.transplant-observatory.org/

only if his/her associated patient is guaranteed to receive a kidney as well. Hence, the set of exchanges to be determined must give rise to *cycles of donations*. The transplants associated with a cycle of donation are usually performed simultaneously to avoid the risk of donor withdrawals. Hence, for practical reasons (e.g., limited facilities or medical staff), regulations impose a maximal number of pairs in a cycle, usually set up to four (Biró *et al.*, 2019a). However, we report the breakthrough cases of cycles involving five, six or seven pairs in the Czech-Austrian kidney exchange programme (Viklicky *et al.*, 2020). In recent years, *altruistic donors*, i.e., donors who are not paired with any patient, may join the pool of a kidney exchange programme (Roodnat *et al.*, 2010). Thanks to their presence, the set of exchanges may also include *chains of donations* (domino donations) where the altruistic donors are at the start of the chains. In a chain of donation, surgeries are not required to be performed simultaneously, hence, the number of incompatible pairs involved in them, although usually limited by regulations, is larger than the one of the cycles*

Typically, kidney exchange programmes are run regularly (e.g., each one, three, four months depending on the country regulation (Biró *et al.*, 2021)) within one or more hospitals belonging to a single or different countries. At each round, an optimisation algorithm is called to solve the *Kidney Exchange Problem* (KEP) whose aim is to determine a set of cycles and chains of donations in the current pool of incompatible patient-donor pairs and altruistic donors such that the overall medical benefit is maximised.

In our work, we study a unified exact approach to solve the KEP with long cycles and chains. Recently, Riascos-Álvarez *et al.* (2020) and Arslan *et al.* (2022) proposed two *Branch-and-Price* (BP) algorithms to solve the same problem. Both works employ an extended formulation based on the *Extended Edge Formulation* introduced in Constantino *et al.* (2013). The exponentially-many variables correspond to the cycles and chains. Hence, in the column generation procedure, the pricing problem is decomposed in several subproblems to price out variables associated with cycles or chains. All these subproblems can be formulated as variants of the *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC), where the unique resource is the length. The ESPPRC is strongly NP-hard. However, the pricing subproblems associated with the cycles can be solved in polynomial time. Indeed, even for a generic resource, the elementarity constraints can be dropped (sub-tours are cycles themselves). Hence, pricing

*https://www.uwhealth.org/news/uw-health-led-nation-in-paired-kidney-exchanges-in-2020

cycles can be done in pseudo-polynomial time by solving the *Shortest Path Problem with Resource Constraints* (SPPRC). In addition, when the resource is the length the SPPRC can be solved in polynomial time, given that the maximal length is bounded by the number of vertices in the graph. The same does not apply in the case of pricing variables associated with chains even if the only resource is the length. The complexity remains strongly NP-hard due to the elementarity requirement.

In this work, we propose a *Branch-Price-and-Cut* (BPC) algorithm based on the *cycle formulation* for the KEP introduced by Roth *et al.* (2007) where the exponentially-many variables corresponds to cycles and chains. We exploit the theoretical results of Arslan *et al.* (2022) to solve as few as possible ESPPRCs. However, instead of calling an integer program to solve the ESPPRC in the case of the chains, as done by Arslan *et al.* (2022), we make use of a labelling algorithm. Labelling algorithms are more efficient than solving integer programs and they allow to manage the dual variables of non-robust inequalities in an easier manner. In this respect, we separate two families of non-robust valid inequalities, namely the *subset-row* and *odd-hole* inequalities, which help in strengthening the linear relaxation for the hardest instances. In a thorough computational experiments, the BPC algorithm shows its superiority against other approaches for the KEP in the literature on most of the benchmark instances.

The reminder of the paper is organized as follows. In section 5.2, we review the existing literature regarding the KEP. In Section 5.3, we formally introduce the KEP and we present a set packing formulation for the problem. The main components of our Branch-Price-and-Cut algorithm are described in Section 5.4. Finally, in Section 5.5, we report and analyse the results obtained by the BPC algorithm on three sets of benchmark instances.

5.2 Literature review

In this section, we survey the existing literature on solution approaches for the Kidney Exchange Problem (KEP).

Roth *et al.* (2005) study the KEP with only cycles of length two. Such variant of the KEP can be reduced to a *Maximum Weighted Matching Problem* and thus solved in polynomial time by, e.g., the Edmond's algorithm (Edmonds, 1965). However, the KEP with cycles of length larger than two (and chains of arbitrary length) is proven

to be NP-hard (Abraham *et al.*, 2007; Roth *et al.*, 2007). For this reason, several formulations and solution methods have been proposed through the years.

In the following we review the main formulations proposed for the KEP with cycles. Roth *et al.* (2007) introduced the *edge formulation* for the KEP, a first formulation with an exponential number of constraints and a polynomial number of variables w.r.t. the size of the instance. Whereas the first extended formulation, the *cycle formulation*, was introduced by Roth *et al.* (2007) and Abraham *et al.* (2007) the same year. The authors proved that the cycle formulation dominates the edge formulation in terms of linear relaxation. As a solution method, they considered a column generation approach where the pricing problem is solved by enumeration. Constantino *et al.* (2013) proposed the first compact formulation for the KEP with cycles only, the *extended edge formulation*. Klimentova *et al.* (2014) presented the *disaggregated cycle formulation*, a variant of the cycle formulation where the set of cycles is decomposed in subsets, one per each pair, containing the cycles starting at that pair. Both Constantino *et al.* (2013) and Klimentova *et al.* (2014) adopt an idea to break symmetries: a cycle starting at pair i does not contain pairs with an index lower than i.

In addition, Dickerson *et al.* (2016) introduced the *position-indexed edge formulation*, a compact formulation which achieves equivalent linear relaxation bounds as the cycle formulation. More recently, Delorme *et al.* (2023) exploited the idea of representing a cycle by two compatible halves and introduced the *half-cycle formulation* to solve the KEP with cycles only and where the objective is to maximise the number of transplants. Such formulation has less variables than the cycle formulation while keeping the same quality of the linear relaxation bound. The authors proposed to solve the formulation by enumerating all the possible half-cycles and by applying a destructive bound procedure coupled with a variable-fixing strategy. Precisely, first, an upper bound on the maximum number of transplants is obtained by solving the linear relaxation of the formulation. Then, an iterative procedure starts: the half-cycle formulation is solved by a commercial solver with the upper bound imposed as value for the objective function, if an integer solution is found the procedure stops, otherwise the upper bound is decreased by one and the procedure repeats. Their results considered instances with up to 1000 incompatible pairs and cycles of length up to eight. All these formulations can be easily adapted to solve the KEP with altruistic donors and thus with chains. For example, Pansart *et al.* (2022) extended the cycle formulation

to include the chains, as well. The interested reader may refer to Mak-Hau (2015) for a survey on formulations for the KEP up to 2015.

For scalability reasons, several works focus on developing column generation approaches for the KEP. As already mentioned, Abraham *et al.* (2007) presented the first Branch-and-Price (BP) algorithm to solve the KEP with cycles only. The works of Glorie *et al.* (2012), Glorie *et al.* (2014), Klimentova *et al.* (2014) and Plaut *et al.* (2016a) developed BP algorithms to solve the KEP with both cycles and chains. However, Plaut *et al.* (2016b) proved that the algorithms of Glorie *et al.* (2012), Glorie *et al.* (2014) and Plaut *et al.* (2016a) were not correct. In addition, Klimentova *et al.* (2014) did not provide any computational experiments with KEP instances with chains. Based on the work of Klimentova *et al.* (2014), Lam & Mak-Hau (2020) built the first BPC algorithm for the KEP with cycles only. They tested the BPC algorithm on the 80 instances of the PrefLib dataset with 16 to 2048 pairs and without altruistic donors. The length of the cycles was limited to three or four. The BPC algorithm was able to solve all the instances when the length is limited to three and the majority of them when the length is limited to four. However, the considered valid inequalities are rather ineffective in strengthening the linear relaxation. Pansart *et al.* (2022) is the first column generation approach capable of dealing with both cycles and chains. The objective is to maximise the total medical benefit. The authors proposed a restricted master heuristic where the cycles are enumerated and the chains are detected via the solution of the pricing problem. Precisely, they proposed new heuristic algorithms to solve the pricing. The correctness of their algorithm is based on the solution of the pricing problem via an exact dynamic programming labelling algorithm. Pansart *et al.* (2022) presented results on instances with up to 1000 incompatible pairs, 111 altruistic donors, cycles of length up to three and chains with length up to 12. Precisely, their approach solved to optimality the majority of the instances with up to 250 pairs.

The first work proposing a column generation approach which prices variables associated with cycles and chains is Riascos-Álvarez *et al.* (2020). The authors developed a BP algorithm based on the disaggregated cycle formulation of Klimentova *et al.* (2014) where the pricing problem is tackled via *multi-valued decision diagrams* and the solution of linear and integer linear programs. The authors exploited the idea of Constantino *et al.* (2013) to represent each cycle by its vertex with the lowest index, that permits to apply a preprocessing procedure which reduces the subset of cycles to consider in the formulation of Klimentova *et al.* (2014). Consequently, the number of the pricing

subproblems associated with the cycles is heavily reduced. Their approach is tested on the instances of the PrefLib dataset with cycles and chains of maximal length equal to three or four and chains of maximal length from three to six. The large majority of the instances are solved to optimality at the root node. However, their results are outperformed by the BP algorithm of Arslan *et al.* (2022). Precisely, the BP algorithm of Arslan *et al.* (2022) builds on the one of Riascos-Álvarez *et al.* (2020) by identifying theoretical conditions where also the pricing subproblems associated with the chain variables can be solved in polynomial time. Precisely, if the maximal length of the chains is less than or equal to the maximal length of the cycles plus one, proving that no more positive reduced cost chains exist can be done by relaxing the elementarity constraints. The authors make use of variants of the Bellman-Ford algorithm to price both cycles and chains (when the elementarity constrains are relaxed) in polynomial time. However, to ensure the correctness of the algorithm, it might be necessary to solve an ESPPRC to prove that no more elementary positive reduced cost path exist. Arslan *et al.* (2022) do so by modelling the ESPPRC as an integer program with an exponential number of constraints and solve it by means of a branch-and-cut algorithm.

5.3 Problem formulation

Let $\mathcal{I}$ be the set of incompatible patient-donor pairs and let $\mathcal{D}$ be the set of altruistic donors. The KEP can be defined on a directed weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ referred to as *compatibility graph*. Vertex set $\mathcal{V} = \mathcal{I} \cup \mathcal{D}$ contains a vertex for each incompatible patient-donor pair of $\mathcal{I}$ and each altruistic donor of $\mathcal{D}$. The arcs in $\mathcal{A}$ model all the possible transplants between donors and patients. Hence, arc set $\mathcal{A}$ contains an arc (i, j) from each vertex $i \in \mathcal{V}$ to each patient-donor pair $j \in \mathcal{I}$, if the kidney of the donor associated with i is compatible with the patient of pair j. We assign a weight W_{ij} to each arc $(i, j) \in \mathcal{A}$ representing the utility (medical benefit) of the associated transplant. Cycles and chains of kidney exchanges between donors and patients are modelled in graph $\mathcal{G}$ by two types of subgraphs, namely circuits and paths. The length of a cycle or a chain is equal to the number of arcs in the circuit or path. In this respect, we call an *exchange circuit* an elementary circuit in graph $\mathcal{G}$ of length at most $L^C > 1$. Similarly, we call an *exchange path* an elementary path in graph $\mathcal{G}$ of length at most $L^P > 1$. Given that vertices associated with altruistic donors do not have

in-going arcs, exchange circuits are composed only of vertices associated with patient-donor pairs and exchange paths must start with a vertex associated with an altruistic donor. Given an exchange circuit or path e, the weight W_e of e is defined as the sum of the weights of the arcs traversed by e, i.e., $W_e \coloneqq \sum_{(i,j)\in\mathcal{A}(e)} W_{ij}$, where $\mathcal{A}(e)$ is the set of arcs traversed by e. Finally, we define an *exchange scheme* as a union of pairwise vertex-disjoint exchanges circuits and paths. The KEP aims to determine an exchange scheme of maximum weight, where the weight of an exchange scheme is the sum of the weights of the exchanges circuits and paths composing it. An exchange scheme may not contain all the vertices of $\mathcal{V}$. Remark that if all weights W_{ij}, $(i,j) \in \mathcal{A}$, are set to one, the objective of the KEP is to maximise the number of transplants. In what follows, with abuse of language, we refer to an exchange circuit or path with the term exchange.

We now report the set packing formulation proposed in Pansart *et al.* (2022) to model the KEP.

We denote by $\mathcal{E} = \mathcal{E}^C \cup \mathcal{E}^P$ the set of the exchanges in graph $\mathcal{G}$, where $\mathcal{E}^C$ ($\mathcal{E}^P$) denotes the set of the exchange circuits (paths) in graph $\mathcal{G}$. Let a_i^e be a binary parameter equal to one if vertex $i \in \mathcal{V}$ is involved in exchange $e \in \mathcal{E}$ and zero otherwise. For each exchange $e \in \mathcal{E}$, we define a binary variable λ_e taking value one if exchange e is part of the exchange scheme and zero otherwise.

The Set Packing formulation [SP] for the KEP reads as follows:

$$[\text{SP}] \quad \max \sum_{e\in\mathcal{E}} W_e \lambda_e \tag{5.1}$$

$$\text{s.t.} \sum_{e\in\mathcal{E}} a_i^e \lambda_e \leq 1 \qquad \forall i \in \mathcal{V} \tag{5.2}$$

$$\sum_{e\in\mathcal{E}} \lambda_e \leq \frac{|\mathcal{V}|}{2} \tag{5.3}$$

$$\lambda_e \in \{0,1\} \qquad \forall e \in \mathcal{E}. \tag{5.4}$$

Objective function (5.1) maximises the weight of the exchange scheme. Constraints (5.2) referred to as *packing constraints* ensure that each incompatible pair and altruistic donor is involved in at most one exchange circuit or path. Constraint (5.3) imposes an upper bound on the number of the exchanges in an exchange scheme. Remark that such constraint is redundant, indeed, each exchange circuit or path cannot be composed of less than two vertices. We include it in the formulation to compute a

valid upper bound in the column generation procedure, namely the Lagrangian bound. Such bound may be used as a termination condition in the column generation procedure (see Desrosiers & Lübbecke, 2005, for more details). Finally, Constraints (5.4) define variables λ_e as binary.

Valid inequalities to strengthen formulation [SP] will be presented in Section 5.4.4.

5.4 A Branch-Price-and-Cut algorithm

Formulation [SP] is an integer model defined over exponentially-many variables λ_e, $e \in \mathcal{E}$. We solve [SP] by means of an exact algorithm based on a Branch-Price-and-Cut (BPC) paradigm (Barnhart *et al.*, 1998). BPC algorithms are variants of the branch-and-bound algorithm where at each node of the branch-and-bound tree the linear relaxation of [SP] is solved via a column generation procedure (Desrosiers & Lübbecke, 2005). The linear relaxation of [SP] is commonly referred to as the Master Problem (MP). If the solution of the MP is fractional, a separation procedure may be called to identify violated valid inequalities (see Section 5.4.4). If such inequalities are found, they are included in the MP and the column generation procedure is repeated. Finally, when no valid inequalities are detected, branching rules are applied to ensure the correctness of the BPC algorithm.

The outline of this section is as follows. Section 5.4.1 is devoted to the column generation procedure we employ in our BPC algorithm. In Sections 5.4.2 and 5.4.3, we formulate the pricing problem and we present an exact procedure to solve it. In Section 5.4.4, we present the family of valid inequalities we consider in our separation procedure and the cut generation strategy, as well. Finally, the branching strategies are discussed in Section 5.4.5.

5.4.1 Column generation

The MP is solved by means of a column generation procedure at each node of the branch-and-bound tree. The procedure starts by solving the Restricted Master Problem (RMP), i.e., the MP restricted to a subset of λ variables. Each iteration of a column generation procedure comprises two consecutive steps: solve the RMP and solve the so-called *pricing problem*. When the RMP is a maximisation problem, the role of the pricing problem is to detect the most positive reduced cost variable (column), i.e., the variable which will increase the most the value of the RMP when included in it. When

the pricing problem fails to detect such variable, the current solution of the RMP is then proven to be optimal for the MP, as well.

5.4.2 Pricing problem formulation

Let $\pi_i \geq 0$, $i \in \mathcal{V}$ and $\beta \geq 0$ be the dual prices associated with Constraints (5.2) and (5.3), respectively. The reduced cost of a λ_e variable is:

$$\bar{W}_e = W_e - \sum_{i \in \mathcal{V}} a_i^e \pi_i - \beta.$$

The pricing problem reads as follows:

$$[\text{PP}] \ \max\{\bar{W}_e : e \in \mathcal{E}\}.$$

We observe that exchange set $\mathcal{E}$ can be partitioned in the following manner:

$$\mathcal{E} = \mathcal{E}^C \cup \mathcal{E}^P = \bigcup_{i \in \mathcal{I}} \mathcal{E}_i^C \cup \mathcal{E}^P,$$

where $\mathcal{E}_i^C$ is the set of the exchange circuits starting and ending at incompatible pair $i \in \mathcal{I}$. Hence, pricing problem [PP] can be decomposed in $|\mathcal{I}|+1$ independent subproblems:

$$\begin{aligned} [\text{PP-C}](i) \ & \max\{\bar{W}_e : e \in \mathcal{E}_i^C\}, && i \in \mathcal{I} \\ [\text{PP-P}] \ & \max\{\bar{W}_e : e \in \mathcal{E}^P\}. \end{aligned}$$

The aim of problem [PP-C](i) is to determine the most positive reduced cost variable λ_e, $e \in \mathcal{E}_i^C$, associated with exchange circuits starting and ending in $i \in \mathcal{I}$ or state that no positive reduced cost variable exists. Similarly, the aim of problem [PP-P] is to determine the most positive reduced cost exchange path variable λ_e, $e \in \mathcal{E}^P$ or state that no positive reduced cost variable exists.

In the following, we show that problems [PP-C](i), $i \in \mathcal{I}$, and [PP-P] share the same structure: they can be formulated as an *Elementary Longest Path Problem with Length Constraint* (ELPPLC), i.e., a variant of the *Elementary Shortest Path Problem with Resource Constraints* (ESPPRC).

First, we formally define the ELPPLC having as a reference Pansart *et al.* (2022). Let $\mathcal{H} = (\mathcal{N}, \mathcal{F})$ be a directed weighted graph, where the weights associated with

the arcs $(i,j) \in \mathcal{F}$ are denoted by $\bar{W}_{ij}$. The objective of the ELPPLC is to find an elementary path e^* in graph $\mathcal{H}$ starting at a given vertex $s \in \mathcal{N}$, ending at a given vertex $t \in \mathcal{N}$ and such that its weight $\bar{W}_{e^*} = \sum_{(i,j)\in\mathcal{A}(e^*)} \bar{W}_{ij}$ is maximum and its length is less than or equal to $\bar{L} > 0$. The topology of the graph $\mathcal{H}$ depends on the nature of the pricing subproblem, and we will describe it in more detail in the following.

Now, let $i \in \mathcal{I}$ be a pair. To model [PP-C](i) as an ELPPLC, we define a directed weighted graph $\mathcal{H}_i^C = (\mathcal{N}_i^C, \mathcal{F}_i^C)$ as follows. The vertices of $\mathcal{H}_i^C$ are the incompatible pairs of $\mathcal{I}$ plus a copy i' of pair i, i.e., $\mathcal{N}_i^C \coloneqq \mathcal{I} \cup \{i'\}$. The arcs of $\mathcal{H}_i^C$ are $\mathcal{F}_i^C = \{(h,j) \in \mathcal{A} : h,j \in \mathcal{I}, j \neq i\} \cup \{(j,i') : (j,i) \in \mathcal{A}\}$. Precisely, $\mathcal{F}_i^C$ contains the arcs of compatibility graph $\mathcal{G}$ between the incompatible pairs, except those entering in pair i and for all arcs $(j,i) \in \mathcal{A}$ entering in i, we add an arc (j,i') entering in the copy i' of i. Remark that circuits in graph $\mathcal{G}$ starting and ending in i correspond to paths in graph $\mathcal{H}_i^C$ starting in i and ending in i'. Weights $\bar{W}_{hj}$ assigned to arcs $(h,j) \in \mathcal{F}$ are defined as follows

$$\bar{W}_{hj} = \begin{cases} W_{hj} - \pi_j - \beta, & \text{if } h = i \\ W_{hi} - \pi_i, & \text{if } j = i' \\ W_{hj} - \pi_j, & \text{otherwise .} \end{cases}$$

To consider the path length, we define a resource consumption $L_{hj} = 1$ for each arc $(h,j) \in \mathcal{F}$. Solving problem [PP-C](i) consists in finding the most positive reduced cost elementary path starting at i and ending at i' in graph $\mathcal{H}_i^C$ such that the length of the path is less than or equal to L^C.

We model [PP-P] as an ELPPLC on graph $\mathcal{H}^P = (\mathcal{N}^P, \mathcal{F}^P)$ defined as compatibility graph $\mathcal{G}$ augmented with source vertex s and an arc from s to each altruistic donor of $\mathcal{D}$. Precisely, we set $\mathcal{N}^P \coloneqq \{s\} \cup \mathcal{V}$ and $\mathcal{F}^P = \mathcal{A} \cup \{(s,i) : i \in \mathcal{D}\}$. Given an arc $(i,j) \in \mathcal{F}^P$, its weight $\bar{W}_{ij}$ is defined as follows

$$\bar{W}_{ij} = \begin{cases} W_{ij} - \pi_j - \beta, & \text{if } i = s \\ W_{ij} - \pi_j, & \text{otherwise.} \end{cases}$$

Resource consumption L_{ij} is set to be one on all arcs $(i,j) \in \mathcal{F}^P$. Problem [PP-P] consists in finding the most positive reduced cost elementary path starting at $s \in \mathcal{N}$ and ending in a vertex $j \in \mathcal{N}^P \setminus \{s\}$ in graph $\mathcal{H}^P$ such that the length of the path is less than or equal to $L^P + 1$. Remark that to be consistent with the definition of exchange paths given in Section 5.3, here, [PP-P] looks for paths of length at most

$L^P + 1$. Indeed, in the underlying graph $\mathcal{H}^P$ used to solve [PP-P], we add a source vertex s which is not included in the compatibility graph $\mathcal{G}$.

In the following, we report two theoretical results which identify conditions under which subproblems [PP-C](i), $i \in \mathcal{I}$, and [PP-P] can be formulated as *Longest Path Problems with Length Constraint* (LPPLC). The first result states that the subproblems to price the circuits reduce to an LPPLC. Remark that a non-elementary circuit is composed of at least two elementary circuits. Hence, if the LPPLC returns a positive reduced cost non-elementary circuit when solving problem [PP-C](i), $i \in \mathcal{I}$, then at least one of the elementary circuits composing it must have a positive reduced cost. Remark that such circuit might not be a solution of [PP-C](i), but of another subproblem. The LPPLC always identifies a positive reduced cost elementary circuit even if it may not be the most positive one.

The second result is an original result of Arslan *et al.* (2022) and it identifies a subclass of the problem instances where it is possible to solve problem [PP-P] by dropping the elementarity requirement of the paths, i.e., [PP-P] reduces to a LPPLC, as well. Precisely, if $L^P \leq L^C + 1$ and problems [PP-C](i), $i \in \mathcal{I}$, do not identify any positive reduced cost circuit, then problem [PP-P] reduces to a LPPLC. Indeed, a non-elementary path of length l contains a sub-tour of length at most $l - 1$ which is composed only of pairs. Hence, if problems [PP-C](i), $i \in \mathcal{I}$, do not identify any circuit with a reduced cost larger than $-\beta$ (of length at most L^C), then [PP-P] will not provide any positive reduced cost path which is non-elementary.

Remark that if $L^P > L^C + 1$, solving the LPPLC on graph $\mathcal{H}^P$ provides an upper bound on the value of [PP-P] since the LPPLC is a relaxation of the ELPPLC.

5.4.3 Pricing problem solution

In this section, we present an exact procedure similar to the one of Arslan *et al.* (2022) to solve the pricing problem.

The pseudocode of such a procedure is presented in Algorithm 9. It can be noticed that an ELPPLC needs to be solved for [PP-P] only if (i) the maximal length of the chains is strictly larger than the maximal length of the cycles plus one, i.e., $L^P > L^C+1$; (ii) no positive reduced cost circuits is detected, i.e, [PP-C](i) $W_i^* = \max\{\bar{W}_e : e \in \mathcal{E}_i^C\} \leq 0$, for all $i \in \mathcal{I}$; (iii) the LPPLC version of the [PP-P] provides no positive reduced cost elementary path, but provides a positive reduced cost non-elementary path of length greater than or equal to $L^C + 2$. Indeed, if the LPPLC do not find any

positive reduced cost non-elementary path, then [PP-P] does not admit any positive reduced cost elementary path, as well. In addition, if some positive reduced cost elementary paths are found when solving the LPPLC, these can be included in the RMP without the need to solve [PP-P] with an ELPPLC.

Finally, the following proposition provides a correctness result for the column generation procedure shown in Algorithm 9.

Proposition 5.1. *The procedure of Algorithm 9 solves the MP to optimality at each node of the branch-and-bound tree.*

Algorithm 9: Column generation procedure.

```
do
    solve the RMP;
    get the optimal solution of the dual of the RMP;
    build pricing subproblems [PP-C](i), i ∈ ℐ and [PP-P];
    solve [PP-C](i) and let W̄*_i be its optimal value, for all i ∈ ℐ;
    let W̄*^C = max{W̄*_i : i ∈ ℐ};
    Solve the LPPLC on graph ℋ^P and let W̄*^P be its optimal value;
    if L^P > L^C + 1 then
        if W̄*^C ≤ −β and W̄*^P > 0 and solving the LPPLC on graph ℋ^P do not provide any positive
          reduced cost elementary path and it provides a positive reduced cost non-elementary path of
          length l ≥ L^C + 2 then
            solve the ELPPLC on graph ℋ^P and let W̄*^P be its optimal value;
        end
    end
    add positive reduced cost elementary circuits and paths to the RMP, if any is detected;
while max{W̄*^C, W̄*^P} > 0;
```

In the following, we describe the algorithms to solve: (i) the LPPLC to provide solutions for subproblems [PP-C](i) and [PP-P], and (ii) the ELPPLC to provide optimal solutions for [PP-P].

First, we remark that although the SPPRC is NP-hard (Di Puglia Pugliese & Guerriero, 2013) for a generic resource constraint, it becomes solvable in polynomial time if the resource is the length of the path. Indeed, the SPPRC on a graph $\mathcal{H} = (\mathcal{N}, \mathcal{F})$ can be solved by pseudo-polynomial algorithms with a temporal complexity of $O(\bar{R}|\mathcal{F}|)$ (Desrochers, 1988) where $\bar{R}$ is the maximal amount of resource.

In the context of a LPPLC, the maximal amount of resource $\bar{R}$ is the maximal length of the path $\bar{L}$, which is bounded from above by the number of vertices in the graph $|\mathcal{N}|$. Hence, the LPPLC can be solved with a polynomial temporal complexity of $O(|\mathcal{N}||\mathcal{F}|)$.

We report a polynomial time procedure based on the Bellman-Ford algorithm to solve the LPPLC. Precisely, such algorithm corresponds to the first $\bar{L}$ iterations of the

Bellman-Ford algorithm, and is detailed in Algorithm 10, where we make use of the generic notation introduced in Section 5.4.2. In addition, we denote by W_i^l the weight of a path of length $l = 0, \ldots, \bar{L}$ starting in s and ending in vertex $i \in \mathcal{N}$. Matrix $(p_i^l)_{l=0,\ldots,\bar{L}, i \in \mathcal{N}}$ stores the predecessor p_i^l of vertex $i \in \mathcal{N}$ a path of length l starting in s and ending in vertex i. We write $\mathcal{P}$ for the set of the positive reduced cost paths found by the algorithm.

Algorithm 10: A Bellman-Ford algorithm for the LPPLC

Input: Graph $\mathcal{H} = (\mathcal{N}, \mathcal{F})$.
Initialization: $W_s^0 := 0$ and $p_s^0 := s$, $W_i^l := -\infty$ and $p_i^l = null$ for all $i \in \mathcal{N} \setminus \{s\}$ and $l = 0, \ldots, \bar{L}$ and $\mathcal{P} = \emptyset$.
forall $l = 1, \ldots, \bar{L}$ **do**
  forall $i \in \mathcal{N}$ **do**
    forall $(i, j) \in \delta^+(i)$ **do**
      if $W_i^{l-1} + \bar{W}_{ij} > W_j^l$ **then**
        set $W_j^l := W_i^{l-1} + \bar{W}_{ij}$;
        set $p_j^l = i$;
      end
    end
  end
end
// retrieve positive reduced cost paths
forall $l = 1, \ldots, \bar{L}$ **do**
  forall $i \in \mathcal{N}$ **do**
    if $W_i^l > 0$ **then**
      apply backtracking to p_i^l to build the path;
      add the path to $\mathcal{P}$;
    end
  end
end
return $\mathcal{P}$;

To complete this section, we describe a label correcting dynamic programming algorithm (Feillet *et al.*, 2004) to solve the ELPPLC. In the work of Arslan *et al.* (2022), the ELPPLC is tackled by a mixed integer programming formulation with an exponential number of constraints. Here, we chose a label correcting dynamic programming algorithm for its efficiency and for its flexibility when it comes to incorporate non-robust valid inequalities to strengthen the formulation (see Section 5.4.4).

In a label correcting algorithm, vertices are repeatedly processed and their associated labels, which identify paths in the graph, are extended. Precisely, a label $l = (L, \bar{W}, \mathcal{U}, i)$ represents a path in graph $\mathcal{H}$ starting at vertex s and ending in vertex $i \in \mathcal{N}$ characterised by its accumulated weight $\bar{W}$, its length L and the subset of visited vertices $\mathcal{U} \subseteq \mathcal{N}$. A label $l = (L, \bar{W}, \mathcal{U}, i)$ associated with vertex i can be extended along arc $(i, j) \in \mathcal{F}$ if the length constraint $(L < \bar{L})$ and the elementarity constraint $(j \notin \mathcal{U})$ are respected. If this is the case, a new label $l' = (L+1, \bar{W} + W_{ij}, \mathcal{U} \cup \{j\}, j)$ associated

with vertex $j \in \mathcal{V}$ is obtained. Dominance rules are commonly applied in labelling algorithms to prune labels which would not lead to any optimal solution. Given two labels $l = (L, \bar{W}, \mathcal{U}, i)$ and $l' = (L', \bar{W}', \mathcal{U}', i)$ associated with the same vertex i, we say that l dominates l' if:

$$\begin{cases} L \leq L' \\ \bar{W} \geq \bar{W}' \\ \mathcal{U} \subseteq \mathcal{U}' \end{cases} \tag{5.5}$$

and one of the inequalities (inclusion) is strict.

The elementarity constraint makes the ELPPLC strongly NP-hard and slows down the solution algorithm. Adopting the *ng-path* relaxation (Baldacci *et al.*, 2011), which partially relaxes the elementarity constraint of the paths, helps in accelerating the solution of the ELPPLC. Each vertex $i \in \mathcal{N}$ is assigned with a neighbourhood $\mathcal{U}_i$ of a given size. Circuits along a path are then allowed only if each vertex visited more than once does not belong to the neighbourhoods of its predecessors in the path. The label definition and extension rule are modified accordingly. Precisely, the set of visited customers $\mathcal{U}$ becomes the *memory* of the label. The extension of $l = (L, \bar{W}, \mathcal{U}, i)$ along arc (i, j) is $l' = (L + 1, \bar{W} + W_{ij}, (\mathcal{U} \cap \mathcal{U}_j) \cup \{j\}, j)$.

5.4.4 Cut generation

We consider two families of valid inequalities, namely, the subset-row (SR) inequalities (Jepsen *et al.*, 2008) and the odd-hole (OH) inequalities (Padberg, 1973). Both families of valid inequalities are *non-robust*, that is, considering their dual variables in the pricing problem solution modifies the structure of the pricing problem itself. Regarding the OH inequalities, we separate them by the procedure of Hoffman & Padberg (1993), however, we insert them in the RMP as subset-row inequalities, as presented in Section 5.4.4.2.

5.4.4.1 Subset-row inequalities

Given a subset $\mathcal{S} \subseteq \mathcal{V}$ and a multiplier p_i for each $i \in \mathcal{S}$, SR inequalities are obtained as a Chvátal-Gomory rounding of the Constraints (5.2) associated with elements in $\mathcal{S}$:

$$\sum_{e \in \mathcal{E}} \left\lfloor \sum_{i \in \mathcal{S}} p_i a_i^e \right\rfloor \lambda_e \leq \left\lfloor \sum_{i \in \mathcal{S}} p_i \right\rfloor, \qquad \mathcal{S} \subseteq \mathcal{V}. \tag{5.6}$$

We restrict ourselves to consider only subsets $\mathcal{S}$ of cardinality three and multipliers $p_i = 1/2$ for each $i \in \mathcal{S}$. Hence, the separation of these inequalities is performed by enumeration. Let $\sigma_{\mathcal{S}} \geq 0$ be the dual variable associated with SR inequality (5.6) defined for subset $\mathcal{S} \subseteq \mathcal{V}$. To consider such dual variable in the ELPPLC, we follow the procedure proposed in Pecin *et al.* (2017). We choose to do so because Pecin *et al.* (2017) provides a general management rule of the SR inequalities which is valid also in the case where the elementarity of the paths is relaxed. In the label definition, we include a state for each SR inequality in the RMP whose dual variable is different from zero. The state of a SR inequality identified by $\mathcal{S} \subseteq \mathcal{V}$ is a parameter $M(\mathcal{S})$ which records the visits of a path to vertices in $\mathcal{S}$ and establishes when dual variable $\sigma_{\mathcal{S}}$ has to be discounted from the reduced cost of the path. Precisely, parameter $M(\mathcal{S})$ is initialized to 0 and every time a path visits a vertex $i \in \mathcal{S}$, $M(\mathcal{S})$ is incremented by $p_i = 1/2$. If $M(\mathcal{S}) \geq 1$, dual variable $\sigma_{\mathcal{S}}$ is discounted from the reduced cost and $M(\mathcal{S})$ is decremented by one unit.

The second condition in dominance rule (5.5) has to be replaced by

$$\bar{W} \geq \bar{W}' + \sum_{\substack{\mathcal{S} \in \mathcal{M}: \\ M(\mathcal{S}) > M'(\mathcal{S})}} \sigma_{\mathcal{S}},$$

where $\mathcal{M}$ is the set of subsets $\mathcal{S} \subseteq \mathcal{V}$ representing SR inequalities in the RMP whose dual variable $\sigma_{\mathcal{S}}$ is different from zero.

5.4.4.2 Odd-hole inequalities

Given an exchange $e \in \mathcal{E}$, we denote by $\mathcal{V}(e)$ the set of vertices in exchange e. We say that two exchanges $e, e' \in \mathcal{E}$ are *in conflict* if they share common vertices, i.e., if $\mathcal{V}(e) \cap \mathcal{V}(e') \neq \emptyset$. Variables associated with exchanges in conflict between each other cannot both be one in an integer solution. Given a fractional solution $\tilde{\lambda}$, we define the so-called *conflict graph* $\mathcal{C} = (\mathcal{Q}, \mathcal{T})$ as follows. Node set $\mathcal{Q}$ contains the exchanges $e \in \mathcal{E}$ whose corresponding variable appear in the fractional solution with value $\tilde{\lambda}_e \in (0, 1)$. Edge set $\mathcal{T}$ contains an edge (e, e') if exchanges $e \in \mathcal{E}$ and $e' \in \mathcal{E}$ are in conflict. Odd-hole inequalities correspond to particular subgraphs in graph $\mathcal{C}$, namely to odd cycles without chords. An *odd cycle without chords* is a set of nodes $\mathcal{P} = \{e_1, e_2, \ldots, e_{2k+1}\}$, $k \geq 2$, such that edge set $\mathcal{T}$ contains edges $t_i = (e_i, e_{i+1})$, $i = 1, \ldots, 2k$, and $t_{2k+1} = (e_{2k+1}, e_1)$ and no other edge incident only in nodes of $\mathcal{P}$.

The odd-hole inequality associated with $\mathcal{P}$ is:

$$\sum_{e \in \mathcal{P}} \lambda_e \leq \frac{|\mathcal{P}| - 1}{2}. \tag{5.7}$$

The separation of these inequalities is done through the exact procedure of Hoffman & Padberg (1993). Such procedure considers each vertex of the conflict graph and determines all the odd-hole inequalities associated to odd cycles without chords starting at the considered vertex. This procedure is rather time consuming. Therefore, as suggested by Hoffman & Padberg (1993), when determining the inequalities, we randomly consider the 30% of the vertices of the conflict graph as starting vertex.

OH inequalities are defined over a subset of variables $\mathcal{P}$, hence, the management of the associated dual variable $\theta_{\mathcal{P}} \geq 0$ when solving the pricing problem is not tractable. Indeed, when solving the pricing problem, we need to identify circuits and paths associated with the variables in $\mathcal{P}$ to discount dual variable $\theta_{\mathcal{P}}$ from the reduced cost.

For this reason, we show how it is possible to retrieve a SR inequality from an OH inequality defined on cycle $\mathcal{P}$. Moreover such SR inequality is a lifted version of the associated OH inequality. For each edge $t_i = (e_i, e_{i+1}) \in \mathcal{T}$, $i = 1, \ldots, 2k$ and $t_{2k+1} = (e_{2k+1}, e_1) \in \mathcal{T}$, of the odd cycle $\mathcal{P}$, we select elements $v_i \in \mathcal{V}(e_i) \cap \mathcal{V}(e_{i+1})$ and $v_{2k+1} \in \mathcal{V}(e_{2k+1}) \cap \mathcal{V}(e_1)$, which are vertices in compatibility graph $\mathcal{G}$. Remark that in the odd cycle there are no chords, hence, all the v_i are different from each other. The associated SR inequality is defined over $\mathcal{S} = \{v_1, \ldots, v_{2k+1}\}$ and multipliers $p_i = 1/2$ for each $i \in \mathcal{S}$:

$$\sum_{e \in \mathcal{E}} \left\lfloor \frac{1}{2} \sum_{i \in \mathcal{S}} a_i^e \right\rfloor \lambda_e \leq \left\lfloor \frac{|\mathcal{S}|}{2} \right\rfloor. \tag{5.8}$$

Lemma 5.2. *If a fractional solution $\tilde{\lambda}$ violates OH inequality (5.7) then it violates also SR inequality (5.8) associated with inequality (5.7).*

Proof. First, we observe that the right hand-side of inequality (5.8) coincides with the one of inequality (5.7), indeed, it holds $|\mathcal{S}| = |\mathcal{P}|$ by construction and $|\mathcal{P}|$ is odd. Then, in the left hand-side of inequality (5.8), at least the variables associated with exchanges in $\mathcal{P}$ appear with coefficient one. Indeed, such exchanges visit exactly two vertices in $\mathcal{S}$. Finally, it holds

$$\sum_{e \in \mathcal{E}} \left\lfloor \frac{1}{2} \sum_{i \in \mathcal{S}} a_i^e \right\rfloor \tilde{\lambda}_e \geq \sum_{e \in \mathcal{P}} \left\lfloor \frac{1}{2} \sum_{i \in \mathcal{S}} a_i^e \right\rfloor \tilde{\lambda}_e = \sum_{e \in \mathcal{P}} \tilde{\lambda}_e > \frac{|\mathcal{P}| - 1}{2} = \left\lfloor \frac{|\mathcal{S}|}{2} \right\rfloor \tag{5.9}$$

and $\tilde{\lambda}$ violates inequality (5.8). □

Note that Inequality (5.8) is a lifted version of Inequality (5.7). Indeed, the left hand-side of (5.8) contains additional variables in $\mathcal{E} \setminus \mathcal{P}$ with positive coefficients.

The impact on the pricing problem is the same as the one of the SR inequalities.

5.4.4.3 Cut generation strategy

We adopt the following cut generation strategy. We separate at most 100 inequalities in the branch-and-bound tree. We allow ten minutes for the overall time spent in the separation procedures, beyond such threshold, we do not separate any inequality. At each separation round, we separate OH and SR inequalities in this order, and when 20 violated inequalities have been detected, we stop looking for other inequalities in that round. A time limit of one minute is imposed for the separation of each of the two families of inequalities at each round. In addition, if during one round no OH inequalities are detected, we keep separating only SR inequalities at that node.

Remark that if non-robust valid inequalities are active in the RMP, the circuits are priced by a labelling algorithm instead of the variant of the Bellman-Ford algorithm presented in Section 5.4.3. Indeed, dealing with non-robust inequalities makes the pricing problem more complicated as it requires more resources, one per each non-robust inequality. In that context, labelling algorithms are well-suited for the management of these additional resources.

5.4.5 Branching scheme

To ensure the integrality of the solution returned by the BPC algorithm, we branch on the use of the arcs in $\mathcal{A}$. Let $\tilde{\lambda}$ be an optimal fractional solution of the MP at a node of the branch-and-bound tree. We branch on values $f_{ij} = \sum_{e \in \mathcal{E}} b_{ij}^e \tilde{\lambda}_e$, $(i,j) \in \mathcal{A}$, where parameter b_{ij}^e takes value one if arc $(i,j) \in \mathcal{A}$ belongs to exchange $e \in \mathcal{E}$ and zero otherwise. Precisely, we select the branching candidate whose value is the closest to 0.5. Remark that $f_{ij} \in (0,1)$, hence, in one branch we force arc (i,j) to be used ($\sum_{e \in \mathcal{E}} b_{ij}^e \lambda_e = 1$) while in the other branch, we forbid its use ($\sum_{e \in \mathcal{E}} b_{ij}^e \lambda_e = 0$). In the former case, the dual variable associated with the branching constraint is considered when solving the pricing problem. In this latter case, arc (i,j) is removed from graph $\mathcal{H}$ when we solve the pricing problem. Last, the branch-and-bound tree is explored according to the best-first strategy.

5.4.6 Accelerating techniques

We make use of the following accelerating techniques to speed up our BPC algorithm:

Preprocessing of Pansart *et al.* (2022). Imposing length constraints on cycles and chains allows us to perform a preprocessing procedure to reduce the size of compatibility graph $\mathcal{G}$. Such procedure was introduced by Pansart *et al.* (2022) and is based on the Floyd-Warshall algorithm to compute the shortest path between each pair of vertices in a graph. Precisely, we consider a graph $\tilde{\mathcal{G}} = (\{s\} \cup \mathcal{V}, \tilde{A})$, where $\tilde{\mathcal{A}} = \mathcal{A} \cup \{(s, i) : i \in \mathcal{D}\}$. Graph $\tilde{\mathcal{G}}$ is defined as graph $\mathcal{H}^P$ used to price out variables associated with exchange paths. The weights on the arcs are set to one. Then, the Floyd-Warshall algorithm is applied to $\tilde{\mathcal{G}}$ to detect the minimum length $l^*(i, j)$ paths between each pair of vertices $i, j \in \{s\} \cup \mathcal{V}$. We remove all arcs $(i, j) \in \mathcal{A}$ that satisfy the following two conditions:

1. the length of the shortest path from source s to vertex i is larger than or equal to the maximal chain length, i.e., $l^*(s, i) \geq L^P + 1$;
2. the length of the shortest path from vertex j to vertex i is larger than or equal to the maximal cycle length, i.e., $l^*(j, i) \geq L^C$.

Finally, isolated vertices are removed from $\mathcal{V}$ as well.

Preprocessing of Riascos-Álvarez *et al.* (2020). Riascos-Álvarez *et al.* (2020) proposed a procedure to reduce the number of pricing subproblems [PP-C](i), $i \in \mathcal{I}$, to be processed in the pricing problem solution. In addition, the same procedure allows to reduce the size of pricing graphs $\mathcal{H}_i^C$. As in Constantino *et al.* (2013) and Klimentova *et al.* (2014), Riascos-Álvarez *et al.* (2020) exploits the symmetry of the circuits subproblems: a circuit starting at vertex i considered in subproblem [PP-C](i) and containing vertex j will also appear as a circuit starting at vertex j when solving [PP-C](j). The procedure reads as follows. In the following, we work on a copy of compatibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ where we progressively remove vertices and arcs. For the ease of readability, we keep notation $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ to refer to this copy. We write $l^*(i, j)$ for the value of the shortest path, in terms of length, from i to j in compatibility graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$. First, the incompatible pairs with no in-going or no out-going arcs are removed from $\mathcal{I}$. Indeed, these pairs cannot be part of any circuit. The remaining ones are sorted by decreasing number of incident arcs. Then, the first pair $i \in \mathcal{I}$ is selected and

graph $\mathcal{H}_i^C = (\mathcal{N}_i^C, \mathcal{F}_i^C)$ is built as follows. We include in $\mathcal{N}_i^C$ all pairs $j \in \mathcal{I}$ whose distance from i is less than or equal to the cycle length, i.e., $l^*(i,j)+l^*(j,i) \leq L^C$. Once the vertex set is built, arc set $\mathcal{F}_i^C$ contains all the arcs $(i',j') \in \mathcal{A}$ whose extremes are in $\mathcal{N}_i^C$ and such that $l^*(i,i') + l^*(j',i) + 1 \leq L^C$. Finally, vertex i is removed from graph $\mathcal{G}$ along with its incident arcs. Set $\mathcal{I}$ is re-ordered according to the reduced graph $\mathcal{G}$ and a new iteration starts. The procedure terminates when either all the pairs in $\mathcal{I}$ are processed or there is no pair with both in-going or out-going arcs. Those reduced graphs $\mathcal{H}_i^C$ are then used to price the variables associated to the circuits.

Initialization of set $\mathcal{E}$. It is well known that column generation suffers from degeneracy which may slow down its convergence (Desrosiers & Lübbecke, 2005). A good initialisation of the set of columns helps in reducing this inconvenience. Hence, we generate at most 30000 exchange circuits and paths of length up to three and insert them in set $\mathcal{E}$. Precisely, to ensure diversity, we generate $30000/|\mathcal{V}|$ exchange circuits/paths starting at each incompatible pair/altruistic donor in the compatibility graph.

Primal heuristic. Embedding primal heuristics in a BPC algorithm helps in reducing the integrality gap and, consequently, the size of the branch-and-bound tree (Archetti *et al.*, 2013). We adopt the so-called *restricted master heuristic*, i.e., we solve formulation [SP] restricted to the subset of variables generated so far by means of a commercial solver. We call such heuristic after the root node is solved and every time 500 new columns are inserted in the RMP. We impose a time limit of 60 seconds for the first call and of 10 seconds for the other ones.

Tabu list. As in Arslan *et al.* (2022), when a pricing subproblem [PP-C](i), $i \in \mathcal{I}$, does not provide any positive reduced cost circuit, we insert such subproblem in a tabu list. All subproblems in the tabu list are not solved in the following column generation iterations. They are solved only when subproblem [PP-P] does not provide any positive reduced cost path to ensure the correctness of the column generation procedure.

5.5 Computational experiments

Our BPC algorithm is implemented in C++ and compiled in release mode under a 64-bit version of MS Visual Studio 2019. The linear programming models in the column generation procedure and the integer programming models in the restricted master heuristic are solved by GUROBI 9.5.2 (64-bit version). All the experiments are run on a 64-bit Windows machine equipped with a Intel(R) Xeon(R) Silver 4214 processor with 24 cores hyper-threaded to 48 virtual cores, with a base clock frequency of 2.2 GHz, and 96 GB of RAM. For each run of the algorithm, we impose one hour time limit and allow a single thread.

In this section, first, we present the characteristics of the benchmark instances we consider. Then, we compare the results obtained with the BPC algorithm on the benchmark instances with those reported in Pansart *et al.* (2022), Arslan *et al.* (2022) and Delorme *et al.* (2023). Note that these authors reported computational results on different datasets. We assess the impact of the valid inequalities considered in Section 5.5.5. Finally, in Section 5.5.6, we assess the impact of the length constraints on the objective function value.

5.5.1 Benchmark instances

We assess the efficiency of the BPC algorithm on three different sets of instances considered in the literature: the "Kidney Data (00036)" set from the *PrefLib* dataset (Mattei & Walsh, 2013), the set proposed by Pansart *et al.* (2022) and the one considered by Delorme *et al.* (2023). The instances belonging to the first two sets are produced by the so-called *Saidman generator* (Saidman *et al.*, 2006), while the ones belonging to the third set are produced by the generator introduced by Delorme *et al.* (2022). Both generators take as input the number of incompatible patient-donor pairs ($|\mathcal{I}|$), the number of altruistic donors ($|\mathcal{D}|$) and several medical related parameters to build a realistic compatibility graph $\mathcal{G}$. The medical related parameters influence the density of the graph and the medical benefit associated with the arcs of the graph. Note that default values for such parameters are suggested in the generator of Delorme *et al.* (2022). The interested reader may refer to Saidman *et al.* (2006) and Delorme *et al.* (2022) for more information.

Table 5.1 summarises the characteristics of the instances in the three sets. The rows of the tables are associated with the sets of instance. The column headings are

as follows: *set*: name of the set; #: total number of instances in the set; $|\mathcal{I}|$: number of incompatible patient-donor pairs in the set; $|\mathcal{D}|$: percentage of altruistic donors w.r.t. the number of pairs; *avg. density of* $\mathcal{G}$: average density of compatibility graph $\mathcal{G}$ expressed as a percentage of arcs w.r.t. the number of arcs in a complete graph with the same number of vertices as $\mathcal{G}$; L^C: maximum length of the cycles; L^P: maximum length of the chains; *obj.*: type of objective function, either maximisation of the number of transplants ($\#TR$) or of the medical benefit (MB).

Table 5.1: Characteristics of the sets of instances.

set	#	Characteristics					
		$\lvert\mathcal{I}\rvert$	$\lvert\mathcal{D}\rvert$	avg. density of $\mathcal{G}$	L^C	L^P	obj.
PrefLib	2000	16, 32, 64, 128, 256, 512, 1024, 2048	0%, 5%, 10%, 15%	25%	3, 4	0, 3, 4, 5, 6	#TR
Pansart *et al.* (2022)	270	50, 100, 250, 500, 750, 1000	0%, 5%, 10%, 15%	5%	3	3, 6, 12	MB
Delorme *et al.* (2023)	840	50, 100, 200, 400, 600, 800, 1000	0%	10%	3, 4, 5, 6, 7, 8	0	#TR, MB

5.5.2 Results on the whole testbed

In this section, we present the aggregated results obtained by the BPC algorithm on the three sets of instances. We compare our results with those reported in the literature.

Throughout the section the percentage optimality gap is computed as $((UB - LB)/UB)100$, where UB and LB are the upper and lower bounds returned by the BPC algorithm.

5.5.2.1 Results on the PrefLib dataset

In this section, we compare the results obtained by the BPC algorithm on the instances of the PrefLib dataset with those obtained by the BP algorithm of Arslan *et al.* (2022).

Table 5.2 reports the results on the PrefLib dataset. The rows of the table group instances with given maximal path lengths and given numbers of incompatible pairs. Note that the first row groups all small size instances with a maximum of 512 pairs, while the results for large instances with 1024 or 2048 pairs are reported in the other rows. The first three columns report some information about the instance subset. The

next three columns summarise the results obtained by the BPC algorithm: number of instances solved to optimality (*#opt.*), average number of nodes of the branch-and-bound tree (*avg.#nodes*) and average computational time (*avg.t[s]*). The last column reports the computational time of the BP algorithm of Arslan *et al.* (2022) (*avg.t[s]*). The authors implemented their BP algorithm using Julia language and ran the experiments on a 2 Dodeca-core Haswell Intel Xeon E5-2680 v3 2.5 GHz machine with 128Go RAM running Linux OS. In this column, we make use of character '-' when no result is available: Arslan *et al.* (2022) does not test the BP algorithm on instances where only circuits are to be priced, i.e., those with $L^P = 0$. No further information is required about their algorithm, since, it manages to solve to optimality all the instances at the root node.

Table 5.2: Results on the PrefLib dataset.

Instances			BPC			Arslan *et al.* (2022)
L^P	$\|\mathcal{J}\|$	#	#opt.	avg.#nodes	avg.t[s]	avg.t[s]
0, 3, 4, 5, 6	16, 32, 64, 128, 256, 512	1480	1480	1.00	1.13	0.99
0	1024	20	20	1.00	6.14	-
	2048	20	20	1.00	53.61	-
3	1024	60	60	1.00	9.82	3.75
	2048	60	60	1.02	148.94	11.93
4	1024	60	60	1.00	10.66	3.70
	2048	60	60	1.00	148.28	12.46
5	1024	60	60	1.00	11.00	3.72
	2048	60	60	1.00	163.91	12.56
6	1024	60	60	1.00	11.05	3.72
	2048	60	60	1.00	149.17	12.46

From the results of Table 5.2, it can be observed that both approaches yield comparable results on instances with up to 1024 pairs, regardless of the maximal length of the paths. Indeed, both algorithms solve those instances to optimality at the root node within computational times of the same order of magnitude. Even though all the instances characterised by a number of pairs equal to 2048 are solved to optimality by both approaches, the BP algorithm of Arslan *et al.* (2022) is faster than the BPC algorithm by an order of magnitude. The majority of the computational time of the BPC algorithm is spent during the column generation procedure. Although the same

pricing procedure is applied, the implementation of Arslan *et al.* (2022) is probably more efficient than the one of the BPC algorithm.

5.5.3 Results on the set of instances of Pansart *et al.* (2022)

In this section, we discuss the results obtained by the BPC algorithm on the instances introduced by Pansart *et al.* (2022). We compare the BPC algorithm performances with the restricted master heuristic of Pansart *et al.* (2022) on the subset of small instances characterised by a number of pairs up to 250. We recall that the approach of Pansart *et al.* (2022) is a heuristic with a performance guarantee, that is, it provides valid lower and upper bounds on the optimal value of formulation [SP]. The restricted master heuristic is run on a machine equipped with an Intel Xeon E5-2440 v2 @ 1.9 GHz processor and 32 GB of RAM. Note that on instances with more than 500 pairs, we do not compare with the results of Pansart *et al.* (2022) since it is clear that the BPC algorithm provides better results. Moreover, Pansart *et al.* (2022) do not test their procedure on all these instances.

Table 5.3 summarises the results of the comparison between the BPC algorithm and the restricted master heuristic of Pansart *et al.* (2022). The rows of the table group instances with the same maximal path length and the same number of incompatible pairs. The first three columns report some information about the instance subset: maximal path lengths (L^P), number of incompatible patient-donor pairs ($|\mathcal{I}|$) and number of instances ($\#$). The next three columns show the results obtained by the BPC algorithm: number of instances solved to optimality ($\#opt.$), average number of nodes of the branch-and-bound tree ($avg.\#nodes$) and average computational time ($avg.t[s]$). The last three columns report some statistic related to the algorithm of Pansart *et al.* (2022): number of instances solved to optimality ($\#opt.$), average computational time ($avg.t[s]$) and optimality gap expressed as a percentage for the instances not solved to optimality ($avg.gap[\%]$ $noOpt.$). If all the instances in a row are solved to optimality by the algorithm of Pansart *et al.* (2022), we make use of character '-' in column $avg.gap[\%]$ $noOpt.$.

Table 5.3: Results on the set of instances of Pansart *et al.* (2022) with up to 250 pairs.

Instances			BPC			Pansart *et al.* (2022)			
L^P	$\|\mathcal{J}\|$	#	#opt.	avg.#nodes	avg.t[s]	#opt.	avg.t[s]	avg.gap[%]	noOpt.
	50	15	15	1.00	0.03	15	1.07	-	
3	100	15	15	1.00	0.14	9	1.21	0.33	
	250	15	15	1.00	1.03	7	3.45	0.07	
	50	15	15	1.27	0.04	14	1.20	0.05	
6	100	15	15	2.07	0.33	8	2.17	0.64	
	250	15	15	40.27	93.60	2	27.74	0.32	
	50	15	15	1.00	0.04	15	2.52	-	
12	100	15	15	1.13	0.29	14	26.59	0.19	
	250	15	15	40.47	142.81	8	1420.20	1.26	

From the results in Table 5.3, the BPC algorithm solves all the 135 instances to optimality. The optimality is proven at the root node for 111 instances in on average 0.52 seconds. For the remaining ones, the BPC algorithm explores a few tens of nodes on 22 instances and a few hundreds on the remaining two. For these instances, the computational time increases. On average, it is equal to 153 seconds and it is above 1000 seconds for one instance only. Conversely, the heuristic with a performance guarantee of Pansart *et al.* (2022) proves the optimality of 92 instances, leaving the remaining ones with a positive optimality gap equal to 0.47%, on average. In terms of computational time, the BPC algorithm is faster than the heuristic of Pansart *et al.* (2022) on 126 instances. For the other nine instances where the BPC algorithm is slower, it corresponds to cases where the BPC algorithm is not able to prove the optimality at the root node. Hence the BPC algorithm branches, while the heuristic of Pansart *et al.* (2022) stops at the root node.

In Table 5.4, we report the results obtained by the BPC algorithm on the instances of Pansart *et al.* (2022) with $|\mathcal{J}| = 500, 750, 1000$. The rows of the table correspond to instances with the same maximal path length and the same number of incompatible pairs. The first three columns have the same meaning as those in Table 5.3. The next three columns summarise the results obtained by the BPC algorithm on the instances solved to optimality: number of instances solved to optimality (#), average number of nodes of the branch-and-bound tree (*avg.#nodes*) and average computational time (*avg.t[s]*). The last three columns summarise the results obtained by the BPC algorithm on the instances not solved to optimality: number of instances not solved to

optimality (#), average number of nodes of the branch-and-bound tree (*avg.#nodes*) and optimality gap expressed as a percentage (*avg.gap[%]*). If the BPC algorithm is not able to prove the optimality of any instance of a row, we write character '-' in columns *avg.#nodes* and *avg.t[s]*.

Table 5.4: Results on the set of instances of Pansart *et al.* (2022) with at least 500 pairs.

Instances			BPC results					
			solved instances			unsolved instances		
L^P	$\|\mathcal{I}\|$	#	#	avg.#nodes	avg.t[s]	#	avg.#nodes	avg.gap[%]
3	500	15	14	23.71	117.09	1	439.00	0.03
	750	15	10	227.40	817.41	5	371.60	0.04
	1000	15	5	91.80	544.26	10	241.80	0.08
6	500	15	5	3.80	15.95	10	242.90	0.25
	750	15	5	268.60	1200.19	10	208.30	0.64
	1000	15	0	-	-	15	206.13	0.60
12	500	15	7	57.86	707.15	8	185.13	0.98
	750	15	3	23.67	867.48	12	126.83	1.35
	1000	15	0	-	-	15	83.73	1.68

Table 5.4 shows that the performances of the BPC algorithm decrease as the size of the instances increases, both in terms of maximal path length and number of pairs. The BPC algorithm identifies 49 optima out of the 135 instances: 29 of them are attained on instances with $L^P = 3$ and the remaining ones are equally distributed on the instances with $L^P = 6$ and $L^P = 12$. Detailed results show that the time required to prove the optimality varies heavily even within instances characterised by the same values of L^P and $|\mathcal{I}|$. Precisely, it increases with the number of nodes of the branch-and-bound tree explored by the BPC algorithm. As an example, in the 14 instances solved to optimality with $L^P = 3$ and $|\mathcal{I}| = 500$, the computational time is less than 3 seconds when the instances are solved to optimality at the root node while it is larger than 500 seconds when more than 100 nodes are explored.

The BPC algorithm is not able to prove the optimality of 86 instances. However, they are left with a small optimality gap, on average, equal to 0.06%, 0.51% and 1.41% for the instances with $L^P = 3$, $L^P = 6$ and $L^P = 12$, respectively. We observe that exploring the branch-and-bound tree is rather inefficient in improving the upper bound.

Indeed, the improvement of the upper bound w.r.t. the one obtained at the root node is on average equal to 0.05%.

Although we do not report a detailed comparison with the results of Pansart *et al.* (2022), we mention that their restricted master heuristic is tested on a subset of 45 instances with at most 750 incompatible pairs and path length up to six. The results provided on those instances are worse than ours. Only one instance is solved to optimality, the remaining ones are left with an average optimality gap of 0.24%.

5.5.4 Results on the set of instances of Delorme *et al.* (2023)

Table 5.5 reports the results obtained on the instances of Delorme *et al.* (2023) when the objective function is the maximisation of the number of transplants. A comparison with the results of Delorme *et al.* (2023) is also conducted. The algorithm of Delorme *et al.* (2023) was implemented in C++ and ran on a machine equipped with an Intel(R) Core(TM) i5-1135G7, 2.40GHz and 32GB of memory.

Each row of the table corresponds to a subset of instances characterised by the same number of pairs and the same maximal length of the cycles. The first three columns report the characteristics of the subset of instances: the maximal length of the cycles (L^C), the number of pairs ($|\mathcal{J}|$) and the number of instances in the subset ($\#$). The next two columns report the results obtained by the BPC algorithm, namely, the number of instances solved to optimality ($\#opt.$) and the average time expressed in seconds to solve the instances to optimality ($avg.t[s]$). The last two columns report the same two statistics on the results obtained by Delorme *et al.* (2023). Columns are doubled to include all the instances in the table.

The results of Table 5.5 clearly highlight the superiority of the BPC algorithm w.r.t. the destructive bound procedure of Delorme *et al.* (2023). Indeed, the BPC algorithm solves to optimality all the 840 instances within 30 seconds. 821 instances are solved at the root node, the remaining ones by exploring at most 13 nodes of the branch-and-bound tree. Conversely, the procedure of Delorme *et al.* (2023) suffers from lack of scalability. Indeed, such procedure entails enumerating all the half-cycles of length up to $\lceil L^C \rceil + 1$. This operation becomes intractable when the size of the instance grows.

In Table 5.6, we present the results obtained by the BPC algorithm where the objective function is the maximisation of the medical benefit. Delorme *et al.* (2023) do not test their procedure in the case of maximisation of the medical benefit. The rows and the first three columns of the table have the same meaning as those in Table 5.5.

Table 5.5: Results on the set of instances of Delorme *et al.* (2023) where the objective is the maximisation of the number of transplants.

Instances			BPC		Delorme *et al.* (2023)		Instances			BPC		Delorme *et al.* (2023)	
L^C	$\|\mathcal{I}\|$	#	#opt.	avg.t[s]	#opt.	avg.t[s]	L^C	$\|\mathcal{I}\|$	#	#opt.	avg.t[s]	#opt.	avg.t[s]
	50	20	20	0.04	20	0.00		50	20	20	0.02	20	0.00
	100	20	20	0.02	20	0.00		100	20	20	0.11	20	0.00
	200	20	20	0.08	20	0.00		200	20	20	2.41	20	15.00
3	400	20	20	0.56	20	0.00	6	400	20	20	0.96	20	680.00
	600	20	20	2.02	20	1.00		600	20	20	2.40	0	3600.00
	800	20	20	3.25	20	3.00		800	20	20	4.02	-	-
	1000	20	20	4.26	20	6.00		1000	20	20	6.85	-	-
	50	20	20	0.02	20	0.00		50	20	20	0.02	20	0.00
	100	20	20	0.04	20	0.00		100	20	20	0.57	20	0.00
	200	20	20	2.06	20	0.00		200	20	20	1.74	20	57.00
4	400	20	20	0.82	20	7.00	7	400	20	20	0.95	10	3138.00
	600	20	20	2.25	20	41.00		600	20	20	2.55	0	-
	800	20	20	3.36	20	107.00		800	20	20	4.77	-	-
	1000	20	20	4.58	20	294.00		1000	20	20	7.59	-	-
	50	20	20	0.02	20	0.00		50	20	20	0.02	20	0.00
	100	20	20	0.11	20	0.00		100	20	20	0.47	20	1.00
	200	20	20	1.39	20	2.00		200	20	20	1.78	20	250.00
5	400	20	20	0.83	20	76.00	8	400	20	20	1.03	0	-
	600	20	20	2.58	20	623.00		600	20	20	2.62	-	-
	800	20	20	3.73	13	2735.00		800	20	20	5.16	-	-
	1000	20	20	5.64	0	3600.00		1000	20	20	8.23	-	-

The next three columns report results regarding the instances solved to optimality by the BPC algorithm: the number of such instances (#), the average number of nodes of the branch-and-bound tree to prove the optimality (*avg.#nodes*) and the average time expressed in seconds (*avg.t[s]*). The last three columns report the results regarding the instances not solved to optimality by the BPC algorithm: the number of such instances (#), the average number of nodes of the branch-and-bound tree explored by the BPC algorithm (*avg.#nodes*) and the average optimality gap expressed as a percentage (*avg.gap[%]*). We consider only instances characterised by a number of pairs up to 400.

Table 5.6: Results on the set of instances of Delorme *et al.* (2023) where the objective is the maximisation of the medical benefit.

Instances			BPC results					
			solved instances			unsolved instances		
L^C	$\|\mathcal{I}\|$	#	#	avg.#nodes	avg.t[s]	#	avg.#nodes	avg.gap[%]
3	50	20	20	1.00	0.04	0	-	-
	100	20	20	1.10	0.02	0	-	-
	200	20	20	4.80	0.10	0	-	-
	400	20	20	62.50	28.23	0	-	-
4	50	20	20	1.00	0.02	0	-	-
	100	20	20	2.10	0.17	0	-	-
	200	20	20	61.90	44.49	0	-	-
	400	20	6	420.83	1502.51	14	511.14	0.22
5	50	20	20	1.00	0.02	0	-	-
	100	20	20	3.40	0.47	0	-	-
	200	20	20	158.75	245.58	0	-	-
	400	20	0	-	-	20	247.30	0.51
6	50	20	20	1.00	0.02	0	-	-
	100	20	20	8.40	2.72	0	-	-
	200	20	17	391.06	534.63	3	42.33	0.25
	400	20	0	-	-	20	201.05	0.93
7	50	20	20	1.00	0.02	0	-	-
	100	20	20	18.90	9.32	0	-	-
	200	20	13	510.38	940.12	7	425.57	0.18
	400	20	0	-	-	20	173.00	1.10
8	50	20	20	1.10	0.02	0	-	-
	100	20	20	6.20	6.98	0	-	-
	200	20	15	395.53	823.12	5	201.80	0.20
	400	20	0	-	-	20	155.80	1.29

From the results of Table 5.6, it can observed that the type of the objective function has a huge impact on the results. Maximising the medical benefit makes the instances

much harder to be solved by the BPC algorithm. The BPC algorithm still provides 371 optima out of the 480 instances. However, contrary to the results where the objective is to maximise the number of transplants, those optima are obtained by exploring more nodes of the branch-and-bound tree (115, on average) and by spending more time (133 seconds, on average). The 110 instances not solved to optimality are characterised by either 200 or 400 pairs and a maximal length of the cycles at least equal to four. None of the instances with 400 pairs and maximal length larger or equal to five is solved to optimality. Nonetheless, the average optimality gap when the time limit is reached is small, 0.76%, on average. The same behaviour was observed when considering instances with more tha 400 pairs.

5.5.5 Impact of the valid inequalities

In this section, we evaluate the impact of the valid inequalities presented in Section 5.4.4, namely the SR inequalities (5.6) and the OH inequalities (5.7). We conduct this analysis on the most challenging instances in the case where the objective is the maximisation of the medical benefit, namely those of Pansart *et al.* (2022) with a number of pairs larger than 500 and those of Delorme *et al.* (2023) with a number of pairs larger than 200.

Other than the configuration of the BPC algorithm we used to run the experiments, we consider a configuration of it where no valid inequality is separated. We denote the two configurations `BPC` and `BPC basic`, respectively.

Table 5.7 compares the results obtained by the two configurations when the instances are solved to optimality. The rows of the table correspond to the two subsets of instances we consider. The first two columns report the name of the set of instances (*set*) and the number of instances in the set (#). For each configuration, the columns report the number of instances solved to optimality over the number of instances in the subset (*#opt.*), the average number of nodes of the branch-and-bound tree (*avg.#nodes*) and the average computational time (*avg.t[s]*). An additional column reports the average time spent in the separation procedure (*avg.t[s] sep.*) for configuration `BPC`.

Table 5.7: Comparison of two variant of the BPC algorithm on the instances solved to optimality.

		BPC basic			BPC			
set	#	#opt.	avg.#nodes	avg.t[s]	#opt.	avg.#nodes	avg.t[s]	avg.t[s] sep.
Pansart *et al.* (2022)	135	46	287.93	531.46	49	100.06	534.04	212.33
Delorme *et al.* (2023)	240	126	403.02	357.54	131	209.92	374.36	93.58

The results are similar on both sets of instances. Indeed, configuration `BPC` solves to optimality a few more instances than `BPC basic`: three more for the instances of Pansart *et al.* (2022) and five more for the instances of Delorme *et al.* (2023). On average, `BPC` explores at least half of the number of nodes of the branch-and-bound tree w.r.t. `BPC basic`. The average time to solve the instances remains comparable between the two configurations (see columns *avg.t[s]*). Probably, the explanation is that the average time taken by the separation procedure is rather high: a more efficient implementation of such procedure may be required. Moreover, considering the instances of Pansart *et al.* (2022), it can be noticed that only 44 instances out of the 46 solved to optimality by `PBC basic` are also solved to optimality by `BPC`. When we consider only these 44 instances solved to optimality in both configurations, the potential of `BPC` clearly emerges. On average, `BPC basic` explores 297.11 nodes of the branch-and-bound tree and spends 447.71 seconds to solve the instances, whereas `BPC` explores 54 nodes and spends 392.44 seconds. The same trend emerges on the instances of Delorme *et al.* (2023).

Table 5.8 compares the results obtained by the two configurations when the instances are not solved to optimality. The rows and the first two columns of the table have the same meaning as in Table 5.7. For each configuration, the columns report the number of instances not solved to optimality over the number of instances in the subset (*#noOpt.*), the average number of nodes of the branch-and-bound tree (*avg.#nodes*) and the average upper bound when the time limit is hit (*avg.UB*). An additional column reports the average time spent in the separation procedure (*avg.t[s] sep.*) for configuration `BPC`.

Table 5.8: Comparison of two variant of the BPC algorithm on the instances not solved to optimality.

		BPC basic			BPC			
set	#	#noOpt.	avg.#nodes	avg.UB	#noOpt.	avg.#nodes	avg.UB	avg.[s] sep.
Pansart *et al.* (2022)	135	89	331.11	38 241.97	86	192.77	38 679.58	591.94
Delorme *et al.* (2023)	240	114	452.39	17 738.34	109	246.00	17 898.47	569.34

`BPC` explores on average less nodes of the branch-and-bound tree. This reflects in the average upper bound returned by both configurations at the time limit. The upper bound of `BPC basic` is slightly better than the one of `BPC`. Hence, it seems that exploring the branch-and-bound tree is more promising than including valid inequalities. However, also in this case, the potential of `BPC` emerges when the comparison is done on the instances not solved to optimality by both configurations. Indeed, on the instances of Pansart *et al.* (2022), `BPC` returns upper bounds of slightly better quality by exploring less nodes. For `BPC` the average upper bound and number of nodes are 38 528.57 and 195.56, respectively, whereas for `BPC basic` they are 38 532.40 and 282.46, respectively. The same observation can be done for the instances of Delorme *et al.* (2023). Again, this is another hint for the need to improve the efficiency of the separation procedure which in both cases reaches the 10 minutes time limit, on average. In addition, remark that when valid inequalities are active in the RMP, the circuits are priced by a labelling algorithm in lieu of the less time consuming variant of the Bellman-Ford algorithm presented in Section 5.4.3.

Finally, Table 5.9 reports the average number of OH and SR inequalities added to the RMP. The two rows of the table correspond to the two sets of instances we consider. The first column reports the name of the set of instances (*set*). The next two columns report the average number of OH inequalities (*avg.#OH*) and of SR inequalities (*avg.#SR*) added to the RMP, for the instances solved to optimality by the BPC algorithm. The last two columns show the same statistics for the instances not solved to optimality by the BPC algorithm.

Table 5.9: Average number of SR and OH inequalities added to the RMP.

set	solved instances		unsolved instances	
	avg.#OH	avg.#SR	avg.#OH	avg.#SR
Pansart *et al.* (2022)	44.8	14.2	24.5	27.7
Delorme *et al.* (2023)	43.5	25.0	29.2	26.5

From Table 5.9, we observe that the number of inequalities added to the RMP is far from 100, i.e., the maximal number of inequalities we separate in the branch-and-bound tree. Hence, in particular in the case of the unsolved instances, the separation procedure reaches the 10 minutes time limit without providing the possible inequalities.

5.5.6 Impact of the length constraints on the objective function

In this section, we assess the impact of the length constraints of the exchange paths on the objective function value. To conduct such analysis, we take into account both objective function types, i.e., maximisation of the number of transplants (#TR) and maximisation of the medical benefit (MB). For the former case, we only consider the instances of the PrefLib dataset with maximal length of the circuits $L^C = 3$; for the latter case, we only consider the instances of Pansart *et al.* (2022) with 50, 100, or 250 incompatible pairs. In both sets, we group the instances characterised by the same compatibility graph and we observe how the objective function value behaves when the maximal length of the exchange paths increases. Remark that all these instances are solved to optimality by the BPC algorithm.

In the case of the maximisation of the number of transplants, we do not report any table because the improvements are extremely sporadic. Indeed, on the 230 instances characterised by the same compatibility graph, the number of transplants improves only 19 times when we increase the length of the paths from 3 to 4, from 4 to 5 or from 5 to 6. Therefore, it seems that the influence of the path length on the number of transplants is negligible.

Finally, Table 5.10 reports the average improvements of the medical benefit when the maximal length of the paths increases on the instances of Pansart *et al.* (2022) with up to 250 incompatible pairs. The rows of the table group instances with the same number of incompatible pairs (same compatibility graph). The first two columns report the number of incompatible pairs ($|\mathcal{I}|$) and the number of instances (#) in each

group. The remaining two columns report the average improvement of the medical benefit (objective function value) expressed as a percentage (*avg.impr[%]*) when the maximal path length increases from three to six ($L^P : 3 \rightarrow 6$) and from six to twelve ($L^P : 6 \rightarrow 12$), respectively.

Table 5.10: Impact of the path length increase on the set of instances of Pansart *et al.* (2022) when the objective is the maximisation of the medical benefit (MB).

Instances		avg.impr[%]	
$\|\mathcal{I}\|$	#	$L^P : 3 \rightarrow 6$	$L^P : 6 \rightarrow 12$
50	15	9.31	2.13
100	15	15.77	5.41
250	15	14.36	5.13

A different trend emerges from the results in Table 5.10 w.r.t. the case of the maximisation of the number of transplants. Indeed, we observe that the maximal length constraints on the paths has a great impact on the value of the objective function. The largest improvements of the medical benefit are obtained when the path length is increased from three to six. Such improvements are lessened when the maximal length is increased from six to twelve.

5.6 Conclusions

In this paper, we introduced an exact approach which is able to solve the Kidney Exchange Problem (KEP) where long cycles and chains of donations are considered. Our approach is based on a Branch-Price-and-Cut (BPC) algorithm where the pricing problem is decomposed in subproblems to price out variables associated with cycles and chains, respectively. We consider two families of non-robust valid inequalities to strengthen the relaxation, namely subset-row and odd-hole inequalities. We test the BPC algorithm against the three algorithms recently proposed in the literature, namely, the restricted master heuristic of Pansart *et al.* (2022), the destructive bound procedure of Delorme *et al.* (2023) and the Branch-and-Price algorithm of Arslan *et al.* (2022). The BPC algorithm outperforms the results of the first two and, against the third one, it provides the same optima, but in slower computational time. In addition, the impact of the valid inequalities is assessed: they permit to reduce both the time and the exploration of the branch-and-bound tree when the instances are solved to optimality.

However, the results on the instances which are not solved to optimality reveal that a more efficient separation procedure is to investigate as future work. In addition, another research direction may consider the development of a BPC algorithm based on the formulation introduced by Delorme *et al.* (2023), where the exponentially-many variables are the *half-cycles*.

Chapter 6

Collaborative and fairness aspects in the Iterative International Kidney Exchange Problem

Contents

Abstract: Kidney exchange programmes aim at determining a set of kidney transplants between patients and donors such that the medical benefit associated with the transplants is maximised. In order to perform a transplant, the patient and the donor must be compatible. i.e., they must meet certain medical requirements. Commonly, kidney exchange programmes are run individually by a single country over multiple rounds. However, in recent years, several countries subscribe to a common programme to increase the chances of performing transplants. In this collaborative environment, we study the *Iterative International Kidney Exchange Problem* (IIKEP). The aim of the IIKEP is to periodically determine a set of transplants between the patients and donors of different countries such that the disparities between the countries are minimised. Precisely, we borrow concepts from cooperative game theory to model stability and fairness conditions for the system.

The nature of the problem induces an iterative solving procedure. The problem arising at each iteration (round) is a Kidney Exchange Problem (KEP) enriched with stability and fairness conditions. Compared with the existing literature on the IIKEP, our underlying KEP considers the possibility of performing both cycles and long chains of kidney transplants. We model such problem by means of a set packing formulation involving an exponential number of variables and constraints and we solve it by means of a Branch-Price-and-Cut (BPC) algorithm.

The experimental results show that the stability and fairness conditions we impose help in reducing the disparities between the countries and make the collaborative system more sound.

Keywords: Kidney exchange, stability, fairness, cooperative game theory, Branch-Price-and-Cut.

6.1 Introduction

Kidney transplantation remains the preferable treatment for patients with a severe kidney condition (Yoo *et al.*, 2016). Such operations are performed between patients and donors who are compatible with each other according to several medical indicators (Kälble *et al.*, 2005). Indeed, one of the major concerns related to the transplants is the risk of rejection. In this respect, each transplant is assigned with a medical

benefit which measures the goodness of performing it. Although the majority of the kidneys to transplant come from deceased donors, transplants involving a living donor are on the rise*. Indeed, human beings can have a healthy life with one functioning kidney. Usually, a patient is able to find a willing donor among his/her acquaintances. If the donor is compatible with him/her, the kidney transplantation can be performed. However, in many cases, the compatibility requirements are not met and the transplantation cannot be performed. Under these circumstances, the patient and donor form a so-called *incompatible patient-donor pair*. The donor of such a pair is willing to give a kidney to a patient of another pair, if its associated patient is guaranteed to receive a kidney, as well. Kidney transplants which involve incompatible patient-donor pairs are organised by the *Kidney Exchange Programmes*. Other than incompatible pairs, such programmes may also involve *altruistic donors*, i.e., donors who are willing to give a kidney to whomever patient. Nowadays, kidney exchange programmes are commonly run periodically, every three or four months, at country levels (see, e.g., Biró *et al.*, 2019a; Biró *et al.*, 2021, for the regulations of the European programmes), i.e., they involve incompatible pairs and altruistic donors of several hospitals of a same country to increase the medical benefit arising from the transplants. Precisely, at each round, they entail solving an optimisation problem referred to as *Kidney Exchange Problem* (KEP). In the KEP, the objective is to determine a set of kidney exchanges in a pool of incompatible patient-donor pairs and altruistic donors such that the medical benefit is maximised. The kidney exchanges to be determined can be of two types: *cycles of donations* or *chains of donations*. The former are composed of only incompatible pairs where the donor of a pair gives its kidney to the patient of the following pair in the cycle and so on. The latter is a domino donation which is started by an altruistic donor. Both cycles and chains of donations are of limited length due to practical reasons or regulations. We refer to Chapter 5 for a more detailed review on the KEP.

Nowadays, running a kidney exchange programme with incompatible pairs and altruistic donors of a unique country is a well-established practice. In this context, a state institution acts as a central coordinator and a unique regulation is applied. However, in order to increase the possibilities of finding compatible patients and donors, different countries merge their pool of incompatible patient-donor pairs and altruistic donors to solve the KEP jointly (see, e.g., Böhmig *et al.*, 2017; Scandiatransplant, 2023; Valentín *et al.*, 2019). Such practice appeared only in recent years due to the

*https://www.transplant-observatory.org/

difficulties in the logistics of performing transplants with patients and donors of different countries and in the merging of the regulations applied in the programmes of the different countries. In this context, the KEP is often referred to as *International Kidney Exchange Problem* (IKEP). Clearly, solving the IKEP leads to solutions which are globally better than those obtained by solving the KEP for the single countries separately (Kute *et al.*, 2018). However, the solutions of the IKEP may be detrimental for some countries or group of countries. Indeed, the medical benefit obtained for their transplanted patients can be worst in a solution of the IKEP in comparison with the solution they could achieve on their own. In order to prevent some countries from leaving the system, it is necessary to impose stability conditions in this collaborative setting. As for the KEP within a single country, also the IKEP is run periodically, so we refer to it as *Iterative International Kidney Exchange Problem* (IIKEP). In this context, at every round of the IKEP, there is a global surplus of medical benefit w.r.t. the situation where each country would act individually. In an ideal situation, this surplus should be fairly allocated to the countries participating in the programme. Note that at a given round, since a set of transplants has to selected, it is not possible to guarantee this ideal fair allocation to the countries, and there is usually a deviation between the solution and the ideal fair allocation. In the long term, however, fairness conditions can be considered. Precisely, at each round of the IKEP, the medical benefit assigned to each country should not deviate too much from an ideal fair target of medical benefit computed for the countries taking into account the existing deviations from the previous rounds of the IKEP.

Recently, several works dealing with stability and fairness in the context of the IIKEP appeared in the literature. Many of them make use of concepts coming from non-cooperative and cooperative game theory. In the former case, the idea is to determine a game plan which gives no incentive to the countries to hide information from the central authority that determines the kidney exchanges in the merged pool. In the latter case, the countries usually reveal all the information about their pool and form a coalition to maximise the global medical benefit. In the following we review these works.

Carvalho *et al.* (2016) is the first to model the IKEP as a non-cooperative game. The authors focus on studying the game associated with the IKEP with only cycles of length two and where the objective is the maximisation of the number of transplants. Precisely, they prove that the game admits a pure Nash equilibrium which can be

computed in polynomial time. In addition, they show that a Nash equilibrium where the maximum number of transplants is achieved always exists. Carvalho & Lodi (2023) generalises the result of Carvalho *et al.* (2016) to games with more than two players, i.e., countries. In Carvalho *et al.* (2016) and Carvalho & Lodi (2023), the stability of the system is ensured by means of a two-round procedure: *round 1*: each country solves the KEP on its own; *round 2*: the countries share their patients and donors not involved in the transplants determined in the first round and an independent agent determines a solution for the associated IKEP. In addition, the game arising from the IKEP when the objective is the maximisation of the medical benefit is also studied.

In the following, we review the works which model the problem as a cooperative game. Smeulders *et al.* (2022) consider a special case where the countries have to decide which subset of incompatible pairs participate in the common pool. They model the problem as a Stackelberg game and draw some theoretical results about its complexity. Hereinafter, we review the works where all the incompatible pairs of each country participate in the common pool.

Biró *et al.* (2019b) studied the generalised matching game arising when modelling the IKEP with only cycles of length two as a cooperative game with transferable utility. Remark that solving the IKEP with cycles of length two entails determining a matching in a graph, that can be performed in polynomial time.

Klimentova *et al.* (2021) impose both stability and fairness conditions in an IIKEP. Precisely, the authors exploit the fact that the (I)KEP has several equivalent optimal solutions and, at each round, they select the one that is the most stable and fair for the system. The stability is modelled as an intra-round condition: each single country achieves a number of transplants larger than or equal to the one obtained by solving the KEP on its own. The fairness is modelled as an inter-round condition: at each round, each country should achieve a number of transplants as close as possible to a target number of transplants. Such target number is computed by taking into account information of the previous rounds and an ideal number of transplants that the country should achieve at the current round.

Benedek *et al.* (2021) apply the fairness measure of Klimentova *et al.* (2021) in the IIKEP with only cycles of length two. Contrary to Klimentova *et al.* (2021), the target number of transplants at each round is computed by borrowing concepts from the theory of cooperative games, such as the Shapley value and the nucleolus. The authors, first, compute a target number of transplants for each country, then, they

determine a solution of the IKEP which minimises the deviation from such targets. They are the first to test their approach on instances with a large number of countries, up to 15.

Finally, Benedek *et al.* (2023) also consider the IIKEP with only cycles of donations. Here, the IKEP arising at each round is modelled as a newly introduced class of games, the partitioned matching games. Fairness conditions are applied as in Klimentova *et al.* (2021). The authors provide computational complexity results on determining a maximal matching which minimises the deviation from the target fair allocations of the countries, at each round.

In this work, we extend the works of Klimentova *et al.* (2021) and Benedek *et al.* (2021) for the IIKEP. First, we impose the stability of the system by borrowing the concept of *least-core* from the cooperative game theory. The idea is to minimise the possibility for any subset of countries to leave the system. Klimentova *et al.* (2021) ensure this condition only for single countries (subsets of cardinality one), while we ensure this condition for all subsets of countries. We impose fairness as an inter-round condition as in Klimentova *et al.* (2021). In addition, Klimentova *et al.* (2021) and Benedek *et al.* (2021) consider the maximisation of the number of transplants as objective for the IKEP while, here, we consider a more general case that is the maximisation of the medical benefit. We conduct computational experiments considering large instances as in Benedek *et al.* (2021) with initial pools with 1000 and 2000 pairs. However, contrary to the previous works in the literature which either do not consider chains of donation or consider chains of small length (see, e.g., Klimentova *et al.*, 2021), our instances also include altruistic donors and the possibility of building chains of length up to 7. Under this assumption, the IKEP cannot be solved by polynomial-time algorithms as in Benedek *et al.* (2021) or by enumeration as in Klimentova *et al.* (2021), but requires more sophisticated techniques as Branch-Price-and-Cut algorithms (see Chapter 5).

The reminder of the chapter is organized as follows. In section 6.2, we recall the concepts of the theory of cooperative game with transferable utility we use in this work. In Section 6.3, we formally describe the IIKEP. In Section 6.4, we present a formulation with an exponential number of variables and constraints for the IKEP arising at each round of the IIKEP where stability and fairness constraints are applied. The solution procedure for the IIKEP is detailed in Section 6.5. Finally, in Section 6.6, we analyse the stability and fairness of the solutions we obtained on the IIKEP.

6.2 Concepts of cooperative game theory

In this section, we recall some useful concepts from the theory of cooperative games with transferable utility. Given a set $\mathcal{N}$, we denote by $\mathcal{P}(\mathcal{N})$ its power set. Let $\Gamma = (\mathcal{N}, v)$ be a cooperative game with transferable utility in coalitional form defined over set of players $\mathcal{N} \neq \emptyset$ and coalition function $v : \mathcal{P}(\mathcal{N}) \to \mathbb{R}$, $v(\emptyset) = 0$. Given a *coalition* $\mathcal{S} \in \mathcal{P}(\mathcal{N})$, i.e., a subset of players playing the game together, value $v(\mathcal{S})$ represents the profit that the players in $\mathcal{S}$ can achieve. The coalition $\mathcal{N}$ containing all the players is referred to as *grand coalition*. If $v(\mathcal{N}) \geq v(\mathcal{S})$ for all $\mathcal{S} \in \mathcal{P}(\mathcal{N})$, then the set of players $\mathcal{N}$ has an interest to form the grand coalition. In the context of transferable utility, when a coalition $\mathcal{S} \in \mathcal{P}(\mathcal{N})$ is formed, profit $v(\mathcal{S})$ can be divided in any possible way among the players in $\mathcal{S}$. An *allocation* of $v(\mathcal{N})$ can be represented by a vector $x \in \mathbb{R}^{|\mathcal{N}|}$. The question is then how to allocate $v(\mathcal{N})$ among the players such that no set of players has interest to leave the grand coalition.

A reasonable answer to this question is given by the allocations which belong to the *core* of the game, i.e., to set

$$\mathcal{C}(\Gamma) = \left\{ x \in \mathbb{R}^{|\mathcal{N}|} : \sum_{i \in \mathcal{N}} x_i = v(\mathcal{N}) \wedge \sum_{i \in \mathcal{S}} x_i \geq v(\mathcal{S}), \forall \mathcal{S} \in \mathcal{P}(\mathcal{N}) \right\}.$$

Indeed, an allocation x in the core distributes value $v(\mathcal{N})$ of the grand coalition among the player in a *stable* manner. It is ensured that value $x(\mathcal{S}) \coloneqq \sum_{i \in \mathcal{S}} x_i$ allocated to each coalition $\mathcal{S}$ is larger than or equal to value $v(\mathcal{S})$ that the players of $\mathcal{S}$ could achieve if playing on their own. Hence, no coalition would get any benefit to disagree on a profit allocation in the core.

Example 6.1 (The core)**.** *We consider a toy game with three players:* $\Gamma = (\mathcal{N} = \{1,2,3\}, v)$*, where* $v(\{1\}) = v(\{2\}) = v(\{3\}) = 10$*,* $v(\{1,2\}) = 30$*,* $v(\{1,3\}) = v(\{2,3\}) = 80$*, and* $v(\{1,2,3\}) = 100$*. It is easy to see that, for example, allocation* $x = (10, 20, 70)$ *is in the core, indeed, all the conditions are satisfied:*

$$x_1 + x_2 + x_3 = v(\{1,2,3\}) = 100,$$

$$\begin{array}{ccc} x_1 = 10 \geq v(\{1\}), & x_2 = 20 \geq v(\{2\}), & x_3 = 70 \geq v(\{3\}), \\ x_1 + x_2 = 30 \geq v(\{1,2\}), & x_1 + x_3 = 80 \geq v(\{1,3\}), & x_2 + x_3 = 90 \geq v(\{2,3\}). \end{array}$$

The non-emptiness of the core ($\mathcal{C}(\Gamma) \neq \emptyset$) depends on the property of the game and in several applications it is not guaranteed. To overcome this issue, several other profit

allocation methods have been proposed whose aim is to determine an allocation "as close as possible" to satisfy the conditions that define the core. Among these methods, we present in the following the *least-core* (see Peleg & Sudhölter, 2007). First, given $\epsilon \in \mathbb{R}$, the ϵ-core $\mathcal{C}_\epsilon(\Gamma)$ is defined as a relaxation of the core as follows:

$$\mathcal{C}_\epsilon(\Gamma) = \left\{ x \in \mathbb{R}^{|\mathcal{N}|} : \sum_{i \in \mathcal{N}} x_i = v(\mathcal{N}) \wedge \sum_{i \in \mathcal{S}} x_i \geq v(\mathcal{S}) - \epsilon, \forall \mathcal{S} \in \mathcal{P}(\mathcal{N}) \right\}.$$

Value ϵ can be interpreted as the cost to pay when the coalitions form. Hence, the profit allocations in an ϵ-core cannot be improved by any coalition, if cost ϵ is to be paid.

The least-core is defined as the intersection of all non-empty ϵ-cores, that is the ϵ-core with the smallest cost to pay for the formation of the coalitions.

Another profit allocation method is the so-called *Shapley value* $\phi(\Gamma) \in \mathbb{R}^{|\mathcal{N}|}$ introduced by Shapley *et al.* (1953) and defined by the following expression:

$$\phi_i(\Gamma) = \sum_{\substack{\mathcal{S} \in \mathcal{P}(\mathcal{N}) \\ i \notin \mathcal{S}}} \frac{|\mathcal{S}|!(|\mathcal{N}| - |\mathcal{S}| - 1)!}{|\mathcal{N}|!} (v(\mathcal{S} \cup \{i\}) - v(\mathcal{S})), \qquad i \in \mathcal{N}.$$

Value $\phi_i(\Gamma)$ is a weighted sum of the contribution that player $i \in \mathcal{N}$ brings to each coalition, where the weights are the probabilities of the occurrence of the coalitions. Note that the Shapley value permit to allocate $v(\mathcal{N})$ among the players, that is: $\sum_{i \in \mathcal{N}} \phi_i(\Gamma) = v(\mathcal{N})$. However, if the core is non-empty, the Shapley value may not belong to it. In addition, in many applications its computation is time consuming when $|\mathcal{N}|$ is large.

Finally, we report the definitions of the *benefit* and *contribution* values, denoted by $\beta(\Gamma) \in \mathbb{R}^{|\mathcal{N}|}$ and by $\kappa(\Gamma) \in \mathbb{R}^{|\mathcal{N}|}$, respectively. The benefit value measures the additional profit obtained when considering a player i in the grand coalition rather than considering that player alone:

$$\beta(\Gamma)_i = v(\mathcal{N}) - v(\mathcal{N} \setminus \{i\}) - v(\{i\}), \qquad i \in \mathcal{N}.$$

The contribution value measures the additional profit brought by a player i to the grand coalition:

$$\kappa(\Gamma)_i = v(\mathcal{N}) - v(\mathcal{N} \setminus \{i\}), \qquad i \in \mathcal{N}.$$

Both such values require less computational effort to be computed than the Shapley value. Note that these value do not permit to share the grand coalition value $v(\mathcal{N})$ directly, however, they can be used to share the extra profit generated by forming the grand coalition w.r.t. the case where each player acts individually, that is value $v(\mathcal{N}) - \sum_{i\in\mathcal{N}} v(\{i\})$. Following Klimentova *et al.* (2021), using the benefit value, the value x_i allocated to each player $i \in \mathcal{N}$ is computed as:

$$x_i = v(\{i\}) + \left(v(\mathcal{N}) - \sum_{j\in\mathcal{N}} v(\{j\})\right) \frac{\beta(\Gamma)_i}{\sum_{j\in\mathcal{N}} \beta(\Gamma)_j}.$$

An allocation based on the contribution value is computed as in the above formula, where the benefit value is replaced with the contribution value.

Example 6.2 (Shapley, benefit and contribution values)**.** *We consider the toy game* $\Gamma = (\mathcal{N}, v)$ *as defined in Example 6.1. The Shapley value associated with* Γ *is* $\phi(\Gamma) = (25, 25, 50)$*. Note that allocation* $\phi(\Gamma)$ *does not belong to the core, indeed, e.g., it violates the condition for coalition* $\{1, 3\}$*:* $\phi(\Gamma)_1 + \phi(\Gamma)_3 = 75 < v(\{1, 3\}) = 80$*.*

The allocation associated with the benefit value is $x = (18.75, 18.75, 62.5)$ *and the one associated with the contribution value is* $x = (22.7, 22.7, 54.6)$*.*

6.3 Problem description

The aim of this section is to introduce the *Iterative International Kidney Exchange Problem* (IIKEP). To do so, we first describe the basic *Kidney Exchange Problem* (KEP) in Section 6.3.1. Then, in Section 6.3.2, we describe the *International Kidney Exchange Problem* (IKEP), i.e., the KEP arising when different countries share their pool of donors and patients. Finally, the IIKEP is presented in Section 6.3.3.

6.3.1 Kidney Exchange Problem

Let $\mathcal{I}$ be the set of incompatible patient-donor pairs and let $\mathcal{D}$ be the set of altruistic donors. The KEP can be defined on a directed weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{A})$ referred to as *compatibility graph*. Vertex set $\mathcal{V} = \mathcal{I} \cup \mathcal{D}$ contains a vertex for each incompatible patient-donor pair and each altruistic donor. The arcs in $\mathcal{A}$ model all the possible transplants between donors and patients. Hence, arc set $\mathcal{A}$ contains an arc (i, j) from each vertex $i \in \mathcal{V}$ to each patient-donor pair $j \in \mathcal{I}$ if the kidney of the donor associated with

i is compatible with the patient of pair j. We assign a weight W_{ij} to each arc $(i,j) \in \mathcal{A}$ representing the medical benefit of the associated transplant. Kidney exchanges between donors and patients are modelled in graph $\mathcal{G}$ by two types of subgraphs, namely circuits and paths. In this respect, we call an *exchange circuit* an elementary circuit in graph $\mathcal{G}$ of length at most $L^C > 1$. Similarly, we call an *exchange path* an elementary path in graph $\mathcal{G}$ of length at most $L^P > 1$. To avoid any confusion we emphasize that the length of a circuit or a path is defined as the number of edges it contains, that is a circuit with n vertices has length n and a path with n vertices has length $n-1$. Given that vertices associated with altruistic donors do not have in-going arcs, exchange circuits are composed only of vertices associated with patient-donor pairs and exchange paths must start with a vertex associated with an altruistic donor. We denote by $\mathcal{E}$ the set of the exchanges in graph $\mathcal{G}$, i.e., of the exchange circuits and paths in graph $\mathcal{G}$. We define the weight of an exchange $e \in \mathcal{E}$ as the sum of the weights of the arcs traversed by e, i.e., $W_e \coloneqq \sum_{(i,j)\in\mathcal{A}(e)} W_{ij}$, where $\mathcal{A}(e)$ is the set of arcs traversed by e. Finally, we define an *exchange scheme* as a union of pairwise vertex-disjoint exchanges of $\mathcal{E}$. The KEP aims to determine an exchange scheme of maximum weight, where the weight of an exchange scheme is the sum of the weights of the exchanges composing it.

6.3.2 International Kidney Exchange Problem with stability

When different countries join a common kidney exchange programme, an *International Kidney Exchange Problem* (IKEP) arises. In the IKEP, the countries share their pool of patients and donors to solve the KEP jointly. Doing so, the total medical benefit of the system increases. However, stability conditions need to be applied in order to ensure the soundness of the system.

Let $\mathcal{H}$ be the set of countries participating in a kidney exchange programme. We suppose that the countries agree on a common maximal length of the exchange circuits and paths. We denote by $\mathcal{G}_h = (\mathcal{V}_h = \mathcal{I}_h \cup \mathcal{D}_h, \mathcal{A}_h)$ the compatibility graph arising from the set of incompatible patient-donor pairs $\mathcal{I}_h$ and the set of altruistic donors $\mathcal{D}_h$ available for country $h \in \mathcal{H}$. We set $\mathcal{G} = (\mathcal{V} = \bigcup_{h\in\mathcal{H}} \mathcal{V}_h, \mathcal{A})$ to be the compatibility graph whose vertex set is the union of the incompatible patient-donor pairs and altruistic donors available for the countries in $\mathcal{H}$. Arc set $\mathcal{A}$ models the possible transplants in vertex set $\mathcal{V}$ and it is defined similarly to the KEP (see Section 6.3.1). As for the KEP, in the IKEP, an exchange scheme X on graph $\mathcal{G}$ is to be determined. However, under this collaborative setting, the interest is not only to determine an exchange scheme,

but also to ensure the stability of the system. We say that the system is *stable* if no country or coalition of countries has interest in leaving the system in order to get a better profit, i.e., a better medical benefit by doing a kidney exchange programme on its own.

We model the stability of the system by means of the concept of the least-core of a cooperative game with transferable utility (see Section 6.2). First, we model the IKEP as a cooperative game with transferable utility (Biró *et al.*, 2019b). We consider a game $\Gamma = (\mathcal{H}, v)$ where the set of players is the set of countries and the coalition function $v : \mathcal{P}(\mathcal{H}) \to \mathbb{R}$ is defined as $v(\mathcal{S}) = KEP(\mathcal{S})$. Given a coalition $\mathcal{S} \in \mathcal{P}(\mathcal{H})$, $KEP(\mathcal{S})$ is the value of the KEP computed on the subgraph $\mathcal{G}_\mathcal{S} = (\mathcal{V}_\mathcal{S} = \bigcup_{h \in \mathcal{S}} \mathcal{V}_h, \mathcal{A}_\mathcal{S})$ of compatibility graph $\mathcal{G}$ whose altruistic donors and incompatible pairs are those of the countries in $\mathcal{S}$. To ensure the stability we use the conditions defining the least-core. Hence, the objective of the IKEP is to minimise ϵ such that the exchange scheme solution ensures that for each coalition of countries $\mathcal{S}$, the medical benefit it gets in the exchange scheme is greater than $KEP(\mathcal{S})$, if $|\mathcal{S}| = 1$ and greater than $KEP(\mathcal{S}) - \epsilon$ otherwise. Note that if a coalition represents a single country it is more natural to ensure that the medical benefit it gets in such exchange scheme is larger or equal than what it would get on its own. In addition, an exchange scheme for which each country gets a medical benefit greater or equal to what it would get on its own can always be found: it is sufficient to consider the exchanges determined when solving the KEP individually. Retrieving the value $KEP(\mathcal{S})$ requires to solve an optimization problem. Also, note that here we just derive the concept of least-core to model stability, but the problem we consider is not a cooperative game with transferable utility. Indeed, we are not searching for an allocation of the total medical benefit to the countries, but we are looking for a stable exchange scheme.

6.3.3 Iterative International Kidney Exchange Problem

In this section, we present the problem arising when a kidney exchange programme involving different countries is run over multiple rounds. At each round, the countries collaborate to solve an IKEP in order to increase the total medical benefit of the system. However, disparities on how to distribute the medical benefit among the countries may rise. Indeed, even by including stability conditions a solution may produce unfair allocation of the extra medical benefit generated by the grand coalition. It is possible to compute ideal fair allocation of such extra benefit, and a solution should not deviate

too much from these ideal values. Trying to guarantee small deviations in a single period is not easy, given that a set of transplants has to be determined. However, small deviations can be attained on the long term, by exploiting the fact that, at each round, an instance of the (I)KEP usually has multiple equivalent optimal solutions (see Klimentova *et al.*, 2021).

Running the kidney exchange programmes over several rounds helps in reducing such disparities. In the following, we present the *Iterative International Kidney Exchange Problem* (IIKEP) which extends the work of Klimentova *et al.* (2021). Precisely, first, we describe how a kidney exchange programme is run over multiple rounds. Then, we describe the fairness conditions we apply to smooth the disparities between the countries over the rounds.

Let $\mathcal{H}$ be the set of countries participating in a kidney exchange programme which is run for T rounds. We denote by $\mathcal{G}_h^t = (\mathcal{V}_h^t = \mathcal{I}_h^t \cup \mathcal{D}_h^t, \mathcal{A}_h^t)$ the compatibility graph arising from the set of incompatible patient-donor pairs $\mathcal{I}_h^t$ and the set of altruistic donors $\mathcal{D}_h^t$ available for country $h \in \mathcal{H}$ at round $t = 1, \ldots, T$. Arc set $\mathcal{A}_h^t$ models the possible transplants available for country h at round t and it is defined on vertex set $\mathcal{V}_h^t$ as for the KEP (see Section 6.3.1). $\mathcal{G}^t = (\mathcal{V}^t = \bigcup_{h \in \mathcal{H}} \mathcal{V}_h^t, \mathcal{A}^t)$ is the compatibility graph whose vertex set is the union of the incompatible patient-donor pairs and altruistic donors available for the countries in $\mathcal{H}$ at round $t = 1, \ldots, T$. Arc set $\mathcal{A}^t$ models the possible transplants available for all the countries at round t, i.e., it is defined on vertex set $\mathcal{V}^t$. At each round $t = 1, \ldots, T$, an exchange scheme X^t on graph $\mathcal{G}^t$ is to be determined. Remark that such exchange scheme may not involve all the patients and donors available at round t. We refer to the incompatible pairs involved in an exchange scheme at round t as *served*, and *unserved* otherwise. Similarly, we refer to the altruistic donors involved in an exchange scheme at round t as *exploited*, and *unexploited* otherwise. In addition, if $e \in X^t$ is an exchange path, the patient of the last pair in the path receives a kidney, but the associated donor does not donate a kidney. We call such pairs *partially served pairs*. The unserved pairs and the unexploited altruistic donors at round t need to be considered in the next round. From a practical point of view, we suppose that if an incompatible pair is unserved for the first time at round t, it stays available in the waiting list only for a maximum number of rounds which covers one year. Indeed, after one year the patient may consider a different treatment or may have deceased (Benedek *et al.*, 2021). Usually, the programmes are run quarterly, hence, a pair entering the programme at round t exits it after round

$t+3$, if it has never been served. The donors of the partially served pairs at round t need to be considered as altruistic donors in the next round $t+1$. Hence, the system evolves from a round $t = 1, \ldots, T-1$ to a round $t+1$ in the following manner:

- altruistic donors and pairs newly disclosed at round $t+1$ enter the system;
- all served pairs and exploited altruistic donors at round t exit the system and are not considered at round $t+1$;
- all partially served pairs at round t enter the system as altruistic donors at round $t+1$;
- all unserved pairs at round t, entering the system at rounds $t-2$, $t-1$ or t are considered at round $t+1$, the ones entering the system before round $t-2$ are not considered at round $t+1$;
- all unexploited altruistic donor at round t are considered at round $t+1$.

The aim of the *Iterative International Kidney Exchange Problem* (IIKEP) is to determine an exchange scheme X^t at each round $t = 1, \ldots, T$ such that the disparities between the distribution of the medical benefit among the countries is minimised. In the context of the IIKEP, minimising the disparities between the countries encompasses two elements. First, at each round, the system should be stable. Hence, the stability is an intra-round condition which is modelled as in Section 6.3.2, in the context of the IKEP. Then, the system should be fair in sharing the extra benefit incurred by the countries when joining the programme. We follow the idea of Klimentova *et al.* (2021) to incorporate the fairness in the system, but here, such fairness is expressed in terms of medical benefit associated with the transplants rather than the number of transplants. Precisely, at each round $t = 1 \ldots, T$, given an ideal medical benefit γ_h^t for each country $h \in \mathcal{H}$, we define a fair target medical benefit τ_h^t which considers past information. For each country $h \in \mathcal{H}$, value τ_h^t can be defined recursively as follows:

$$\begin{cases} \tau_h^1 = \gamma_h^1 \\ \tau_h^t = \gamma_h^t + \tau_h^{t-1} - Z_h^{t-1}, \end{cases} \tag{6.1}$$

where we denoted by Z_h^{t-1} the medical benefit obtained by country h in the exchange scheme X^{t-1} at round $t-1$, i.e., $Z_h^{t-1} = \sum_{e \in X^{t-1}} W_h^{(t-1)e}$, where $W_h^{(t-1)e} = \sum_{(i,j) \in \mathcal{A}(e): j \in \mathcal{J}_h^{(t-1)}} W_{ij}$ is the sum of the medical benefits associated with incompatible

pairs of country h involved in exchange e. By developing recursive relation (6.1) for each country $h \in \mathcal{H}$, we obtain:

$$\tau_h^t = \sum_{s=1}^{t} \gamma_h^s - \sum_{s=1}^{t-1} Z_h^s. \tag{6.2}$$

Hence, the target fair allocation for country h at round t is equal to the cumulative ideal medical benefits for country h up to round t minus the cumulative medical benefits really obtained for country h in the past.

In the following, we present how the ideal medical benefit γ_h^t for country $h \in \mathcal{H}$ at round $t = 1, \ldots, T$ can be computed. We make use of the benefit value as underlying fairness measure as proposed by Klimentova *et al.* (2021). Indeed, it is easy to compute and Klimentova *et al.* (2021) report good results using this value.

As in Section 6.3.2, we model the problem arising at each round as a cooperative game with transferable utility to define the fairness measures. We add an index t to the notation of the game to stress its dependency on the round, i.e., we write $\Gamma^t = (\mathcal{H}, v^t)$ where $v^t(\mathcal{S})$, $\mathcal{S} \subseteq \mathcal{P}(\mathcal{H})$ is the value of the KEP on compatibility graph $\mathcal{G}_\mathcal{S}^t = (\mathcal{V}_\mathcal{S}^t = \bigcup_{h \in \mathcal{S}} \mathcal{V}_h^t, \mathcal{A}_\mathcal{S}^t)$.

The fairness measure we consider, i.e., the benefit value, allows to allocate the surplus in the medical benefit obtained when solving the KEP for all countries w.r.t. solving the KEP individually for each country. At each round $t = 1, \ldots, T$, value γ_h^t is the medical benefit $v^t(\{h\})$ obtained by country h individually, plus a share of the total surplus computed according to a fairness measure f, i.e.,

$$\gamma_h^t = v^t(\{h\}) + \left(v^t(\mathcal{H}) - \sum_{k \in \mathcal{H}} v^t(\{k\}) \right) \frac{f_h^t}{\sum_{k \in \mathcal{H}} f_k^t}. \tag{6.3}$$

For each country $h \in \mathcal{H}$, value f_h^t represents a measure of the "power" of country h in the grand coalition. Such measure is computed as the benefit value (see Section 6.2):

$$f_h^t = v^t(\mathcal{H}) - v^t(\mathcal{H} \setminus \{h\}) - v^t(\{h\}). \tag{6.4}$$

As done by Benedek *et al.* (2021), the Shapley value and the contribution value introduced in Section 6.2 could also be considered as fairness measures in the computation of allocations γ_h^t, $h \in \mathcal{H}$.

6.4 Formulation for the IKEP with stability and fairness in a single round

In this section, we propose a mathematical formulation for the IKEP arising at each round $t = 1, \ldots, T$. Such formulation is based on the formulation for the KEP introduced by Pansart *et al.* (2022), and is enriched to consider the stability and fairness conditions of the IIKEP.

For each round $t = 1, \ldots, T$, let $\mathcal{E}^t$ be the set of exchanges in graph $\mathcal{G}^t$, and let a_i^e be a binary parameter equal to one if vertex $i \in \mathcal{V}^t$ is involved in exchange $e \in \mathcal{E}^t$ and zero otherwise. For each round $t = 1, \ldots, T$ and each exchange $e \in \mathcal{E}^t$, we define a binary variable λ_e taking value 1 if exchange e is part of the exchange scheme X^t and 0 otherwise. Let ϵ be the non-negative continuous variable used to model the epsilon-core-like condition. Let δ_h be the non-negative continuous variable which registers the distance between the medical benefit effectively allocated to country $h \in \mathcal{H}$ in exchange scheme X^t and the fair target allocation τ_h^t which considers the past rounds.

Formulation $[\mathrm{P}_t]$ for the IKEP with stability and fairness arising at round t reads as follows:

$$[\mathrm{P}_t] \quad \min \ \epsilon + \sum_{h \in \mathcal{H}} \delta_h \tag{6.5}$$

$$\text{s.t.} \sum_{e \in \mathcal{E}^t} a_i^e \lambda_e \leq 1 \qquad \forall i \in \mathcal{V}^t \tag{6.6}$$

$$\sum_{h \in \mathcal{S}} \sum_{e \in \mathcal{E}^t} W_h^{te} \lambda_e \geq v^t(\mathcal{S}) \qquad \forall \mathcal{S} \subseteq \mathcal{H}, |\mathcal{S}| = 1 \tag{6.7}$$

$$\sum_{h \in \mathcal{S}} \sum_{e \in \mathcal{E}^t} W_h^{te} \lambda_e \geq v^t(\mathcal{S}) - \epsilon \qquad \forall \mathcal{S} \subseteq \mathcal{H}, |\mathcal{S}| > 1 \tag{6.8}$$

$$\sum_{e \in \mathcal{E}^t} W_h^{te} \lambda_e + \delta_h \geq \tau_h^t \qquad \forall h \in \mathcal{H} \tag{6.9}$$

$$\sum_{e \in \mathcal{E}^t} W_h^{te} \lambda_e - \delta_h \leq \tau_h^t \qquad \forall h \in \mathcal{H} \tag{6.10}$$

$$\lambda_e \in \{0, 1\} \qquad \forall e \in \mathcal{E}^t \tag{6.11}$$

$$\delta_h \geq 0 \qquad \forall h \in \mathcal{H} \tag{6.12}$$

$$\epsilon \geq 0. \tag{6.13}$$

Objective function (6.5) minimizes ϵ variable in the ϵ-core conditions and the dis-

tance from the fair target allocation τ_h^t, $h \in \mathcal{H}$. Packing constraints (6.6) ensure that each incompatible pair or altruistic donor is involved in at most one exchange. Constraints (6.7) impose that each single country is allocated with a medical benefit which is at least as good as the one it could get on its own. Constraints (6.8) are the ϵ-core like constraints and permit to ensure the stability of the system. Constraints (6.9) and (6.10) define variables δ_h as the deviation of the medical benefit effectively allocated to country h from the fair target allocation τ_h^t initially computed. Finally, Constraints (6.11), (6.12) and (6.13) define variables domains.

When considering the linear relaxation of formulation [P$_t$], i.e., when Constraints (6.11) become $\lambda_e \geq 0$ for all $e \in \mathcal{E}^t$, we denote by $\pi_i \leq 0$ the dual variables associated with Constraints (6.6), by $\phi_{\mathcal{S}} \geq 0$ the ones associated with Constraints (6.7) and (6.8) and by $\theta_h^1 \geq 0$ and $\theta_h^2 \leq 0$ the ones associated with Constraints (6.9) and (6.10), respectively. We report the dual of the linear relaxation of formulation [P$_t$]:

$$[\mathrm{D}_t] \quad \max \sum_{i \in \mathcal{V}^t} \pi_i + \sum_{\mathcal{S} \in \mathcal{P}(\mathcal{H})} v^t(\mathcal{S}) \phi_{\mathcal{S}} + \sum_{h \in \mathcal{H}} \tau_h^t (\theta_h^1 + \theta_h^2) \tag{6.14}$$

$$\text{s.t.} \sum_{i \in \mathcal{V}^t} a_i^e \pi_i + \sum_{\mathcal{S} \in \mathcal{P}(\mathcal{H})} \sum_{h \in \mathcal{S}} W_h^{te} \phi_{\mathcal{S}} + \sum_{h \in \mathcal{H}} W_h^{te} (\theta_h^1 + \theta_h^2) \leq 0 \qquad \forall e \in \mathcal{E}^t \tag{6.15}$$

$$\sum_{\mathcal{S} \in \mathcal{P}(\mathcal{H}), |\mathcal{S}| > 1} \phi_{\mathcal{S}} \leq 1 \tag{6.16}$$

$$\theta_h^1 - \theta_h^2 \leq 1 \qquad \forall h \in \mathcal{H} \tag{6.17}$$

$$\pi_i \leq 0 \qquad \forall i \in \mathcal{V}^t \tag{6.18}$$

$$\theta_h^1 \geq 0, \theta_h^2 \leq 0 \qquad \forall h \in \mathcal{H} \tag{6.19}$$

$$\phi_{\mathcal{S}} \geq 0 \qquad \forall \mathcal{S} \subseteq \mathcal{H}. \tag{6.20}$$

Note that Constraints (6.15) allow to express the reduced cost of variables λ_e, $e \in \mathcal{E}^t$.

6.5 Solution procedure for the IIKEP

In this section, we briefly describe the solution procedure for the IIKEP. Algorithm 11 displays the pseudocode of such solution procedure. The procedure is iterative and the iterations correspond to the rounds $t = 1, \ldots, T$. At each round t, compatibility graph $\mathcal{G}^t$ is built and formulation [P$_t$] is solved by means of the BPC algorithm that we will describe in Section 6.5.1. Remark that to build formulation [P$_t$], we need to

solve $|\mathcal{P}(\mathcal{H})|$ KEPs, one per each coalition of players, to compute the right hand-sides of Constraints (6.7) and (6.8). The KEPs are solved thanks to the BPC algorithm described in Chapter 5. Some of those values are also needed to compute fair target medical benefits τ_h^t, $h \in \mathcal{H}$.

Algorithm 11: Solution procedure for the IIKEP.

```
for t = 1, ..., T do
    build compatibility graph G^t as described in Section 6.3;
    foreach S ∈ P(H) do
        build compatibility graph G^t_S;
        compute value v^t(S) = KEP(S) by solving the KEP on compatibility graph G^t_S by means of the
          BPC algorithm described in Chapter 5;
    end
    foreach h ∈ H do
        compute value τ^t_h as described in Section 6.3;
    end
    solve formulation [P_t] by means of the BPC algorithm described in Section 6.5.1;
end
```

6.5.1 Branch-Price-and-Cut algorithm to solve formulation $[P_t]$

Formulation $[P_t]$ involves the exponentially-many variables λ_e, $e \in \mathcal{E}^t$ and the exponentially-many Constraints (6.8). In this section, we adapt the BPC algorithm presented in Chapter 5 to solve formulation $[P_t]$. Such adaptation is trivial, indeed, the additional Constraints (6.7),(6.8),(6.9) and (6.10) do not modify the structure of the pricing problem, as we show in what follows. In addition, the size of the instances we consider allows to enumerate Constraints (6.8). We only need to show how the dual variables associated with Constraints (6.7),(6.8),(6.9) and (6.10) are incorporated in the definition of the reduced cost of the variables.

By looking at Constraints (6.15), the reduced cost of a λ_e, $e \in \mathcal{E}^t$ variable is:

$$\bar{C}_e = -\sum_{i \in \mathcal{V}^t} a_i^e \pi_i - \sum_{\mathcal{S} \in \mathcal{P}(\mathcal{H})} \sum_{h \in \mathcal{S}} W_h^{te} \phi_{\mathcal{S}} - \sum_{h \in \mathcal{H}} W_h^{te} (\theta_h^1 + \theta_h^2).$$

The Pricing Problem is [PP] $\min\{\bar{C}_e : e \in \mathcal{E}^t\}$. As shown in Chapter 5, [PP] can be decomposed in $|\mathcal{I}^t| + 1$ independent subproblems: subproblems [PP-C](i), $i \in \mathcal{I}^t$, price the variables associated with the circuits and subproblem [PP-P] prices the variables associated with the chains. In addition, all the subproblems can be formulated as *Elementary Longest Path Problem with Length Constraint* (ELPPLC) on given directed weighed graphs as defined in Chapter 5.

Now, recall that value W_h^{te} is defined as $W_h^{te} = \sum_{(i,j)\in\mathcal{A}(e):j\in\mathcal{I}_h^t} W_{ij}$, where $\mathcal{A}(e)$ is the set of arcs traversed by exchange e. Hence, the reduced cost can be reformulated as follows:

$$\bar{C}_e = -\sum_{i\in\mathcal{V}^t} a_i^e \pi_i - \sum_{\mathcal{S}\in\mathcal{P}(\mathcal{H})} \sum_{h\in\mathcal{S}} \sum_{\substack{(i,j)\in\mathcal{A}(e)\\ j\in\mathcal{I}_h^t}} W_{ij}\phi_{\mathcal{S}} - \sum_{h\in\mathcal{H}} \sum_{\substack{(i,j)\in\mathcal{A}(e)\\ j\in\mathcal{I}_h^t}} W_{ij}(\theta_h^1 + \theta_h^2).$$

It is, then, trivial to see that dual variables $\phi_{\mathcal{S}}$, θ_h^1 and θ_h^2 can be incorporated in the cost of the arcs of the graphs for which the ELPPLCs are solved.

No further modification needs to be applied to the BPC algorithm described in Chapter 5 to solve formulation $[\mathrm{P}_t]$.

6.6 Computational experiments

We implemented our BPC algorithm in C++ and compiled it in release mode under a 64-bit version of MS Visual Studio 2019. The (integer) linear programming models in the BPC algorithm are solved by GUROBI 9.5.2 (64-bit version). We run the experiments on a 64-bit Windows machine equipped with a Intel(R) Xeon(R) Silver 4214 processor with 24 cores hyper-threaded to 48 virtual cores, with a base clock frequency of 2.2 GHz, and 96 GB of RAM. We impose one hour time limit to solve formulations $[\mathrm{P}_t]$ at each round $t = 1, \ldots, T$. We allow a single thread for each run of the algorithm.

In this section, first, we present the characteristics of the instances we consider. Then, we analyse the impact of the stability conditions in the IKEP (in a single round) and of the fairness conditions in the IIKEP.

In the following, we will use the term *BPC algorithm* to refer to the algorithm described in Section 6.5.1 to solve formulation $[\mathrm{P}_t]$, and the term *BPC-KEP* to refer to the BPC algorithm described in Chapter 5 to solve the KEP.

The optimality gaps for formulations $[\mathrm{P}_t]$, $t = 1, \ldots, T$, are expressed in percentage and computed as $100((UB - LB)/LB)$, where LB and UB are the lower and upper bounds returned by the BPC algorithm.

6.6.1 Generation of the testbed

Following to some extent Benedek *et al.* (2021), we generate a testbed containing 40 instances. First, we build 10 instances for the KEP using the generator introduced by Delorme *et al.* (2022) and available at `https://wpettersson.github.io/kidney-webapp/#/`. Precisely, we generate five compatibility graphs with 1000 incompatible pairs and five others with 2000 incompatible pairs. All the graphs are characterised by 10% of altruistic donors. The medical parameters to establish the compatibility relations are set by choosing the option "Use recipient blood group distribution from paper" in the generator.

The maximal length of the cycles is set to $L^C = 3$, whereas the maximal length of the chains is set to $L^P = 7$.

Given these 10 instances for the KEP we build the 40 instances of the IIKEP as follows. First, we consider the weights on the arcs of the compatibility graphs are either all set to one or represent the medical benefit associated with the transplants. In other words, the underlying KEPs consider as objective either the maximisation of the number of transplants (`obj:#TR`) or the maximisation of the medical benefit (`obj:MB`). Second, we consider either four or eight countries ($|\mathcal{H}| \in \{4, 8\}$). Incompatible pairs and altruistic donors are randomly and equally distributed between the countries, i.e., the countries have the same number of pairs and altruistic donors. All the instances are characterised by 24 rounds ($T = 24$), that is we consider six years kidney exchange programs run quarterly, as in Klimentova *et al.* (2021) and Benedek *et al.* (2021). The distribution of the disclosure of the incompatible pairs and altruistic donors is done as follows. In the first round, we disclose a tenth of the vertices of the compatibility graphs. The remaining vertices of the graphs are assigned to the remaining rounds in an arbitrary, but equal way.

6.6.2 Assessment of the stability conditions

In order to assess the stability conditions, we consider the results obtained by the BPC algorithm to solve formulation $[P_t]$ at the first round with only stability conditions. Indeed, stability is an intra-round condition, hence, there is no need to evaluate its impact over different rounds. Precisely, we consider the results obtained by solving formulation $[\mathrm{P}_1]$, where we removed variables and constraints associated with the fairness conditions, i.e., variables δ and Constraints (6.9), (6.10) and (6.12).

First, in Table 6.1, we report some computational statistics. The rows of the table correspond to the instances grouped by type of objective function considered in the KEP: either number of transplants (`#`), or medical benefit (`mb`). The column headings read as follows: `obj`: type of objective function considered in the KEP; #: the number of instances in the set; *#opt.*: number of instances for which formulation [P$_1$] is solved to optimality by the BPC algorithm; *avg.gap[%]*: average optimality gap in percentage returned by the BPC algorithm on formulation [P$_1$]; *avg.t[s]*: average computational time in seconds of the BPC algorithm on formulation [P$_1$]; *avg.t[s]* $v(\mathcal{H})$: average computational time in seconds for the BPC-KEP algorithm to compute grand coalition value $v(\mathcal{H}) = KEP(\mathcal{H})$ *avg.t[s]* $\bigcup_{\substack{\mathcal{S}\in\mathcal{P}(\mathcal{H})\\|\mathcal{S}|=1}} v(\mathcal{S})$: average computational time in seconds for the BPC-KEP algorithm to solve the KEPs for all the single countries, i.e., to compute all values $v(\mathcal{S}) = KEP(\mathcal{S})$, $\mathcal{S} \in \mathcal{P}(\mathcal{H})$, $|\mathcal{S}| = 1$.

Table 6.1: Computational statistics on solving formulation [P$_1$] with stability conditions.

Instances		BPC algorithm			BPC-KEP algorithm	
`obj`	#	#opt.	avg.gap[%]	avg.t[s]	avg.t[s] $v(\mathcal{H})$	avg.t[s] $\bigcup_{\substack{\mathcal{S}\in\mathcal{P}(\mathcal{H})\\|\mathcal{S}|=1}} v(\mathcal{S})$
#TR	20	20	0.00	84.91	0.91	0.28
MB	20	15	7.81	1527.23	1.45	0.28

From the results of Table 6.1, we observe that formulation [P$_1$] is much easier to solve when the objective of the KEP is the maximisation of the number of transplants. Indeed, the BPC algorithm solves to optimality formulation [P$_1$] on all the 20 instances in 85 seconds on average. This trend is not observed when the objective of the KEP is the maximisation of the medical benefit. In this case, the BPC algorithm solves to optimality 15 out of the 20 instances with more computational effort, being the average time equal to 1527.23 seconds. The five instances not solved to optimality are characterized by eight countries. The average optimality gap is on average nearly 8%. We recall that here the objective function of [P_1] with stability is the value of the decision variable ϵ.

Regardless from the objective function of the KEP, the value of the grand coalition $v(\mathcal{H})$ and the values of the single countries $v(\mathcal{S})$, $\mathcal{S} \in \mathcal{P}(\mathcal{H})$, $|\mathcal{S}| = 1$, are retrieved in negligible time (see last two columns of the table). The explanation is that, in those cases, the BPC-KEP algorithm needs to solve very small KEP instances. Indeed, the

compatibility graphs at the first round contain around a tenth of the vertices of the compatibility graph built for the IIKEP, that is, around 100 and 200 vertices.

In the following, we assess the goodness of the stability conditions. Precisely, we compare the results obtained by solving formulation [P$_1$] by means of the BPC algorithm with those obtained by solving the KEP associated with the grand coalition $\mathcal{H}$ by means of the BPC-KEP algorithm. The analysis is conducted by considering only the cases where formulation [P$_1$] is solved to optimality using the BPC algorithm.

We introduce some useful notation to interpret the results. Let $(\bar{\lambda}, \bar{\epsilon})$ be a solution of formulation [P$_1$] (recall that in this section, we do not consider variables δ). We say that a coalition $\mathcal{S} \in \mathcal{P}(\mathcal{H})$ is *unstable under* $\bar{\lambda}$ if value $\sum_{h\in\mathcal{S}} \sum_{e\in\mathcal{E}} W_h^e \bar{\lambda}_e$ assigned to coalition $\mathcal{S}$ under $\bar{\lambda}$ is strictly smaller than $v(\mathcal{S})$. Then, given $\bar{\lambda}$, we compute the percentage improvement w.r.t. the non collaborative setting where each country solves the KEP individually as

$$100\frac{\sum_{e\in\mathcal{E}} W_e\bar{\lambda}_e - \sum_{h\in\mathcal{H}} v(\{h\})}{\sum_{h\in\mathcal{H}} v(\{h\})}.$$

Now, by abuse of notation, let $\bar{\lambda}$ be a solution of $KEP(\mathcal{H})$ associated with the grand coalition $\mathcal{H}$. The definition of an unstable coalition under $\bar{\lambda}$ and the way we compute the improvement w.r.t. the non collaborative setting are still valid. In addition, it is possible to compute value $\bar{\epsilon}$ associated with $\bar{\lambda}$ also in the case where $\bar{\lambda}$ is a solution of $KEP(\mathcal{H})$:

$$\bar{\epsilon} = \max\left\{\max\left\{v(\mathcal{S}) - \sum_{h\in\mathcal{S}}\sum_{e\in\mathcal{E}} W_h^e\bar{\lambda}_e : \mathcal{S} \in \mathcal{P}(\mathcal{H})\right\}, 0\right\}.$$

Table 6.2 reports the results of the comparison between solving formulation [P_1] with stability conditions and solving KEP($\mathcal{H}$), that is with no stability conditions. Each row corresponds to a subset of instances characterised by the same objective function for the KEP and the same number of countries. The first three columns display information about the subset of instances: `obj`: type of objective function considered in the KEP; $|\mathcal{H}|$: number of countries; #: the number of instances in the subset. The next six columns report the results obtained with stability conditions (formulation [P_1]) against no stability conditions (KEP($\mathcal{H}$)) on three criteria: *avg. value of* $v(\mathcal{H})$: report information about the average value of the grand coalition $v(\mathcal{H})$, *avg. value of* ϵ: report information about the average value of ϵ, *avg.* #*unst. coal.*: report the average number of

unstable coalitions under solution $\bar{\lambda}$. The last five columns report the same information for the results obtained by the BPC-KEP algorithm when solving KEP($\mathcal{H}$).

Table 6.2: Assessment of the stability conditions.

Instances			avg. value of $v(\mathcal{H})$		avg. value of ϵ		avg. #unst. coal.	
obj	$\lvert\mathcal{H}\rvert$	#	stability	no stability	stability	no stability	stability	no stability
#TR	4	10	43.80	44.00	0.20	3.10	0.70	4.90
	8	10	43.50	44.00	0.70	4.50	33.20	90.90
MB	4	10	3714.60	3732.20	18.30	213.60	1.90	3.80
	8	5	4614.20	4643.00	29.20	362.20	18.40	77.60

The general remark by looking at the results of Table 6.2 is that the solutions obtained by considering stability conditions make the collaborative system much more stable, while keeping the number of transplants or medical benefit comparable with the case without stability conditions. Indeed, the decrease of the average value of $v(\mathcal{H})$ is negligible when considering stability conditions. In addition, the average value of ϵ and the average number of unstable coalitions decrease significantly when considering stability conditions.

6.6.3 Assessment of the fairness conditions

In this section, we assess how the fairness conditions impact the collaborative system. Recall that such conditions are inter-round, hence, to conduct this analysis we look at the results obtained by the iterative procedure for the IIKEP described in Section 6.5. First, we examine the results of the iterative procedure from a computational point of view and, then, we assess the impact of the fairness conditions.

Table 6.3 reports some computational information about the iterative procedure run over $T = 24$ rounds. Each row of the table represents a subset of the instances with the same objective for the KEP and the same number of countries. The first three columns report information about the instance: `obj`: type of objective function considered in the KEP; $|\mathcal{H}|$: number of countries; #: the number of instances in the subset. The last two columns show *avg.#opt.*$[P_t]$: the average number of rounds per instance where the BPC algorithm solves to optimality formulation $[\mathrm{P}_t]$, $t = 1, \ldots, T$; *#opt.inst.*: the number of instances where formulation $[\mathrm{P}_t]$ is solved to optimality at each round $t = 1, \ldots, T$.

The results of Table 6.3 show that the BPC algorithm solve to optimality formulation $[\mathrm{P}_t]$ for on average 19 out of the 24 rounds which characterise the instances of the

Table 6.3: Computational statistics when solving the IIKEP.

Instances			results	
obj	$\lvert\mathcal{H}\rvert$	#	avg.#opt.[P_t]	#opt.inst.
#TR	4	10	19.70	0
	8	10	19.00	1
MB	4	10	19.40	2
	8	10	14.50	0

IIKEP. However, the number of instances where formulation [P_t] is solved to optimality at each round $t = 1, \ldots, T$ is very small.

Here, we do not discuss again the stability conditions, indeed, the behaviour observed in Section 6.6.2 at the first round, repeats in the following rounds, as well. The values of variable ϵ which measure the instability of the system are small w.r.t. the values $v(\mathcal{H})$ of the KEP associated with the grand coalition. In addition, adding both stability and fairness conditions do not penalise much the objective of the KEP: its deviation from value $v(\mathcal{H})$ is on average equal to 1.61% for the instances with `obj:#TR` and to 4.52% for the instances with `obj:MB`.

In the following, we assess the impact of the fairness conditions. To do so, for each instance, each round $t = 1, \ldots, T$ and each country $h \in \mathcal{H}$, we compute the deviation of the medical benefit obtained by the country h at round t from the ideal medical benefit of that country at round t:

$$dev_h^t = \frac{|\gamma_h^t - Z_h^t|}{\gamma_h^t}.$$

In addition, for each instance and each period $t = 1, \ldots, T$, we compute the average of the above defined deviations over the countries:

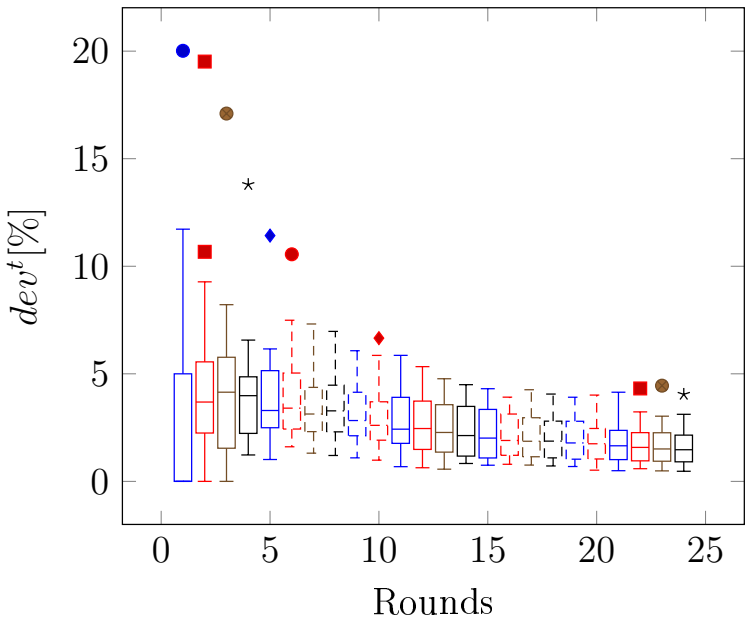
$$dev^t = \frac{1}{|\mathcal{H}|} \sum_{h \in \mathcal{H}} dev_h^t.$$

Precisely, we write $dev_h^t[\%]$ and $dev^t[\%]$ meaning that such deviations are expressed in percentage.

Figures 6.1 and 6.2 display the distribution of values $dev^t[\%]$ for instances where the objective of the KEP is the maximisation of the number of transplants (`obj:#TR`) and the maximisation of the medical benefit (`obj:MB`), respectively. Precisely, in the two

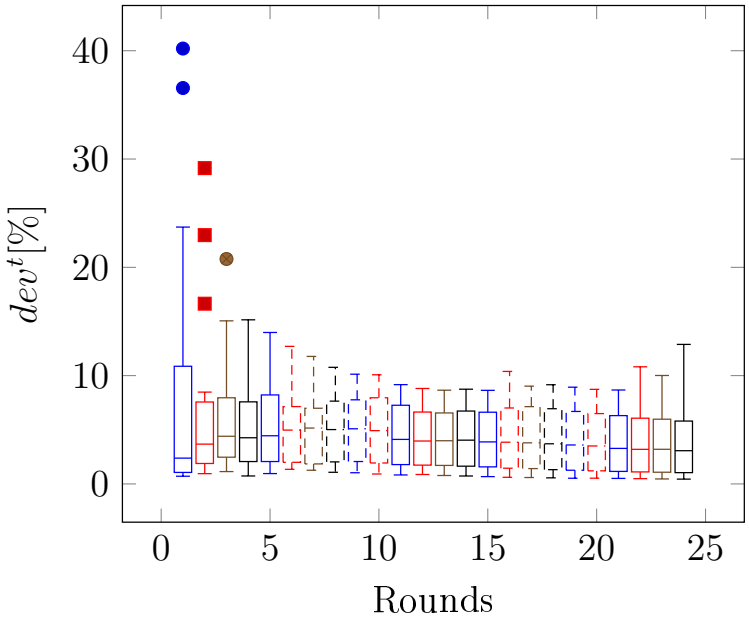
figures, the abscissas represent the rounds and the ordinates the magnitude of values $dev^t[\%]$. In each figure, for each round $t = 1, \ldots, T$, the distribution of values $dev^t[\%]$ for the 20 instances is graphically represented through a *box-and-whisker plot* (Tukey *et al.*, 1977). The elements of a box-and-whisker plot are: *end of the lower whisker*: the lowest of the values, outliers excluded; *bottom edge of the box*: first quartile of the values; *edge inside the box*: median of the values; *upper edge of the box*: third quartile of the values; *end of the upper whisker*: highest value of the data, outliers excluded; *symbols beyond the whiskers*: outliers (see circle, squares, stars, diamonds), i.e., values which differ significantly from the data.

Figure 6.1: Distribution of values $dev^t[\%]$ over the 24 rounds among the 20 instances with `obj:#TR`.



From Figure 6.1, the global trend is clear: the average deviation decrease over the rounds, with less variability between the instances. Indeed, in the first rounds, the deviations fluctuate heavily and some outliers where the average deviation is large appear, as well. This means that some instances provide solutions with a very high average deviation at the first rounds. Starting from the tenth round, the average deviations tend to stabilise in terms of variability and their maximal values (end of the upper whiskers) are always less than five percent. In the last rounds, the box-and-whisker plots reveal that the average deviations are of similar and small magnitude (see the boxes).

Figure 6.2: Distribution of values $dev^t[\%]$ over the 24 rounds among the 20 instances with `obj:MB`.



A similar behaviour can be observed in Figure 6.2 for the instances where the objective of the KEP is the maximisation of the medical benefit (`obj:MB`). The main difference that we can observe is that the average percentage of deviations tend to have larger values when maximising the medical benefit in comparison with the maximisation of the number of transplants.

For a complete understanding of the results, we stress that in some cases the percentage deviations of some countries might not reach zero. This behaviour is intrinsic to our integrated approach and can be explained by looking at the definition of the ideal target of transplants γ_h^t. At each period $t = 1, \ldots, T$, such values are computed in terms of value $v(\mathcal{H})$, i.e., the ideal medical benefit the grand coalition can guarantee itself without any stability condition (see Equations (6.3) and (6.4)). However, the actual medical benefit the grand coalition can guarantee itself at period t according to formulation $[\mathrm{P}_t]$ is $\sum_{h\in\mathcal{H}}\sum_{e\in\mathcal{E}^t} W_h^{te}\bar{\lambda}_h$, which may be smaller than $v(\mathcal{H})$ (see Constraints (6.8) and results in columns *avg.impr.[%] no collab.* of Table 6.2). As a consequence, at period t, the sum of the medical benefits obtained by the countries can be smaller than the sum of the ideal medical benefits, i.e. $\sum_{h\in\mathcal{H}} Z_h^t < \sum_{h\in\mathcal{H}} \gamma_h^t$. Hence, these discrepancies may propagate over the periods, making impossible for the deviations to be close to zero. A procedure to adjust the values of γ_h^t according to the

actual medical benefit obtained by the solution of formulation $[P_t]$ could be investigated to avoid such positive deviations.

Figures 6.1 and 6.2 show the behaviour of the average deviation over the countries. In what follows, we consider some instances to analyse more in details the disparities between the countries. Figures 6.3 and 6.4 show the trend of the percentage deviations dev_h^t for each country $h \in \mathcal{H}$ over the rounds $t = 1, \ldots, T$ on four representative instances. Precisely, Figure 6.3 considers two instances where the objective of the KEP is the maximisation of the number of transplants and with four or eight countries. Whereas, Figure 6.4 considers two instances where the objective of the KEP is the maximisation of the medical benefit and with four or eight countries. In each figure, each country is represented by a color, and the trend of values dev_h^t for country $h \in \mathcal{H}$ is represented by a piece-wise linear function over the rounds.

Figure 6.3: Assessment of the fairness in two instances with `obj:#TR` and four countries (left) or eight countries (right).

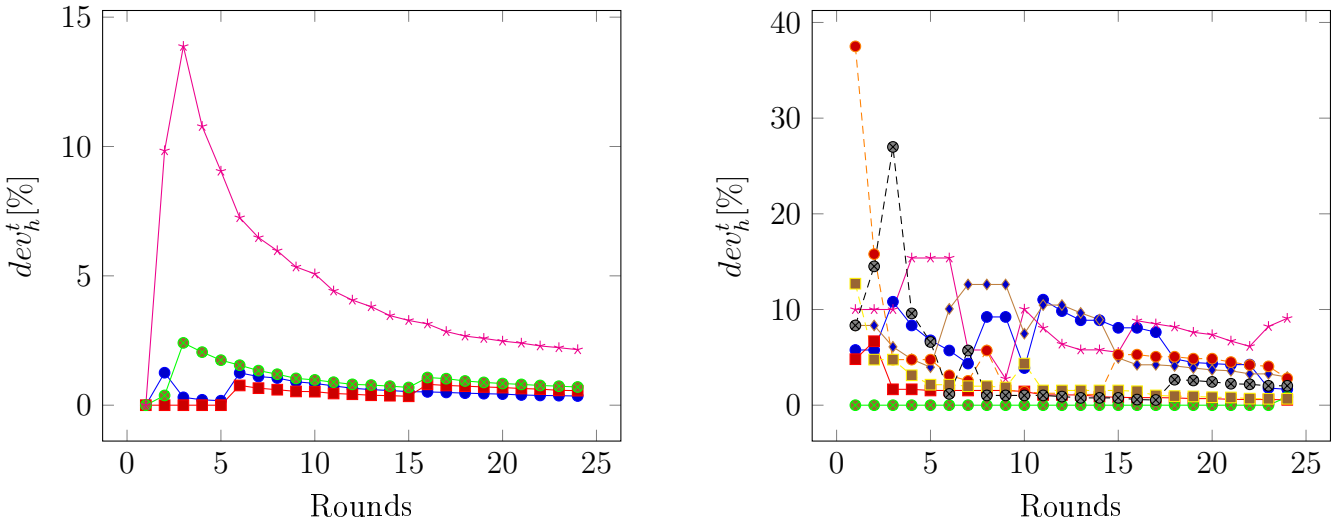
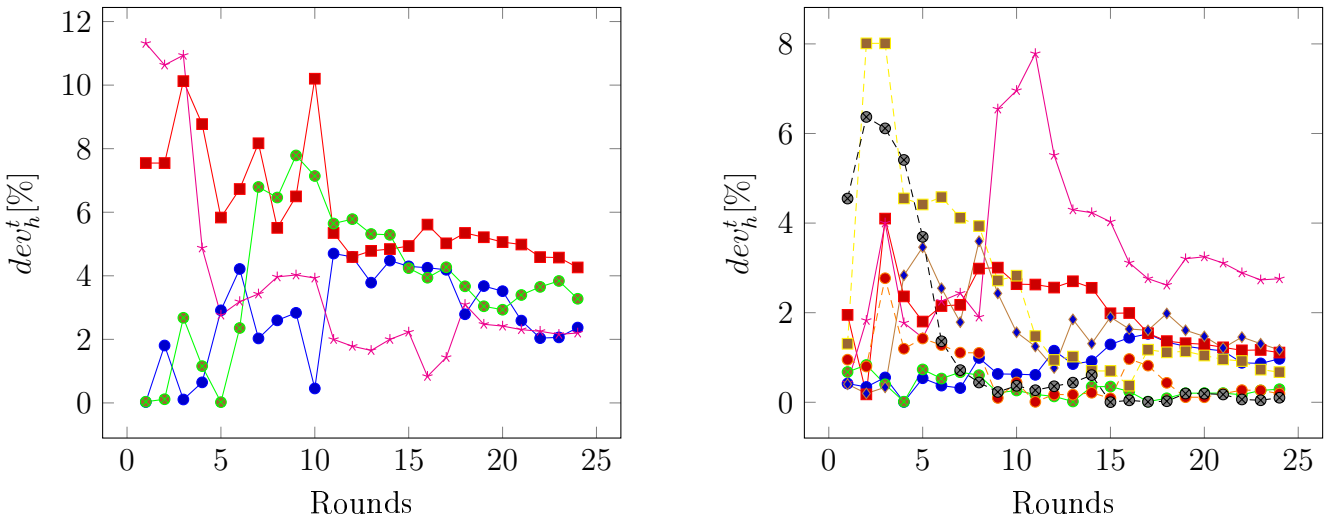


Figure 6.4: Assessment of the fairness in two instances with `obj:MB` and four countries (left) or eight countries (right).



From the plots of Figures 6.3 and 6.4, we can observe how the disparities of the deviations between the countries smooth over the rounds. Indeed, in the early rounds, the variability of the percentage deviation is high. Such variability reduces towards the last rounds of the programme, meaning that the countries are treated more equally.

6.7 Conclusions

In this paper, we studied the Iterative International Kidney Exchange Problem (IIKEP) which arises when different countries agree on a common kidney exchange programme run for multiple rounds. Under this collaborative setting, the objective of the IIKEP is to determine a set of kidney transplants at each round of the programme which minimises the disparities between the countries, that is the stability and the fairness of the collaborative system. We modeled the stability as an intra-round condition by borrowing the concept of ϵ-core from the cooperative game theory and the fairness as an inter-round condition similarly to what is proposed by Klimentova *et al.* (2021).

The nature of the problem entails an iterative solution approach. The problem arising at each round is a Kidney Exchange Problem (KEP) enriched with the aforementioned stability and fairness conditions. We formulated such problem by means of a set packing formulation with exponentially-many variables and constraints. The

former are required because in our underlying KEP we consider both cycles and long chains of donations. The latter are induced by the stability conditions. We adapted a Branch-Price-and-Cut (BPC) algorithm for the KEP to solve the problem at each round.

We performed experimental analysis to assess the impact of the stability and fairness conditions we impose in the system. Precisely, we showed that at each round and in the long run the solutions provided by our approach reduce the disparities between the countries w.r.t. the case where such conditions are not considered. In addition, when stability and fairness conditions are included the results demonstrate that the total medical benefit or number of transplants decreases slightly (3.03% on average) in comparison with the case without neither stability nor fairness.

As future work, a more extensive set of experiments may help in a deeper interpretation of the results. In addition, another research direction is related to the modelling choices, such as the necessity of including all the exponentially many constraints to ensure stability for all possible coalition of countries, or the consideration of weights in the objective function to balance stability and fairness. Furthermore, the results suggest that some enhancements for the BPC algorithm should be considered. Indeed, the BPC algorithm does not systematically solve to optimality the problem at each round. The inclusion of problem specific valid inequalities may help in the solution procedure. Moreover, different choices to model the ideal medical benefit in the fairness conditions may be taken into account, such as the Shapley value or the contribution value.

Conclusions and Perspectives

Conclusions

In this thesis, we studied heuristic algorithms with a performance guarantee or exact methods based on column generation for (integrated) optimisation problems arising in transportation and healthcare. Albeit the fields of applications are different, the problems we consider share a common structure. Precisely, they entail finding a set of optimal paths in a weighted directed graph which may be subject to resource and structural constraints.

In transportation applications, we consider a problem arising in a two-echelon distribution system where multiple commodities are traded between suppliers, distribution centres and customers. In such a system, not all the suppliers provide the same commodities and customer demands are composed of different commodities, as well. In the collection echelon, capacitated vehicles bring the commodities from the suppliers to the distribution centres via direct trips. Differently, in the delivery echelon, each distribution centre owns a fleet of capacitated vehicles which perform routes to deliver the commodities to the customers. All the vehicles involved in the system are flexible, i.e., they can transport any subset of commodities. In addition, customers can be visited by several vehicles to reduce the transportation costs. However, for the customers' convenience, the amount of a single commodity is delivered by one vehicle only. The name of this problem is the *Multi-Commodity two-echelon Distribution Problem* (MC2DP). We also study the core optimisation problem in the MC2DP, the *Commodity constrained Split Delivery Vehicle Routing Problem* (C-SDVRP). The C-SDVRP corresponds to the restriction of the MC2DP to a single distribution centre in the delivery echelon.

In healthcare applications, we consider a problem arising when multiple countries join a common kidney exchange programme. The objective of such programmes is to determine the best set of kidney transplants in a pool of patients and donors where the

term best refers to the medical benefit of the transplants. Such optimisation problem is referred to as *Kidney Exchange Problem* (KEP). Commonly, the KEP is solved by each country individually in a multi-period setting. In recent years, countries start to solve the KEP jointly to obtain better global solutions. However, such solutions may hide disparities between the countries which may weaken the soundness of the system. For this reason, we study the *Iterative International Kidney Exchange Problem* (IIKEP). The aim of the IIKEP is to minimise such disparities by including stability and fairness conditions when the KEP is solved jointly and under a multi-period setting. We model the stability as an intra-period condition and fairness as an inter-period condition by considering concept from cooperative game theory. As for the transportation application, we study both the IIKEP (integrated problem) and the KEP (core problem).

The main contributions of this thesis are listed in the following.

In Chapter 2, we provide an overview on BPC algorithms to solve a generic class of problems where the set of best paths, according to a given objective function, are to be determined in a weighted directed graph. Such paths may be subject to resource and structural constraints such as the elementarity constraints. The objective of the chapter is to provide the tools to develop a first BPC algorithm. First, the mathematical background needed to understand how the BPC algorithms work is presented. Then, the main techniques to embed in an efficient BPC algorithm are presented from a pedagogical point of view, i.e., with examples and implementation details.

In Chapter 3, we propose a heuristic with performance guarantee for the C-SDVRP based on a column generation procedure. We devise a novel pricing heuristic which exploits the multi-commodity aspect of the problem. Such heuristic helps in reducing effectively the computational time on a sub-class of the problem instances. Compared with the state-of-the-art metaheuristic algorithms for the C-SDVRP, our approach produces solutions of comparable quality, while improving the computational time.

In Chapter 4, we develop the first BPC algorithm for the MC2DP. Several state-of-the-art accelerating techniques are embedded and three families of valid inequalities are considered to strengthen the linear relaxation. Two of these families come from the literature, namely, the capacity cuts and valid inequalities arising from the set covering polytope. The third family is new and is based on the number partitioning polytope. Our approach solves to optimality nearly 60% of the benchmark instances from the literature. The remaining ones are left with a small optimality gap, on average equal to 2.1%.

In Chapter 5, we develop an exact algorithm for the KEP based on a BPC approach. The BPC incorporates two families of non-robust valid inequalities, namely, the subset-row and odd-hole inequalities and is able to solve instances with long cycles and chains. It is the first approach to be tested on three sets of benchmark instances from the literature. Our approach outperforms two state-of-the-art algorithms for the KEP on two sets of instances and provides comparable results against the third one.

In Chapter 6, we consider the IIKEP, a problem arising when different countries solve the KEP jointly under a multi-period setting. We exploit the theory of collaborative games with transferable utility to impose stability and fairness constraints for the system. At each period, the IKEP is solved by an adaptation of the BPC algorithm for the KEP presented in Chapter 5. Preliminary computational experiments show that the stability and fairness constraints lead to a more suitable system, i.e., they smooth the disparities between the countries.

Perspectives

In the following, we present possible future work directions for the problems and methods considered in this thesis.

From the application point of view, in transportation, one might consider to extend the MC2DP by allowing routing decisions also in the collection echelon. This choice could model a situation characterised by many more suppliers which provide smaller amounts of products. Here, the suppliers may have interest in collaborating with each other: some of them may be in charge of collecting the products from other suppliers and bring them to the closest distribution centre. Another extension for the problem entails adding connections between the actors in the integrated system, by e.g., allowing direct trips from the suppliers to the customers and/or transshipment between the distribution centres. Direct connections between suppliers and customers are interesting in the context of a short and local supply chain when customers are located nearby the suppliers. Transshipment between distribution centres could help in the consolidation of the products before performing the deliveries. Under this setting, the extended formulation has to be modified by including variables representing routes in the collection echelon and the additional connections. Since the routes in the collection echelon are exponentially-many, they need to be priced in a BPC algorithm to solve the problem. Finally, it seems interesting to incorporate the time aspect in the two-echelon

distribution system. Indeed, time windows may be considered for both suppliers and customers. Moreover, the collection and delivery operations may be synchronised at the distribution centres by explicitly considering the time. To do so, the time at which the vehicles in charge of the collection visit the distribution centres must be known. Furthermore, the vehicles in charge of the delivery start their operations only after the commodities they deliver have been brought to the distribution centres. Such version of the problem could still be modelled by means of an extended formulation which could be solved by a BPC algorithm. However, the synchronisation constraints force to introduce a time discretisation and additional binary variables to model when a collection vehicle arrives at a distribution centre and when a delivery vehicle leaves the distribution centre.

In healthcare application, kidney exchange programmes may suffer from data uncertainties. Indeed, such programmes plan the transplants for the three or four months ahead, hence, in such a long period data may change. For example, unexpected incompatibilities may arise or patients and/or donors may drop out. In addition, the medical benefit assigned to the transplants is modelled by a single value which may be affected by errors in its estimation. As per now, the literature involving uncertainties in the KEP considers instances with chains and cycles of small length or with a small number of patients and donors. Hence, a future research direction may be devoted to introduce such uncertainties in a more realistic KEP instances with long chains and cycles. Under this setting, we could consider a robust version of the problem or a two stage approach where in the first stage a solution for the deterministic problem is found and in the second stage some uncertainties are realised and recourse policies are applied to recover a solution.

Finally, the results presented in all our applications show that the bottleneck of the BPC algorithms we devised are usually the rather large optimality gaps at the root node which can be reduced only by exploring the branch-and-bound tree that is therefore usually quite large. Hence, from the methodological point of view, an enhancement for the BPC algorithms presented in this thesis may be related to the introduction of additional valid inequalities to strengthen the linear relaxation. For the MC2DP, we may consider including non-robust inequalities, classical such as the subset-row inequalities or new ones which exploits the structure of the problem. As an example, the robust inequalities we propose for the MC2DP may be considered in a non-robust manner. For the KEP, non-robust inequalities are considered, however, their

management in the BPC algorithm is to be improved. In addition, instead of separating the subset-row inequalities by enumeration of small-enough sets on which their are defined, one can opt for other separation procedures as, for example, efficient heuristics to detect violated clique inequalities in the conflict graph, which are a particular case of the subset-row inequalities. In addition, note that the inequalities we considered are valid for the stable set polytope. Other families of inequalities valid for the stable set polytope may be investigated.

Bibliography

Abraham, D.J., Blum, A. & Sandholm, T. (2007). Clearing algorithms for barter exchange markets: Enabling nationwide kidney exchanges. In *Proceedings of the 8th ACM conference on Electronic commerce*, 295–304. 120, 121

Achterberg, T. (2007). Constraint integer programming. 50, 100

Alinaghian, M. & Shokouhi, N. (2018). Multi-depot multi-compartment vehicle routing problem, solved by a hybrid adaptive large neighborhood search. *Omega*, **76**, 85–99. 56

Amor, H.B., Desrosiers, J. & de Carvalho, J.M.V. (2006). Dual-optimal inequalities for stabilized column generation. *Operations Research*, **54**, 454–463. 47

Amor, H.M.B., Desrosiers, J. & Frangioni, A. (2009). On the choice of explicit stabilizing terms in column generation. *Discrete Applied Mathematics*, **157**, 1167–1184. 47

Applegate, D., Bixby, R., Chvátal, V. & Cook, W. (1995). Finding cuts in the tsp (a preliminary report). Tech. rep., Citeseer. 50

Archetti, C., Savelsbergh, M.W.P. & Speranza, M.G. (2006a). Worst-case analysis for split delivery vehicle routing problems. *Transportation Science*, **40**, 226–234. 55

Archetti, C., Speranza, M.G. & Hertz, A. (2006b). A tabu search algorithm for the split delivery vehicle routing problem. *Transportation Science*, **40**, 64–73. 55

Archetti, C., Bianchessi, N. & Speranza, M. (2013). Optimal solutions for routing problems with profits. *Discrete Applied Mathematics*, **161**, 547–557. 48, 102, 135

Archetti, C., Bianchessi, N. & Speranza, M.G. (2014). Branch-and-cut algorithms for the split delivery vehicle routing problem. *European Journal of Operational Research*, **238**, 685–698. 55

Archetti, C., Bianchessi, N. & Speranza, M.G. (2015). A branch-price-and-cut algorithm for the commodity constrained split delivery vehicle routing problem. *Computers & Operations Research*, **64**, 1 – 10. 56, 59, 61, 62, 63, 66, 96, 97

Archetti, C., Campbell, A.M. & Speranza, M.G. (2016). Multicommodity vs. single-commodity routing. *Transportation Science*, **50**, 461–472. 55, 56, 70, 71, 73, 75, 83, 103

Arslan, A.N., Omer, J. & Yan, F. (2022). Kidneyexchange. jl: A julia package for solving the kidney exchange problem with branch-and-price. 118, 119, 122, 127, 129, 135, 136, 137, 138, 139, 149

Axelrod, D.A., Schnitzler, M.A., Xiao, H., Irish, W., Tuttle-Newhall, E., Chang, S.H., Kasiske, B.L., Alhamad, T. & Lentine, K.L. (2018). An economic assessment of contemporary kidney transplant practice. *American Journal of Transplantation*, **18**, 1168–1176. 117

Balas, E. & Ng, S.M. (1989). On the set covering polytope: I. all the facets with coefficients in {0, 1, 2}. *Mathematical Programming*, **43**, 57–69. 84, 90, 91

Baldacci, R., Mingozzi, A. & Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**, 1269–1283. 35, 36, 62, 97, 130

Barnhart, C., Johnson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P. & Vance, P.H. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, **46**, 316–329. 3, 18, 21, 94, 124, 206

Battarra, M., Golden, B. & Vigo, D. (2008). Tuning a parametric clarke–wright heuristic via a genetic algorithm. *Journal of the Operational Research Society*, **59**, 1568–1572. 69, 101

Beasley, J.E. & Christofides, N. (1989). An algorithm for the resource constrained shortest path problem. *Networks*, **19**, 379–394. 24, 29

Belenguer, J.M., Martinez, M.C. & Mota, E. (2000). A lower bound for the split delivery vehicle routing problem. *Operations Research*, **48**, 801–810. 55

Benedek, M., Biró, P., Kern, W. & Paulusma, D. (2021). Computing balanced solutions for large international kidney exchange schemes. *arXiv preprint arXiv:2109.06788*. 155, 156, 162, 164, 169

Benedek, M., Biró, P., Kern, W., Pálvölgyi, D. & Paulusma, D. (2023). Partitioned matching games for international kidney exchange. 156

Berti, G. & Mulligan, C. (2016). Competitiveness of small farms and innovative food supply chains: The role of food hubs in creating sustainable regional and local food systems. *Sustainability*, **8**, 616. 1, 2, 203, 204

Biró, P., Haase-Kromwijk, B., Andersson, T., Ásgeirsson, E.I., Baltesová, T., Boletis, I., Bolotinha, C., Bond, G., Böhmig, G., Burnapp, L. *et al.* (2019a). Building kidney exchange programmes in europe—an overview of exchange practice and activities. *Transplantation*, **103**, 1514. 2, 118, 153, 204

Biró, P., Kern, W., Pálvölgyi, D. & Paulusma, D. (2019b). Generalized matching games for international kidney exchange. 155, 161

Biró, P., van de Klundert, J., Manlove, D., Pettersson, W., Andersson, T., Burnapp, L., Chromy, P., Delgado, P., Dworczak, P., Haase, B., Hemke, A., Johnson, R., Klimentova, X., Kuypers, D., Costa, A.N., Smeulders, B., Spieksma, F., Valentín, M.O. & Viana, A. (2021). Modelling and optimisation in european kidney exchange programmes. *European Journal of Operational Research*, **291**, 447–456. 118, 153

Blanquart, C., Gonçalves, A., Vandenbossche, L., Kebir, L., Petit, C. & Traversac, J.B. (2010). The logistic leverages of short food supply chains performance in terms of sustainability. In *12th World Conference on Transport Research*, 10p. 1, 203

Bode, C. & Irnich, S. (2012). Cut-first branch-and-price-second for the capacitated arc-routing problem. *Operations Research*, **60**, 1167–1182. 34, 63, 97

Böhmig, G.A., Fronek, J., Slavcev, A., Fischer, G.F., Berlakovich, G. & Viklicky, O. (2017). Czech-austrian kidney paired donation: first european cross-border living donor kidney exchange. *Transplant International*, **30**, 638–639. 3, 153, 205

Boland, N., Dethridge, J. & Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, **34**, 58–68. 36

Bortfeldt, A. & Yi, J. (2020). The split delivery vehicle routing problem with three-dimensional loading constraints. *European Journal of Operational Research*, **282**, 545–558. 55

Breunig, U., Baldacci, R., Hartl, R. & Vidal, T. (2019). The electric two-echelon vehicle routing problem. *Computers & Operations Research*, **103**, 198–210. 85

Briant, O., Lemaréchal, C., Meurdesoif, P., Michel, S., Perrot, N. & Vanderbeck, F. (2008). Comparison of bundle and classical column generation. *Mathematical programming*, **113**, 299–344. 47

Briant, O., Cambazard, H., Cattaruzza, D., Catusse, N., Ladier, A.L. & Ogier, M. (2020). An efficient and general approach for the joint order batching and picker routing problem. *European Journal of Operational Research*, **285**, 497–512. 28, 57

Carvalho, M. & Lodi, A. (2023). A theoretical and computational equilibria analysis of a multi-player kidney exchange program. *European Journal of Operational Research*, **305**, 373–385. 155

Carvalho, M., Lodi, A., Pedroso, J.P. & Viana, A. (2016). Nash equilibria in the two-player kidney exchange game. *Mathematical Programming*, **161**, 389–417. 154, 155

Cattaruzza, D., Absi, N., Feillet, D. & González-Feliu, J. (2017). Vehicle routing problems for city logistics. *EURO Journal on Transportation and Logistics*, **6**, 51–79. 83

Chen, M.C., Hsiao, Y.H., Reddy, R.H. & Tiwari, M.K. (2016a). The self-learning particle swarm optimization approach for routing pickup and delivery of multiple products with material handling in multiple cross-docks. *Transportation Research Part E: Logistics and Transportation Review*, **91**, 208 – 226. 55

Chen, P., Golden, B., Wang, X. & Wasil, E. (2016b). A novel approach to solve the split delivery vehicle routing problem. *International Transactions in Operational Research*, **24**, 27–41. 55

Clarke, G. & Wright, J.W. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, **12**, 568–581. 68, 101

Constantino, M., Klimentova, X., Viana, A. & Rais, A. (2013). New insights on integer-programming models for the kidney exchange problem. *European Journal of Operational Research*, **231**, 57–68. 118, 120, 121, 134

Contardo, C., Cordeau, J.F. & Gendron, B. (2014). An exact algorithm based on cut-and-column generation for the capacitated location-routing problem. *INFORMS Journal on Computing*, **26**, 88–102. 42, 100

Costa, L., Contardo, C. & Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, **53**, 946–985. 8, 42

Costa, L., Contardo, C., Desaulniers, G. & Yarkony, J. (2022). Stabilized column generation via the dynamic separation of aggregated rows. *INFORMS Journal on Computing*, **34**, 1141–1156. 47

Crainic, T.G., Ricciardi, N. & Storchi, G. (2004). Advanced freight transportation systems for congested urban areas. *Transportation Research Part C: Emerging Technologies*, **12**, 119–137. 85

Crainic, T.G., Ricciardi, N. & Storchi, G. (2009). Models for evaluating and planning city logistics systems. *Transportation Science*, **43**, 432–454. 85

Crainic, T.G., Feliu, J.G., Ricciardi, N., Semet, F. & Van Woensel, T. (2023). 10. operations research for planning and managing city logistics systems. *Handbook on City Logistics and Urban Freight: 0*, 190–223. 83

Cuda, R., Guastaroba, G. & Speranza, M. (2015). A survey on two-echelon routing problems. *Computers & Operations Research*, **55**, 185 – 199. 85

Dantzig, G., Fulkerson, R. & Johnson, S. (1954). Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, **2**, 393–410. 26

de Aragao, M.P. & Uchoa, E. (2003). Integer program reformulation for robust branch-and-cut-and-price algorithms. In *Mathematical program in rio: a conference in honour of nelson maculan*, 56–61. 40

Dellaert, N., Woensel, T.V., Crainic, T.G. & Saridarq, F.D. (2021). A multi-commodity two-echelon capacitated vehicle routing problem with time windows: Model formulations and solution approach. *Computers & Operations Research*, **127**, 105154. 85, 86

Delorme, M., García, S., Gondzio, J., Kalcsics, J., Manlove, D., Pettersson, W. & Trimble, J. (2022). Improved instance generation for kidney exchange programmes. *Computers & Operations Research*, **141**, 105707. 136, 169

Delorme, M., Manlove, D. & Smeets, T. (2023). Half-cycle: A new formulation for modelling kidney exchange problems. *Operations Research Letters*, **51**, 234–241. vi, x, 120, 136, 137, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151

Desaulniers, G. (2010). Branch-and-price-and-cut for the split-delivery vehicle routing problem with time windows. *Operations Research*, **58**, 179–192. 50, 55, 101

Desaulniers, G., Desrosiers, J. & Solomon, M.M. (2006). *Column generation*, vol. 5. Springer Science & Business Media. 8

Desrochers, M. (1988). *An algorithm for the shortest path problem with resource constraints*. École des hautes études commerciales, Groupe d'études et de recherche en 25, 128

Desrochers, M., Desrosiers, J. & Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations research*, **40**, 342–354. 35

Desrosiers, J. & Lübbecke, M.E. (2005). A primer in column generation. In *Column Generation*, 1–32, Springer-Verlag. 8, 45, 60, 94, 124, 135

Di Puglia Pugliese, L. & Guerriero, F. (2013). A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, **62**, 183–200. 25, 65, 128

Dickerson, J.P., Manlove, D.F., Plaut, B., Sandholm, T. & Trimble, J. (2016). Position-indexed formulations for kidney exchange. In *Proceedings of the 2016 ACM Conference on Economics and Computation*, 25–42. 120

Drexl, M. (2012). Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science*, **46**, 297–316. 87

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, **42**, 977–978. 26

Dror, M. & Trudeau, P. (1989). Savings by split delivery routing. *Transportation Science*, **23**, 141–145. 55

Du Merle, O., Villeneuve, D., Desrosiers, J. & Hansen, P. (1999). Stabilized column generation. *Discrete Mathematics*, **194**, 229–237. 47

Edmonds, J. (1965). Maximum matching and a polyhedron with 0, 1-vertices. *Journal of research of the National Bureau of Standards B*, **69**, 55–56. 119

Enthoven, D.L., Jargalsaikhan, B., Roodbergen, K.J., uit het Broek, M.A. & Schrotenboer, A.H. (2020). The two-echelon vehicle routing problem with covering options: City logistics with cargo bikes and parcel lockers. *Computers & Operations Research*, **118**, 104919. 85

Federgruen, A. & Simchi-Levi, D. (1995). Analysis of vehicle routing and inventory-routing problems. *Handbooks in operations research and management science*, **8**, 297–373. 60, 90

Feillet, D. (2010). A tutorial on column generation and branch-and-price for vehicle routing problems. *4OR*, **8**, 407–424. 8, 15, 21

Feillet, D., Dejax, P., Gendreau, M. & Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**, 216–229. iv, 29, 30, 32, 62, 96, 129

Gilmore, P.C. & Gomory, R.E. (1961). A linear programming approach to the cutting-stock problem. *Operations research*, **9**, 849–859. 12

Glorie, K., Wagelmans, A., van de Klundert, J. *et al.* (2012). Iterative branch-and-price for large multi-criteria kidney exchange. *Econometric institute report*, **11**, 2012. 121

Glorie, K.M., van de Klundert, J.J. & Wagelmans, A.P. (2014). Kidney exchange with long chains: An efficient pricing algorithm for clearing barter exchanges with branch-and-price. *Manufacturing & Service Operations Management*, **16**, 498–512. 121

Goeke, D., Gschwind, T. & Schneider, M. (2019). Upper and lower bounds for the vehicle-routing problem with private fleet and common carrier. *Discrete Applied Mathematics*, **264**, 43–61. 34

Golden, B.L., Magnanti, T.L. & Nguyen, H.Q. (1977). Implementing vehicle routing algorithms. *Networks*, **7**, 113–148. 21

Grangier, P., Gendreau, M., Lehuédé, F. & Rousseau, L.M. (2016). An adaptive large neighborhood search for the two-echelon multiple-trip vehicle routing problem with satellite synchronization. *European Journal of Operational Research*, **254**, 80–91. 85

Gschwind, T. & Irnich, S. (2016). Dual inequalities for stabilized column generation revisited. *INFORMS Journal on Computing*, **28**, 175–194. 47

Gschwind, T., Bianchessi, N. & Irnich, S. (2019). Stabilized branch-price-and-cut for the commodity-constrained split delivery vehicle routing problem. *European Journal of Operational Research*, **278**, 91–104. 45, 50, 56, 62, 63, 70, 71, 73, 74, 96, 97, 99, 100

Gu, W., Cattaruzza, D., Ogier, M. & Semet, F. (2019). Adaptive large neighborhood search for the commodity constrained split delivery VRP. *Computers & Operations Research*, **112**, 104761. iv, ix, 56, 58, 61, 69, 70, 71, 73, 75, 77, 78, 79, 81

Gu, W., Archetti, C., Cattaruzza, D., Ogier, M., Semet, F. & Speranza, M.G. (2022). A sequential approach for a multi-commodity two-echelon distribution problem. *Computers & Industrial Engineering*, **163**, 107793. 83, 84, 85, 86, 93, 102, 103, 104, 105, 106, 107, 109, 110, 111, 112, 113

Guastaroba, G., Speranza, M.G. & Vigo, D. (2016). Intermediate facilities in freight transportation planning: A survey. *Transportation Science*, **50**, 763–789. 83

Gulczynski, D., Golden, B. & Wasil, E. (2010). The split delivery vehicle routing problem with minimum delivery amounts. *Transportation Research Part E: Logistics and Transportation Review*, **46**, 612–626. 55

Guo, F., Huang, Z. & Huang, W. (2021). Heuristic approaches for a vehicle routing problem with an incompatible loading constraint and splitting deliveries by order. *Computers & Operations Research*, **134**, 105379. 56

Haghani, N., Contardo, C. & Yarkony, J. (2022). Smooth and flexible dual optimal inequalities. *INFORMS Journal on Optimization*, **4**, 29–44. 47

Hintsch, T. & Irnich, S. (2020). Exact solution of the soft-clustered vehicle-routing problem. *European Journal of Operational Research*, **280**, 164–178. 28

Hoffman, K.L. & Padberg, M. (1993). Solving airline crew scheduling problems by branch-and-cut. *Management science*, **39**, 657–682. 130, 132

Irnich, S. & Villeneuve, D. (2006). The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing*, **18**, 391–406. 35

Jacobsen, S. & Madsen, O. (1980). A comparative study of heuristics for a two-level routing-location problem. *European Journal of Operational Research*, **5**, 378–387. 85

Jaumard, B., Semet, F. & Vovor, T. (1999). A two-phase resource constrained shortest path algorithm for acyclic graphs. *Cahiers du GERAD - G9648*. 26

Jepsen, M., Petersen, B., Spoorendonk, S. & Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**, 497–511. 42, 130

Jia, S., Deng, L., Zhao, Q. & Chen, Y. (2023). An adaptive large neighborhood search heuristic for multi-commodity two-echelon vehicle routing problem with satellite synchronization. *Journal of Industrial and Management Optimization*, **19**, 1187. 85, 86

Jie, W., Yang, J., Zhang, M. & Huang, Y. (2019). The two-echelon capacitated electric vehicle routing problem with battery swapping stations: Formulation and efficient methodology. *European Journal of Operational Research*, **272**, 879–904. 85

Kälble, T., Lucan, M., Nicita, G., Sells, R., Revilla, F.B. & Wiesel, M. (2005). Eau guidelines on renal transplantation. *European urology*, **47**, 156–166. 2, 117, 152, 205

Kelley, J.E., Jr (1960). The cutting-plane method for solving convex programs. *Journal of the society for Industrial and Applied Mathematics*, **8**, 703–712. 15, 18

King, R.P., Hand, M.S. & Gómez, M.I. (2014). *Growing local: Case studies on local food supply chains*. U of Nebraska Press. 1, 203

Klimentova, X., Alvelos, F. & Viana, A. (2014). A new branch-and-price approach for the kidney exchange problem. In *Computational Science and Its Applications–ICCSA 2014: 14th International Conference, Guimarães, Portugal, June 30–July 3, 2014, Proceedings, Part II 14*, 237–252, Springer. 120, 121, 134

Klimentova, X., Viana, A., Pedroso, J.P. & Santos, N. (2021). Fairness models for multi-agent kidney exchange programmes. *Omega*, **102**, 102333. 155, 156, 159, 162, 163, 164, 169, 177

Kovesdy, C.P. (2022). Epidemiology of chronic kidney disease: an update 2022. *Kidney International Supplements*, **12**, 7–11. 117

Kute, V.B., Prasad, N., Shah, P.R. & Modi, P.R. (2018). Kidney exchange transplantation current status, an update and future perspectives. *World journal of transplantation*, **8**, 52. 154

Lam, E. & Mak-Hau, V. (2020). Branch-and-cut-and-price for the cardinality-constrained multi-cycle problem in kidney exchange. *Computers & Operations Research*, **115**, 104852. 121

Land, A.H. & Doig, A.G. (1960). An automatic method of solving discrete programming problems. *Econometrica*, **28**, 497–520. 9

Laporte, G., Nobert, Y. & Desrochers, M. (1985). Optimal routing under capacity and distance restrictions. *Operations Research*, **33**, 1050–1073. 41, 84, 90

Lentine, K.L., Smith, J.M., Miller, J.M., Bradbrook, K., Larkin, L., Weiss, S., Handarova, D.K., Temple, K., Israni, A.K. & Snyder, J.J. (2023). Optn/srtr 2021 annual data report: Kidney. *American Journal of Transplantation*, **23**, S21–S120. 2, 117, 204

Levey, A.S. & Coresh, J. (2012). Chronic kidney disease. *The lancet*, **379**, 165–180. 117

Li, H., Liu, Y., Jian, X. & Lu, Y. (2018). The two-echelon distribution system considering the real-time transshipment capacity varying. *Transportation Research Part B: Methodological*, **110**, 239–260. 85

Li, H., Wang, H., Chen, J. & Bai, M. (2020). Two-echelon vehicle routing problem with time windows and mobile satellites. *Transportation Research Part B: Methodological*, **138**, 179–201. 85

Li, H., Chen, J., Wang, F. & Bai, M. (2021a). Ground-vehicle and unmanned-aerial-vehicle routing problems from two-echelon scheme perspective: A review. *European Journal of Operational Research*, **294**, 1078–1095. 85

Li, H., Wang, H., Chen, J. & Bai, M. (2021b). Two-echelon vehicle routing problem with satellite bi-synchronization. *European Journal of Operational Research*, **288**, 775–793. 85

Li, J., Xu, M. & Sun, P. (2022). Two-echelon capacitated vehicle routing problem with grouping constraints and simultaneous pickup and delivery. *Transportation Research Part B: Methodological*, **162**, 261–291. 84, 85

Libralesso, L. (2020). *Anytime tree search for combinatorial optimization*. Ph.D. thesis, Université Grenoble Alpes [2020-....]. 11

Lübbecke, M.E. & Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, **53**, 1007–1023. 8

Magnanti, T.L. & Wong, R.T. (1984). Network design and transportation planning: Models and algorithms. *Transportation Science*, **18**, 1–55. 86

Mak-Hau, V. (2015). On the kidney exchange problem: cardinality constrained cycle and chain problems on directed graphs: a survey of integer programming approaches. *Journal of Combinatorial Optimization*, **33**, 35–59. 121

Marques, G., Sadykov, R., Deschamps, J.C. & Dupas, R. (2020). An improved branch-cut-and-price algorithm for the two-echelon capacitated vehicle routing problem. *Computers & Operations Research*, **114**, 104833. 84, 89

Marques, G., Sadykov, R., Dupas, R. & Deschamps, J.C. (2022). A branch-cut-and-price approach for the single-trip and multi-trip two-echelon vehicle routing problem with time windows. *Transportation Science*. 84

Marsten, R.E. (1975). The use of the box step method in discrete optimization. Tech. rep., National Bureau of Economic Research. 47

Mattei, N. & Walsh, T. (2013). Preflib: A library for preferences http://www.preflib. org. In *Algorithmic Decision Theory: Third International Conference, ADT 2013, Bruxelles, Belgium, November 12-14, 2013, Proceedings 3*, 259–270, Springer. 136

Mhamedi, T., Andersson, H., Cherkesly, M. & Desaulniers, G. (2022). A branch-price-and-cut algorithm for the two-echelon vehicle routing problem with time windows. *Transportation Science*, **56**, 245–264. 84, 85

Miller, C.E., Tucker, A.W. & Zemlin, R.A. (1960). Integer programming formulation of traveling salesman problems. *Journal of the ACM*, **7**, 326–329. 26

Mirchandani, P.B. & Francis, R.L. (1990). *Discrete location theory*. 69

Mirzaei, S. & Wøhlk, S. (2019). A branch-and-price algorithm for two multi-compartment vehicle routing problems. *EURO Journal on Transportation and Logistics*, **8**, 1–33. 56, 57

Munari, P. & Savelsbergh, M. (2022). Compact formulations for split delivery routing problems. *Transportation Science*. 55

Muter, İ., Birbil, Ş.İ. & Şahin, G. (2010). Combination of metaheuristic and exact algorithms for solving set covering-type optimization problems. *INFORMS Journal on Computing*, **22**, 603–619. 57

Nakao, Y. & Nagamochi, H. (2007). A DP-based heuristic algorithm for the discrete split delivery vehicle routing problem. *Journal of Advanced Mechanical Design, Systems, and Manufacturing*, **1**, 217–226. 55

Neame, P.J. (2000). *Nonsmooth dual methods in integer programming*. Ph.D. thesis, University of Melbourne, Department of Mathematics and Statistics. 47

Nemati, E., Einollahi, B., Pezeshki, M.L., Porfarziani, V. & Fattahi, M.R. (2014). Does kidney transplantation with deceased or living donor affect graft survival? *Nephro-urology monthly*, **6**. 2, 117, 204

Padberg, M. & Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, **33**, 60–100. 16

Padberg, M.W. (1973). On the facial structure of set packing polyhedra. *Mathematical programming*, **5**, 199–215. 130

Pansart, L., Cambazard, H. & Catusse, N. (2022). Dealing with elementary paths in the kidney exchange problem. *arXiv preprint arXiv:2201.08446*. vi, x, 120, 121, 123, 125, 134, 136, 137, 139, 140, 141, 142, 145, 146, 147, 148, 149, 151, 165

Parragh, S.N. & Schmid, V. (2013). Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, **40**, 490–497. 57

Pecin, D., Pessoa, A., Poggi, M. & Uchoa, E. (2017). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, **9**, 61–100. 42, 131

Peleg, B. & Sudhölter, P. (2007). *Introduction to the theory of cooperative games*, vol. 34. Springer Science & Business Media. 158

Perboli, G., Tadei, R. & Vigo, D. (2011). The two-echelon capacitated vehicle routing problem: Models and math-based heuristics. *Transportation Science*, **45**, 364–380. 85

Pessoa, A., Sadykov, R., Uchoa, E. & Vanderbeck, F. (2018). Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, **30**, 339–360. 47, 48

Pessoa, A., Sadykov, R., Uchoa, E. & Vanderbeck, F. (2020). A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, **183**, 483–523. 8, 37, 49, 51, 100

Petris, M., Archetti, C., Cattaruzza, D., Ogier, M. & Semet, F. (2023). A heuristic with a performance guarantee for the Commodity constrained Split Delivery Vehicle Routing Problem, working paper or preprint. 50, 97, 101, 102

Plaut, B., Dickerson, J. & Sandholm, T. (2016a). Fast optimal clearing of capped-chain barter exchanges. In *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30. 121

Plaut, B., Dickerson, J.P. & Sandholm, T. (2016b). Hardness of the pricing problem for chains in barter exchanges. *arXiv preprint arXiv:1606.00117*. 121

Qiu, M., Fu, Z., Eglese, R. & Tang, Q. (2018). A tabu search algorithm for the vehicle routing problem with discrete split deliveries and pickups. *Computers & Operations Research*, **100**, 102–116. 56, 57

Ralphs, T., Kopman, L., Pulleyblank, W. & Trotter, L. (2003). On the capacitated vehicle routing problem. *Mathematical Programming*, **94**, 343–359. 68, 98

Rapaport, F. (1986). The case for a living emotionally related international kidney donor exchange registry. *Transplantation proceedings*, **18**, 5—9. 117

Riascos-Álvarez, L.C., Bodur, M. & Aleman, D.M. (2020). A branch-and-price algorithm enhanced by decision diagrams for the kidney exchange problem. *arXiv preprint arXiv:2009.13715*. 118, 121, 122, 134

Ribeiro, C.C., Hansen, P., Desaulniers, G., Desrosiers, J. & Solomon, M.M. (2002). *Accelerating strategies in column generation methods for vehicle routing and crew scheduling problems*. Springer. 8

Righini, G. & Salani, M. (2006). Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, **3**, 255–273. iv, 30, 32, 33, 63, 97

Righini, G. & Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, **51**, 155–170. 36

Roberti, R. & Mingozzi, A. (2014). Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, **48**, 413–424. 36

Roodnat, J., Zuidema, W., van de Wetering, J., de Klerk, M., Erdman, R., Massey, E., Hilhorst, M., IJzermans, J. & Weimar, W. (2010). Altruistic donor triggered domino-paired kidney donation for unsuccessful couples from the kidney-exchange program. *American Journal of Transplantation*, **10**, 821–827. 118

Røpke, S. (2012). Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems. *Presentation in Column Generation*, **2012**. 50, 100

Roth, A.E., Sönmez, T. & Ünver, M.U. (2004). Kidney exchange. *The Quarterly journal of economics*, **119**, 457–488. 117

Roth, A.E., Sönmez, T. & Ünver, M.U. (2005). Pairwise kidney exchange. *Journal of Economic theory*, **125**, 151–188. 119

Roth, A.E., Sönmez, T. & Ünver, M.U. (2007). Efficient kidney exchange: Coincidence of wants in markets with compatibility-based preferences. *American Economic Review*, **97**, 828–851. 119, 120

Ryan, D. & Foster, E. (1981). Rn integer programming approach to scheduling. 45, 100

Sadykov, R., Vanderbeck, F., Pessoa, A., Tahiri, I. & Uchoa, E. (2019). Primal heuristics for branch and price: The assets of diving methods. *INFORMS Journal on Computing*, **31**, 251–267. 48, 57

Sadykov, R., Uchoa, E. & Pessoa, A. (2021). A bucket graph–based labeling algorithm with application to vehicle routing. *Transportation Science*, **55**, 4–28. 32

Saidman, S.L., Roth, A.E., Sönmez, T., Ünver, M.U. & Delmonico, F.L. (2006). Increasing the opportunity of live kidney donation by matching for two-and three-way exchanges. *Transplantation*, **81**, 773–782. 136

Salani, M. & Vacca, I. (2011). Branch and price for the vehicle routing problem with discrete split deliveries and time windows. *European Journal of Operational Research*, **213**, 470–477. 56

Scandiatransplant (2023). Scandiatransplant. http://www.scandiatransplant.org/, accessed: 2023-07-06. 3, 153, 205

Shapley, L.S. *et al.* (1953). A value for n-person games. 158

Silva, M.M., Subramanian, A. & Ochi, L.S. (2015). An iterated local search heuristic for the split delivery vehicle routing problem. *Computers & Operations Research*, **53**, 234–249. 55

Sluijk, N., Florio, A., Kinable, J., Dellaert, N. & Van Woensel, T. (2021). A chance-constrained two-echelon vehicle routing problem with stochastic demands. *Optimization Online*. 85

Sluijk, N., Florio, A.M., Kinable, J., Dellaert, N. & Woensel, T.V. (2023). Two-echelon vehicle routing problems: A literature review. *European Journal of Operational Research*, **304**, 865–886. 85

Smeulders, B., Blom, D.A. & Spieksma, F.C. (2022). Identifying optimal strategies in kidney exchange games is σ 2 p-complete. *Mathematical Programming*, 1–22. 155

Soleilhac, G. (2022). *Optimisation de la distribution de marchandises avec sous-traitance du transport: une problématique chargeur*. Ph.D. thesis, Ecole nationale supérieure Mines-Télécom Atlantique. iv, ix, 56, 58, 70, 71, 73, 75, 77, 78, 79, 81

Solomon, M.M. (1987). Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, **35**, 254–265. 71, 103

Taillard, E.D. (1999). A heuristic column generation method for the heterogeneous fleet VRP. *RAIRO - Operations Research*, **33**, 1–14. 57

Tilk, C., Rothenbächer, A.K., Gschwind, T. & Irnich, S. (2017). Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, **261**, 530–539. 32

Toth, P. & Vigo, D. (2003). The granular tabu search and its application to the vehicle-routing problem. *INFORMS Journal on Computing*, **15**, 333–346. 49, 66

Tukey, J.W. *et al.* (1977). *Exploratory data analysis*, vol. 2. Reading, MA. 174

Valentín, M.O., Garcia, M., Costa, A.N., Bolotinha, C., Guirado, L., Vistoli, F., Breda, A., Fiaschetti, P. & Dominguez-Gil, B. (2019). International cooperation for kidney exchange success. *Transplantation*, **103**, e180–e181. 3, 153, 205

Viklicky, O., Krivanec, S., Vavrinova, H., Berlakovich, G., Marada, T., Slatinska, J., Neradova, T., Zamecnikova, R., Salat, A., Hofmann, M. *et al.* (2020). Crossing borders to facilitate live donor kidney transplantation: the czech-austrian kidney paired donation program–a retrospective study. *Transplant International*, **33**, 1199–1210. 118

Wang, Z., Li, Y. & Hu, X. (2015). A heuristic approach and a tabu search for the heterogeneous multi-type fleet vehicle routing problem with time windows and an incompatible loading constraint. *Computers & Industrial Engineering*, **89**, 162–176. 56

Wentges, P. (1997). Weighted dantzig-wolfe decomposition for linear mixed-integer programming. *International Transactions in Operational Research*, **4**, 151–162. 47

Yoo, K.D., Kim, C.T., Kim, M.H., Noh, J., Kim, G., Kim, H., An, J.N., Park, J.Y., Cho, H., Kim, K.H. *et al.* (2016). Superior outcomes of kidney transplantation compared with dialysis: an optimal matched analysis of a national population-based cohort study between 2005 and 2008 in korea. *Medicine*, **95**. 117, 152

Zbib, H. & Laporte, G. (2020). The commodity-split multi-compartment capacitated arc routing problem. *Computers & Operations Research*, **122**, 104994. 56, 57

Zhou, L., Baldacci, R., Vigo, D. & Wang, X. (2018). A multi-depot two-echelon vehicle routing problem with delivery options arising in the last mile distribution. *European Journal of Operational Research*, **265**, 765 – 778. 85

Résumé étendu en français

L'optimisation mathématique fournit des outils pour aider les décideurs à résoudre les problèmes qui se posent dans différents domaines d'application. Les progrès récents en matière d'algorithmes exacts permettent de résoudre à l'optimum de grandes instances de problèmes bien connus, même si ces derniers sont théoriquement NP-difficile. De tels problèmes sont par exemple le *problème du voyageur de commerce* ou le *problème de tournées de véhicules avec capacité*. Aujourd'hui, les problèmes les plus difficiles sont ceux qui intègrent deux problèmes d'optimisation ou ceux dont la résolution implique de résoudre d'autres problèmes d'optimisation, qui sont généralement complexes en eux-mêmes. Dans le premier cas, le problème intègre des problèmes à deux niveaux de décision différents, par exemple tactique et opérationnel, ou deux sous-problèmes au même niveau de décision. Dans le second cas, la résolution du problème d'optimisation principal nécessite le calcul de certaines valeurs obtenues par la résolution d'autres problèmes d'optimisation.

De tels problèmes peuvent se poser dans différents domaines d'application. Ici, nous nous concentrons sur les domaines du transport et de la santé. Dans le domaine du transport, nous considérons un problème intégré qui intègre deux problèmes au même niveau opérationnel. Plus précisément, nous considérons un problème qui se pose dans les circuits courts et locaux pour des denrées alimentaires (Berti & Mulligan, 2016) : des produits agricoles de haute qualité doivent être livrés à partir d'exploitations agricoles de taille moyenne à des clients dont les principales préoccupations sont la qualité et la traçabilité des produits (King *et al.*, 2014). Ces circuits d'approvisionnement peuvent impliquer plusieurs fournisseurs (exploitations agricoles) et clients, mais leurs emplacements doivent être situés à une distance maximale de moins de 100km (Blanquart *et al.*, 2010). Dans ce contexte, les fournisseurs produisent plusieurs produits alimentaires, tels que des fruits, des légumes et/ou de la viande. En outre, ces types de produits ne peuvent être fournis que par certains fournisseurs, mais ils peuvent différer de l'un à

l'autre. De même, les clients ont une demande pour différents types de produits. Par conséquent, ces circuits d'approvisionnement impliquent généralement des installations intermédiaires, connues sous le nom de plateformes de distribution, dont le rôle est de consolider les produits avant les livraisons et, par conséquent, de réduire les coûts de transport (Berti & Mulligan, 2016). En effet, dans cette application, les fournisseurs considérés sont des exploitations agricoles de petite taille qui n'ont pas les ressources nécessaires pour gérer elles-mêmes l'ensemble du transport et de la logistique. Par conséquent, la collecte et la livraison des produits sont généralement coordonnées par un décideur central incarné, par exemple, par une association d'agriculteurs ou une autorité politique. Le décideur central a pour objectif de minimiser le coût total de transport du système, qui résulte des opérations suivantes. Les fournisseurs possèdent leurs propres véhicules de grande capacité pour acheminer les produits agricoles vers les plateformes de distribution par des trajets directs. Une fois que la demande a été consolidée dans chaque plateforme de distribution, elle est livrée aux clients par une flotte de véhicules de plus petite capacité gérée par le centre de distribution lui-même. Les véhicules suivent des itinéraires qui commencent et se terminent dans la plateforme de distribution à laquelle ils appartiennent.

Le problème d'optimisation qui se pose dans cette application spécifique est le *Multi-Commodity two-echelon Distribution Problem* (MC2DP), un problème de routage défini sur deux échelons où les différents types de produits doivent être pris en compte de manière explicite. En effet, seuls certains fournisseurs proposent les mêmes produits et les demandes des clients peuvent être satisfaites par différents véhicules.

Dans les applications de santé, nous considérons un problème intégré dans lequel le problème principal est enrichi de conditions supplémentaires qui impliquent la résolution d'autres problèmes d'optimisation. Plus précisément, nous examinons un problème qui se pose dans le contexte des greffes de rein impliquant des donneurs vivants. Cette pratique permet de proposer aux patients atteints d'une maladie rénale grave une option de transplantation supplémentaire lorsqu'un rein provenant d'un donneur décédé n'est pas rapidement disponible (Lentine *et al.*, 2023; Nemati *et al.*, 2014). De nos jours, il est courant que chaque pays gère son propre *programme d'échange de reins* pour coordonner les transplantations rénales entre les patients et les donneurs vivants affiliés à tous les hôpitaux de ce pays (voir, par exemple, Biró *et al.*, 2019a). Ces programmes sont souvent organisés périodiquement, généralement tous les 3 ou 4 mois. L'une des principales préoccupations en matière de transplantation rénale est le risque

de rejet. Pour réduire ce risque, les greffes ne sont réalisées qu'entre des patients et des donneurs compatibles l'un avec l'autre selon plusieurs critères médicaux (Kälble *et al.*, 2005). Il est souvent difficile de trouver des patients et des donneurs qui répondent à ces exigences, même lorsque les transplantations sont gérées au niveau national. C'est pourquoi, ces dernières années, certains pays ont commencé à collaborer en fusionnant leurs pools de patients et de donneurs afin d'augmenter les possibilités de réaliser davantage de greffes et/ou des greffes de meilleure qualité (voir, par exemple, Böhmig *et al.*, 2017; Scandiatransplant, 2023; Valentín *et al.*, 2019). Comme pour les pays considérés individuellement, des programmes impliquant plusieurs pays sont menés périodiquement. Dans ce cadre de collaboration entre pays, outre la détermination des greffes à effectuer, l'objectif est d'assurer : (i) la stabilité du système à chaque exécution : aucun pays ne doit être incité à quitter le système ; (ii) l'équité du système : les disparités dans le nombre ou la qualité des greffes attribués aux pays sont lissées au cours du programme. Nous appelons le problème d'optimisation sous-jacent *Iterative International Kidney Exchange Problem* (IIKEP).

Le MC2DP et l'IIKEP appartiennent tous deux à la catégorie des problèmes d'optimisation intégrés. Le MC2DP intègre les activités des échelons de collecte et de livraison au niveau opérationnel. En outre, l'échelon de livraison est composé de plusieurs sous-problèmes de livraison : un par centre de distribution, appelé *Commodity constrained Split Delivery Vehicle Routing Problem* (C-SDVRP). Le C-SDVRP est une extension du CVRP, qui traite explicitement plusieurs types de produits. Chaque client a besoin de plusieurs types produits qui sont compatibles et peuvent être mélangées dans les véhicules. Pour réduire les coûts, la demande d'un client peut être acheminée par plusieurs véhicules, mais pour des raisons pratiques, la livraison d'un seul type de produit ne peut pas être fractionnée. L'IIKEP intègre le problème d'échange de reins (KEP) avec des conditions de stabilité et d'équité. Le KEP vise à déterminer un ensemble de greffes de rein dans un pool de patients et de donneurs de telle sorte que le nombre ou la qualité des greffes soit maximal.

Les problèmes d'optimisation intégrée peuvent être résolus avec différentes méthodes : métaheuristiques, heuristiques dédiées avec ou sans garantie de performance, ou algorithmes exacts. L'objectif de cette thèse est de concevoir des algorithmes exacts ou des méthodes capables de fournir de bonnes bornes sur les valeurs optimales des instances du problème, c'est-à-dire des algorithmes heuristiques avec une garantie de performance. Les techniques bien connues pour obtenir de tels résultats sont basées

sur la génération de colonnes et le paradigme de *Branch-Price-and-Cut* (BPC) (Barnhart *et al.*, 1998). Le principal défi dans la conception d'algorithmes efficaces pour les problèmes intégrés réside dans la capacité à résoudre les problèmes centraux, dans notre cas le C-SDVRP et le KEP, de manière très efficace.

Dans ce qui suit, nous résumons les principales contributions que nous avons apportées dans cette thèse.

1. Pour résoudre le C-SDVRP très efficacement, nous concevons un algorithme heuristique avec une garantie de performance basée sur la génération de colonnes. L'algorithme intègre une nouvelle heuristique de *pricing* dédiée à l'aspect multi-produits du problème. Des expérimentations computationnelles approfondies sur des instances de référence tirées de la littérature montrent que notre algorithme fournit plusieurs nouvelles solutions (les meilleures connues) et améliore considérablement le temps de calcul par rapport à une heuristique de pointe pour le C-SDVRP. La nouvelle heuristique de *pricing* permet de réduire le temps de calcul.

2. Pour le MC2DP, nous proposons une formulation étendue basée sur celle du C-SDVRP, et nous développons un algorithme de BPC exact pour résoudre le problème. L'algorithme comprend deux familles d'inégalités valides robustes qui exploitent l'aspect multi-produits du problème. Nous testons l'algorithme de BPC sur des instances de référence précédemment introduites dans la littérature. Près de 60% des instances sont résolues à l'optimum, tandis que les autres sont laissées avec un écart moyen égal à 2.1%.

3. Afin de résoudre le KEP de manière très efficace, nous proposons un algorithme de BPC qui inclut deux familles d'inégalités valides non robustes. Nous évaluons les performances du BPC proposé sur trois ensembles d'instances de référence tirées de la littérature. Notre BPC est le premier à être testé sur des instances du KEP ayant des caractéristiques différentes. Pour chaque ensemble, nous testons le BPC par rapport à un algorithme de référence. Le BPC produit des résultats comparables à ceux de la littérature sur l'ensemble d'instances le plus simple et surpasse les résultats sur les deux autres ensembles.

4. Pour l'IIKEP, nous proposons une procédure itérative, où le problème qui se pose à chaque itération est un KEP avec des conditions de stabilité et d'équité. Ce

problème est résolu en adaptant l'algorithme de BPC que nous avons conçu pour le KEP. Nous fournissons une analyse managériale de l'impact des conditions de stabilité et d'équité sur les solutions. Dans l'analyse expérimentale, nous montrons que l'instabilité du système est fortement réduite et que les écarts par rapport à un scénario équitable idéal sont faibles. Il est à noter que dans ce cadre de collaboration entre pays, l'amélioration en termes de bénéfices médicaux est importante. Par rapport à cette amélioration, le prix à payer pour les conditions de stabilité et d'équité est faible. Par conséquent, dans ces conditions, les pays sont incités à participer à un programme international d'échange de reins.

Branch-price-and-cut algorithms for integrated optimisation problems in transportation and healthcare

Abstract: In this thesis, we study optimisation problems in transportation and healthcare. For transportation applications, first, we address the *Commodity constrained Split Delivery Vehicle Routing Problem* (C-SDVRP) a routing problem where customer demands are composed of multiple commodities. Several vehicles can deliver a customer, however, the demand for a single commodity must be delivered by one vehicle only. Then, we consider an extension of the C-SDVRP defined on two echelons, the *Multi-Commodity two-echelon Distribution Problem*. Commodities are collected in the first echelon by direct round trips between distribution centres and suppliers, and, in the second echelon, they are delivered to the customers from each distribution centre as in the C-SDVRP. For healthcare applications, first, we study the *Kidney Exchange Problem*, whose aim is to find the best set of transplants, in terms of medical benefit, in a pool of donors and patients. Then, we considered collaboration and fairness in the *International Kidney Exchange Problem* where different countries join a common kidney exchange program. All the problems we consider share a common structure: they reduce in finding the best set of paths in a graph. We modelled them by means of extended formulations where the variables correspond to the paths, and we solve them by heuristic or exact methods based on the branch-and-price paradigm.

Keywords: column generation, integrated operational problems, vehicle routing problems, kidney exchange problems.

Algorithmes de branch-price-and-cut pour des problèmes d'optimisation intégrés en transport et santé

Résumé : Dans cette thèse, nous étudions des problèmes d'optimisation en transport et santé. Pour les applications en transport, nous traitons le problème nommé *Commodity constrained Split Delivery Vehicle Routing Problem* (C-SDVRP) où les demandes des clients sont composées de plusieurs produits. Plusieurs véhicules peuvent livrer un client, cependant la demande d'un produit doit être livrée par un seul véhicule. Ensuite, nous considérons une extension du C-SDVRP définie sur deux échelons, le *Multi-Commodity two-echelon Distribution Problem*. Les produits sont collectés au premier échelon par des trajets directs entre les centres de distribution et les fournisseurs, puis au second échelon ils sont livrés aux clients à partir de chaque centre de distribution comme dans le C-SDVRP. Pour les applications en santé, nous étudions le *Kidney Exchange Problem*, dont l'objectif est de trouver le meilleur ensemble de greffes, en termes de bénéfice médical, dans un pool de donneurs et de patients. Ensuite, nous avons examiné la collaboration et l'équité dans le *International Kidney Exchange Problem* où différents pays adhèrent à un programme commun d'échange de reins. Tous les problèmes que nous considérons ont une structure commune : ils se réduisent à la recherche du meilleur ensemble de chemins dans un graphe. Nous les avons modélisés au moyen de formulations étendues où les variables correspondent aux chemins ; et nous les avons résolus par des méthodes heuristiques ou exactes basées sur un paradigme de type *branch-and-price*.

Mots-clés : génération de colonnes, problèmes opérationnels intégrés, problèmes de tournées de véhicules, problèmes d'échanges de reins.
