The virtual network functions placement and routing problem

Ahlam Mouaci

PSL UNIVERSITÉ PARIS

orange™

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Université Paris Dauphine

The Virtual Network Functions Placement and Routing Problem

Soutenue par
Ahlam MOUACI
Le 30 November 2020

École doctorale n°543
École Doctorale SDOSE

Spécialité
Informatique

Composition du jury :

Ivana LJUBIĆ Professor, LAMSADE	*Directeur de thèse*
Nancy PERROT Ingénieur de recherche, Orange Labs	*Examinateur*
Éric GOURDIN Ingénieur de recherche, Orange Labs	*Examinateur*
Ali Ridha MAHJOUB Professeur des université, Université Paris Dauphine	*Président*
Brigitte JAUMARD Professor, Concordia University	*Rapporteur*
Laurent ALFANDARI Professor, ESSEC Business School	*Rapporteur*
Fabio FURINI Ingénieur de recherche, Consiglio Nazionale delle Ricerche	*Examinateur*
Markus LEITNER Associate Professor, Vrije Universiteit Amsterdam	*Examinateur*

Dauphine | PSL UNIVERSITÉ PARIS

To my beloved family, dear husband, darling daughter,
friends, teachers and all the people I love.

Acknowledgements

I would like to convey my sincere gratitude to my supervisors Ivana Ljubić, Nancy Perrot, and Éric Gourdin, for their advice, availability, and for providing me scientific and moral support throughout this work. They were able to transmit to me the knowledge, the rigor, the motivation, and the passion for research in general and combinatorial optimization in particular. I am grateful for all the efforts that have been deployed in order to make this project successful and for all the constructive remarks and criticism that helped me improving my work and acquiring a lot of knowledge.

I was very honored that Mrs. Brigitte Jaumard, Professor at the Concordia University, Canada, and Mr. Laurent Alfandari, Professor at the ESSEC Business School of Paris, accepted to review my thesis.

My deep gratitude goes to Mr. Ali Ridha Mahjoub, Professor at université Paris-Dauphine, to Mr. Fabio Furini, Researcher at the Italian National Research Council (CNR), and to Mr. Markus Leitner, Associate Professor at Vrije Universiteit Amsterdam, for having accepted to examine my thesis.

I address, likewise, my warmly thanks to all my colleagues in Orange Labs and LAMSADE for the welcoming and friendly environment, for the lively discussions, for their encouragement, their friendship, and the scientific exchanges that greatly contributed to my scientific and personal development during these years of my thesis.

I would like to especially thank Abdelatif, Quentin, Adrien, Wesley, Fatma, Raja, Ons, Roufia, Amel, George, Aymen, Jalal and Isaias.

I would like to thank my husband "Youcef" for being a source of love, strength, inspiration and permanent support during the last three years, also all the members of my family and my husband's family, especially my parents for their help, encouragements and lovely company.

Abstract

Network Functions Virtualization (NFV) and *Software Defined Networking (SDN)* are two promising techniques for the next generation of telecommunication networks. Their introduction allows time, energy, and cost minimization. In this dissertation, we study a problem faced by the network service providers, named, the *Virtual Network Functions Placement and Routing Problem (VNFPRP)* in Software Defined Networks.

In this problem, a set of *Virtual Network Functions (VNFs)* has to be installed in a telecommunication network at minimum cost. For each given origin-destination pair of nodes (commodities), a latency-constrained routing path has to be found visiting the required VNFs. Placing VNFs on network nodes and routing data through these nodes is a very challenging combinatorial optimization problem. Obviously, the problem becomes even more difficult if, in addition, data flows have to be routed using the concept of *Service Functions Chaining (SFC)* for which VNFs have to be encountered in a pre-defined order.

In this thesis, we prove that the VNFPRP is NP-hard in a strong sense, even for a single commodity and without node-capacity, VNF-capacity, latency, and precedence constraints. We provide a compact Mixed Integer Linear Programming (MILP) formulation to model it. This formulation does not seem to be strong enough for finding a solution using an off-the-shelf solver. In order to tackle the problem from a computational perspective, we provide MILP-based heuristic. The obtained results indicate that our MILP-heuristic provides high-quality solutions.

Moreover, we propose two extended formulations for the problem and derive two Branch-and-Price algorithms. For each formulation, we present a column generation procedure to solve the linear relaxation, the way to compute the dual bound, and the branching scheme. We also provide several families of valid inequalities to strengthen the LP-bounds. We present computational results and discuss the performance of each algorithm.

Furthermore, we discuss a variant of the problem, and we provide theoretical results that allow us to derive Benders reformulation of the problem, along with several families of valid inequalities. These ingredients are combined in a Branch-and-Benders-Cut framework and computationally tested on a wide range of realistic instances and

compared with the Automatic Benders decomposition provided by Cplex.

Computational results indicate that our decomposition and reformulation approaches are more efficient compared to the two methods (the MILP compact formulation and the automatic Benders) provided by the off-the-shelf solver on a set of realistic instances, both in terms of CPU time and overall solution quality.

Key words : Combinatorial optimization, Column generation, Branch-and-Price, Valid inequalities, Benders decomposition, Branch-and-Benders-cut, Heuristic, Virtual Network Functions, Software-Defined Networking, Service Functions Chaining.

Résumé

La *Virtualisation des fonctions réseau* et les *réseaux définis par logiciel* sont deux nouvelles technologies prometteuses qui émergent dans la nouvelle génération des réseaux de télécommunication. Leur utilisation permet la minimisation du temps de traitement des services et un gain en énergie et en coûts. Dans cette thèse, nous étudions un problème auquel sont confrontés les fournisseurs des services réseau, intitulé le *problème de placement des fonctions virtuelles et de routage des services* dans les réseaux définis par logiciel.

Étant donnés un ensemble de fonctions de réseau virtuelles (VNF) et un ensemble de paires de nœuds origine-destination (commodités), le problème étudié dans cette thèse consiste à trouver les installations optimales des fonctions virtuelles sur les noeuds du réseau de télécommunication, et le chemin associé satisfaisant les contraintes de latence et passant par les fonctions virtuelles installées. Ce problème est un problème d'optimisation combinatoire difficile à résoudre. Évidemment, le problème devient encore plus difficile, si en plus, les flux de données doivent être acheminés en utilisant le concept de *Chaînage de fonctions de service (SFC)* pour lequel les VNFs doivent être traitées le long du chemin de routage dans un ordre prédéfini. D'autres contraintes techniques peuvent être considérées dans ce problème. L'objectif est de minimiser les coûts d'installation des fonctions virtuelles sur les noeuds du réseau et les coûts d'activation des noeuds.

Dans ce manuscrit, nous commençons par prouver que le problème est NP-Difficile au sens fort, même en considérant une seule commodité, et en relâchant les contraintes de capacité sur les nœuds et sur les fonctions virtuelles, et aussi les contraintes de latence et de précédence. Nous proposons ensuite une formulation PLNE (Programmation linéaire en nombres entiers) compacte pour modéliser le problème. Cette formulation ne semble pas être assez forte pour trouver des solutions réalisables en un temps raisonnable à l'aide d'un solveur standard. Afin de remédier à cela, nous fournissons, une heuristique basée sur une formulation PLNE.

Nous proposons également, deux formulations PLNE étendues pour modéliser le problème et nous définissons l'algorithme de Branch-and-Price associé. Pour chacune de ces deux formulations, nous présentons, la procédure de génération de colonnes afin

de résoudre la relaxation linéaire, le calcul de la borne duale, et aussi le schéma de branchement. De plus, nous définissons plusieurs familles d'inégalités valides afin de renforcer la relaxation linéaire. Nous comparons les deux algorithmes proposés et nous discutons leur performance.

Ensuite, nous étudions une variante du problème et nous présentons des résultats théoriques qui nous permettent de le reformuler en appliquant la décomposition de Benders. Nous renforçons cette reformulation par un ensemble d'inégalités valides. Tout cela est combiné dans un algorithme de Branch-and-Benders-cut que nous avons testé et comparé avec l'algorithme de Benders automatique inclus dans le solveur commercial Cplex.

Les résultats numériques montrent que nos approches de décomposition et de reformulation sont plus efficaces que les deux méthodes (la formulation compacte et l'algorithme de Benders automatique) fournies par un solveur standard sur un ensemble d'instances réalistes, à la fois en terme de temps CPU et en terme de qualité de la solution. Ces résultats montrent également que notre heuristique fournit des solutions de grande qualité.

Mots clef : Optimisation combinatoire, Génération de colonnes, Branch-and-Price, inégalités valides, décomposition de Benders, Branch-and-Benders-Cut, Heuristique, Fonctions virtualisées de réseau, Réseaux définis par logiciel, Chaînage de fonctions de service.

Contents

List of Figures

List of Tables

List of Abbreviations

API	**A**pplication **P**rogramming **I**nterface
B&B	**B**ranch-and-**B**ound
B&P	**B**ranch-and-**P**rice
CAPEX	**Cap**ital **Ex**penditure
CG	**C**olumn **G**eneration
CLI	**C**ommand **L**ine **I**nterface
COTS	**C**ommercial **o**ff-**t**he-**s**helf
CPU	**C**entral **P**rocessing **U**nit
DPI	**D**eep **P**acket **I**nspection
ETSI	**E**uropean **T**elecommunications **S**tandards **I**nstitute (ETSI)
FW	**F**ire**w**all
GB	**G**iga**b**yte
ILP	**I**nteger **L**inear **P**rogram
IP	**I**nternet **P**rotocol
LAN	**L**ocal **A**rea **N**etwork
LB	**L**ower **B**ound
LP	**L**inear **P**rogram
MILP	**M**ixed **I**nteger **L**inear **P**rogram
MIP	**M**ixed **I**nteger **P**rogram
NAT	**N**etwork **A**ddress **T**ranslator
NF	**N**etwork **F**unction
NFV	**N**etwork **F**unctions **V**irtualisation
NFVI	**N**etwork **F**unctions **V**irtualisation **I**nfrastructure
NFVI-PoP	**NFVI P**oint **o**f **P**resence
NFV-MANO	**NFV Man**agement and **O**rchestration
NFVO	**N**etwork **F**unctions **V**irtualisation **O**rchestrator
NS	**N**etwork **S**ervice
OPEX	**Op**erational **Ex**penditure
QoE	**Q**uality **o**f **E**xperience
QoS	**Q**uality **o**f **S**ervice

SDN	**S**oftware-**D**efined **N**etworking
SFC	**S**ervice **F**unction **C**hain
SNDlib	**S**urvivable **N**etwork **D**esign **Lib**rary
UB	**U**pper **B**ound
VNF	**V**irtualised **N**etwork **F**unction
VNFPRP	**V**irtual **N**etwork **F**unctions **P**lacement and **R**outing **P**roblem
VPN	**V**irtual **P**rivate **N**etwork
WAN	**W**ide **A**rea **N**etwork

Introduction

Typically, telecommunication networks are composed of different devices such as switches, routers, and middleboxes, which are used in order to provide network functions such as Proxies, Firewalls (FW), Intrusion Detection Systems (IDS), Load Balancers (LB), etc. The hardware middleboxes are costly, energy-intensive and have short lifecycles [71]. Constantly, network service providers have to deal with a large number of traffic requests in order to satisfy customer demands. These traffic requests represent services such as video streaming, virtual private network (VPN), online gaming, etc. Moreover, each service requires a specific Service Function Chain (SFC), representing a sequence of hardware devices that must be traversed by the data flows in a pre-defined order. Routing packets through these fixed devices is very time consuming and reduces the QoS (e.g., end-to-end latency). Also, when a new service is required, new hardware devices should be used, placed, and connected to the network elements [89]. Furthermore, in traditional telecommunication networks, no device has visibility of the whole network. Each switch has its own data plane, to forward the traffic, and its own control plane, to decide where to send the data. Having a considerable number of devices implies managing a considerable number of control and data planes. This requires a manual configuration, and makes packet forwarding very complicated within a network [48, 111]. Therefore, network management becomes very challenging [110], which is very expensive in terms of CAPEX and OPEX.

During the last decade, the legacy telecommunication networks have started their transformation by introducing two promising technologies, namely, the Network Function Virtualization and Software-Defined Networking. The Network Function Virtualization allows the network functions to be executed as software on commercial off-the-shelf equipment, thus being instantiated on-demand without installing new equipment. Software-defined networking is a new paradigm that simplifies network management and makes the deployment of new services easier by separating the data plane from the control plane. This is done by creating one or multiple central controllers having a global view of the whole network state.

NFV and SDN provide various benefits: specifically, the VNFs could be installed on-demand on network nodes, and the routing paths associated with the Service Function Chains could be computed dynamically to encounter the installed VNFs [11]. These two technologies bring new flexibility in network management. Nevertheless, their introduction generates hard decision problems combining the choice of locations to deploy the VNFs and the computation of efficient routing paths meeting the installed VNFs in the right order. Both problems are widely studied separately in the literature, but their combination, specified by the VNF chaining problem, introduces a new challenge [6].

In this thesis, we study the Virtual Network Functions Placement and Routing Problem faced by Network Service Providers (NSPs). Given a telecommunication network, a set of VNFs and a set of traffic requests (or commodities), the VNFPR problem consists in determining the optimal VNFs placement at network nodes, and the associated latency-constrained routing paths for each commodity, satisfying the SFC constraints. The VNF-installation and the node activation costs are to be minimized. Further technical constraints, allowing a better Quality of the Service (QoS) and Quality of Experience (QoE), can be considered along with nodes capacities, VNFs capacities, or conflict between VNFs of the same service.

In the first part of the thesis, we investigate the problem's basic properties, we show that the problem is strongly NP-hard even for its simplest version, and we propose an MILP compact formulation to model it. This formulation does not seem to be strong enough to find a solution using an off-the-shelf solver in a reasonable time. To tackle the problem from a computational perspective, we propose a path-based heuristic that provides optimal solutions for realistic instances from the literature.

Afterwards, we propose two extended formulations based on Dantzig-Wolfe decomposition to model the problem, namely: path formulation (PF) and Dantzig-Wolfe (DW) formulation. In order to strengthen their LP-bounds, we propose several families of valid inequalities and demonstrate their benefits. We show, theoretically, that the Dantzig-Wolfe formulation is stronger than the path formulation in terms of linear relaxation. We present a branching scheme for each one and develop a Branch-and-Price algorithm. We compare both algorithms with the MILP compact formulation and the automatic Benders of Cplex.

In the last part of the dissertation, we study a variant of the problem in which we relax the VNF-capacity and conflict constraints. We provide theoretical results that allow us to reformulate the problem using Benders decomposition and three families of valid inequalities to strengthen the LP-bounds. We combine all these ingredients in a Branch-and-Benders-Cut framework, and we test it on a set of realistic benchmark

instances.

This manuscript is organized as follows. In Chapter 1, we briefly present the basic notions of combinatorial optimization, its most important approaches, and define some notions of the telecommunication network. This chapter also includes a review of the state-of-the-art methods for the VNFPRP. In Chapter 2, we present the VNFPRP, its proprieties, and an MILP compact formulation to model it. Chapter 3 is dedicated to the MILP-based heuristic. In Chapter 4, we present two extended formulations and their Branch-and-Price algorithms. In Chapter 5, we discuss a variant of the problem, and we provide theoretical results that allow us to derive Benders reformulation and to develop a Branch-and-Benders-Cut algorithm.

Chapter 1

Preliminaries and State-of-the-Art

This chapter introduces and presents the main notions and definitions necessary to ease the manuscript's understanding. First, we summarize the most important methods in combinatorial optimizaion, such as cutting plane, Branch-and-Cut, Benders reformulation, column generation, and the Branch-and-Price algorithms. Furthermore, the principal graph theory's notions and the key definitions in the telecommunication network are reported. The last section of the chapter is related to the review of the literature on the Virtual Network Functions Placement and Routing problem.

Contents

Outline of the chapter. In Section 1.1, we present the most important reformulation and resolution approaches used in combinatorial optimization. Some important notions of graph theory are provided in Section 1.2. Section 1.3 is dedicated to the telecommunication network notions and definitions. The last section summarizes the literature review related to the VNFPRP.

1.1 Polyhedra and Integer Linear Programming methods

Some definitions in this section have been collected from the book chapter of A.R Mahjoub [95].

1.1.1 Elements of polyhedral theory

Let $x \in \mathbb{R}^n$ be a vector, where n is a positive integer. We say that x is a *linear combination* of x_1, x_2, ..., $x_m \in \mathbb{R}^n$ if there exist m scalars λ_1, λ_2, ..., λ_m such that $x = \sum_{i=1}^{m} \lambda_i x_i$. If $\sum_{i=1}^{m} \lambda_i = 1$, then x is said to be a *affine combination* of x_1, x_2, ..., x_m. Moreover, if $\lambda_i \geq 0$, for all $i \in \{1, \ldots, m\}$, we say that x is a *convex combination* of x_1, x_2, ..., x_m.

Given a set $S = \{x_1, \ldots, x_m\} \in \mathbb{R}^{n \times m}$, the *convex hull* of S is the set of points $x \in \mathbb{R}^n$ which are convex combination of x_1, ..., x_m (see Figure 1.1), that is

$$conv(S) = \{x \in \mathbb{R}^n | x \text{ is a convex combination of } x_1, \ldots, x_m\}.$$

The points x_1, ..., $x_m \in \mathbb{R}^n$ are *linearly independent* if the unique solution of the system $\sum_{i=1}^{m} \lambda_i x_i = 0$, is $\lambda_i = 0$, for all $i \in \{1, \ldots, m\}$. Furthermore, they are *affinely independent* if the unique solution of the system $\sum_{i=1}^{m} \lambda_i x_i = 0, \sum_{i=1}^{m} \lambda_i = 1$, is $\lambda_i = 0$, $i = 1$, ..., m.

A *polyhedron* P is the set of solutions of a linear system $Ax \leq b$, that is $P = \{x \in \mathbb{R}^n | Ax \leq b\}$, where A is a m-row n-columns matrix and b is a vector, $b \in \mathbb{R}^m$. A *polytope* is a bounded polyhedron. A point x of P will be also called a *solution* of P.

An inequality $ax \leq \alpha$ is *valid* for a polyhedron $P \subseteq \mathbb{R}^n$ if for every solution $\overline{x} \in P$, $a\overline{x} \leq \alpha$. This inequality is said to be *tight* for a solution $\overline{x} \in P$, if $a\overline{x} = \alpha$.

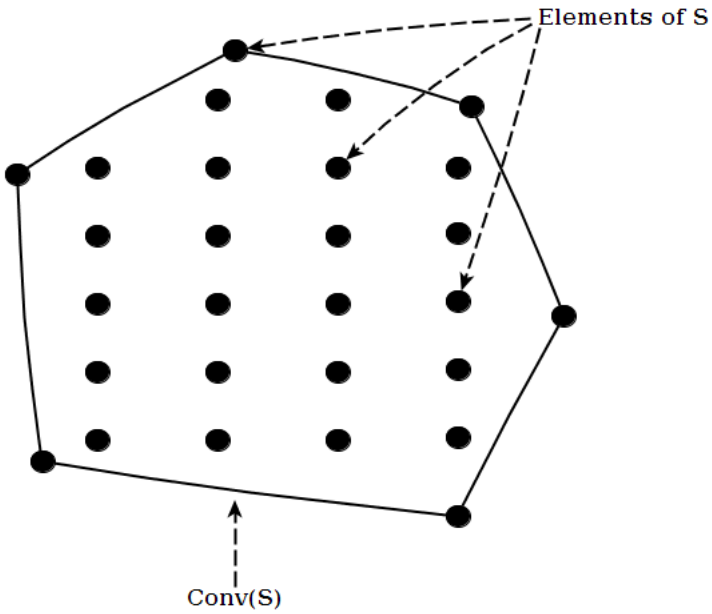


Figure 1.1: Example of a convex hull

The inequality $ax \leq \alpha$ is *violated* by $\overline{x} \in P$, if $a\overline{x} > \alpha$. Let $ax \leq \alpha$ be a valid inequality for the polyhedron P. $F = \{x \in P | ax = \alpha\}$ is called a *face* of P.

An inequality $ax \leq \alpha$ is *redundant*, if the system $\{A'x \leq b'\}$ obtained by removing this inequality from $Ax \leq b$ defines the same polyhedron P. This is the case when $ax \leq \alpha$ can be written as a linear combination of inequalities of the system $A'x \leq b'$.

A solution is an *extreme point* of the polyhedron P, if and only if it cannot be written as the convex combination of two different solutions of P. The polyhedron P can also be described by its extreme points. In fact, every solution of P can be written as a convex combination of some extreme points of P.

Figure 1.2 illustrates the main definitions given is this section.

1.1.2 Cutting plane method

Usually, the characterization of the convex hull of a combinatorial optimization problem is done by defining a large number of inequalities, which is exponential in most of the cases. Define all these linear inequalities is not always obvious. In order to overcome this difficulty, the *cutting plane method* can be used.

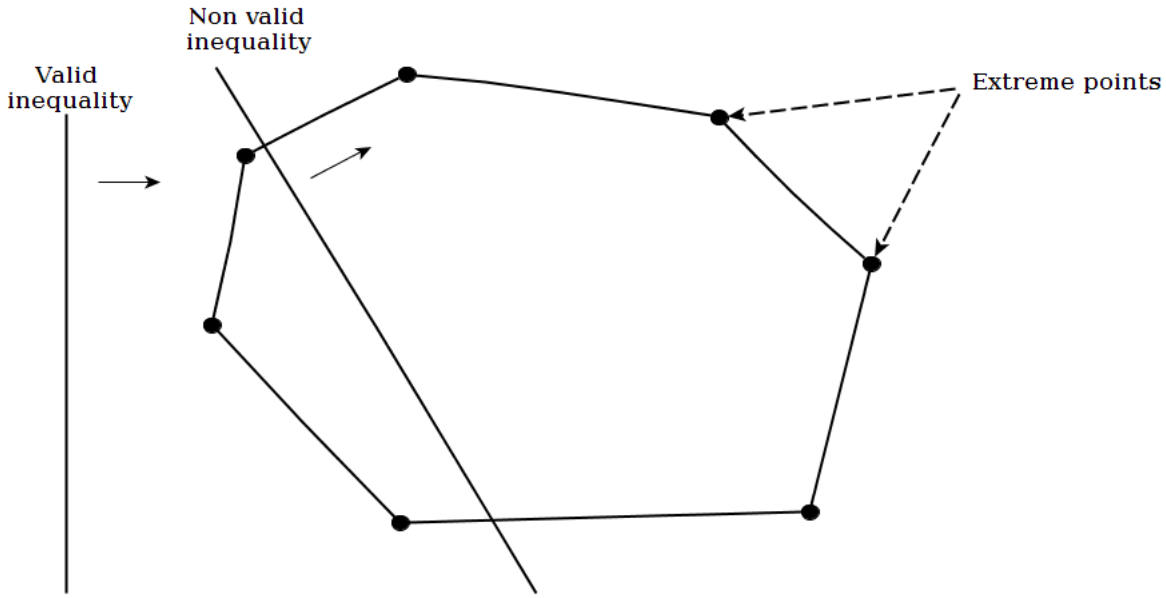


Figure 1.2: Example of valid and non valid inequalities and extreme points

The cutting plane method is based on the so-called *separation problem*. This consists, given a polyhedron P of $\mathbb{R}^n$ and a point $x^* \in \mathbb{R}^n$, in verifying if x^* belongs to P or not, and if not, to identify an inequality $a^T x \geq b$, valid for P and violated by x^*. In the latter case, we say that the hyper-plane $a^T x = b$ separates P and x^* (see Figure 1.3).

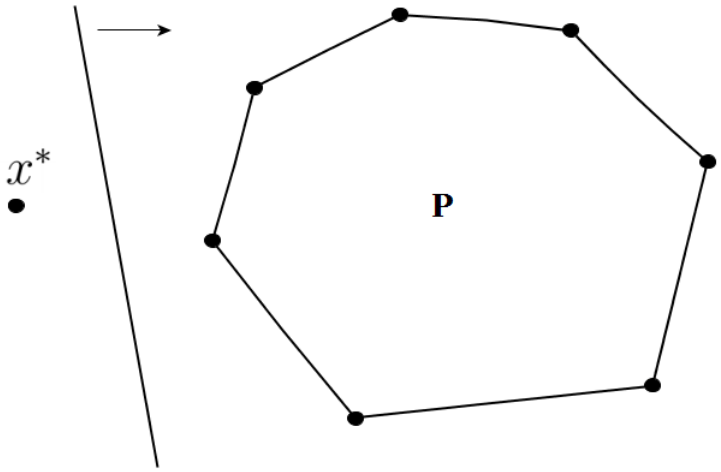


Figure 1.3: A hyper-plane separating x^* and P

The *cutting plane* method consists in solving successive linear programs, with possibly a large number of inequalities, as follows. Let $LP = \min\{cx \mid Ax \geq b\}$ be a

linear program and LP' a linear program obtained by considering a small number of inequalities among $Ax \geq b$. Let x^* be the optimal solution of the latter system. We solve the separation problem associated with $Ax \geq b$ and x^*. This phase is called the *separation phase*. If every inequality of $Ax \geq b$ is satisfied by x^*, then x^* is also optimal for LP. If not, let $ax \geq \alpha$ be an inequality violated by x^*. Then we add $ax \geq \alpha$ to LP' and repeat this process until an optimal solution is found. Algorithm 1 summarizes the different cutting plane steps.

Algorithm 1: A cutting plane algorithm [94]

Data: A linear program LP and its system of inequalities $Ax \geq b$
Result: Optimal solution x^* of LP
1 : Consider a linear program LP' with a small number of inequalities of LP;
2 : Solve LP' and let x^* be an optimal solution;
3 : Solve the separation problem associated with $Ax \geq b$ and x^*;
if *an inequality* $ax \geq \alpha$ *of* LP *is violated by* x^* **then**
    Add $ax \geq \alpha$ to LP';
    Go to step 2 ;
end
else
    x^* is optimal for LP;
    return x^*;
end

The cutting-plane algorithm aims to solve the LP-relaxation of a given problem. The integrality of the provided solution has to be checked, if an integer solution is required and the *Branch-and-Bound algorithm* must be applied if the solution is fractional. Some inequalities have to be added to the problem during the *Branch-and-Bound algorithm* to achieve the optimality. This means that we have to combine both algorithms in order to get an optimal integer solution; the resulting algorithm is called *Branch-and-Cut algorithm*.

1.1.3 Branch-and-Cut algorithm

Consider a combinatorial optimization problem $\mathcal{P}$ which can be stated as: $\min\{cx|Ax \geq b, x \in \{0,1\}^n\}$, where $Ax \geq b$ has a large number of inequalities. A Branch-and-Cut algorithm starts by creating a Branch-and-Bound tree whose root node corresponds to a linear program $LP_0 = \min\{cx|A_0x \geq b_0, 0 \leq x \leq 1\}$, where $A_0x \geq b_0$ is a subsystem of $Ax \geq b$ having a small number of inequalities. Then, we solve the linear relaxation

of the problem that is $LP = \{cx|Ax \geq b, 0 \leq x \leq 1\}$ using a cutting plane algorithm starting from LP_0. Let x_0^* denote its optimal solution and $A_0'x \geq b_0'$ denote the set of inequalities added to LP_0 at the end of the cutting plane phase. If x_0^* is integral, then it is optimal. Otherwise, if x_0^* is fractional, we perform a *branching phase*. This step consists of choosing a variable, say x^1, with a fractional value and adding two nodes P_1 and P_2 in the Branch-and-Cut tree. The node P_1 corresponds to the linear program $LP_1 = \min\{cx|A_0x \geq b_0, A_0'x \geq b_0', x^1 = 0\}$ and $LP_2 = \min\{cx|A_0x \geq b_0, A_0'x \geq b_0', x^1 = 1\}$. We then solve the linear program $\overline{LP}_1 = \min\{cx|Ax \geq b, x^1 = 0\}$ (resp., $\overline{LP}_2 = \min\{cx|Ax \geq b, x^1 = 1\}$) by a cutting plane method, starting from LP_1 (resp. LP_2). If the optimal solution of $\overline{LP}_1$ (resp. $\overline{LP}_2$) is integral then, it is feasible for the problem. Its value is then an upper bound of the optimal solution, and the node P_1 (resp. P_2) becomes a leaf of the Branch-and-Cut tree. If the solution is fractional, then we select a variable with a fractional value and add two children to the node P_1 (resp. P_2), and so on.

Similarly, as in the Branch-and-bound algorithm, infeasible nodes can be generated by adding constraints of type $x^i = 0$ and $x^i = 1$, with x^i fractional variable. Also, this can generate nodes with a worst objective value than the best known lower bound. Both types of nodes are pruned. The Branch-and-cut algorithm terminates when all nodes have been explored.

A good upper bound can be used in the Branch-and-Cut algorithm in order to prune nodes that do not allow its improvement. This allows reducing the size of the Branch-and-Cut tree, and consequently, reduce the time used by the algorithm. Furthermore, this upper bound may be improved by applying a *primal heuristic* which aims to produce a feasible solution from the solution obtained at a given node of the Branch-and-Cut tree when this later solution is fractional (and hence infeasible). When the solution computed is better than the best known upper bound, it may significantly reduce the number of generated nodes, as well as the CPU time. Moreover, this guarantees to have an approximation of the optimal solution before visiting all the nodes of Branch-and-Cut tree, for example, when a CPU time limit has been reached.

1.1.4 Benders decomposition

Benders decomposition is one of the most famous decomposition tools for Mathematical Programming, proposed by JF Benders in [19,20], in order to reformulate and solve specific large-scale optimization problems admitting a set of *complicating variables*. Complicating variables can be defined as those variables that, when temporarily fixed, make the remaining optimization problem considerably more tractable. This means

that fixing their values reduces the given problem to an ordinary linear program [59]. Benders method is based on the cutting-plane method and utilizes the duality theory in order to derive families of cuts. Hence, instead of considering the problem with all its decision variables and constraints of a large-scale problem, Benders decomposition partitions the problem into multiple smaller problems called *Benders subproblems* and one master problem called *Benders master problem*. The computational difficulty of optimization problems increases significantly with a large number of variables and constraints. Solving these smaller problems iteratively can be more efficient than solving a single large problem. The Benders method is generalized for MINLP by Geoffrion in [59].

Let (P) be an MILP admitting two set of variables x and y. (P) is defined as follows:

$$\begin{aligned} (P): \quad & \min cx + dy & & \\ s.t \quad & Ax + By & \geq\ & b \\ & & x \ \in\ & \mathbb{R}^p_+ \\ & & y \ \in\ & Y, \end{aligned}$$

where Y is a "complicated" polyhedron, A and B are matrices, b, c and d are vectors with appropriate dimensions. Once complicated variables y are fixed, the problem (P) becomes significantly easier to solve. The obtained problem represents a linear programming problem in x variables, which (depending on the structure of A) can be decomposed into smaller subproblems for subsets of x. Benders decomposition separates problem (P) into two problems: (i) a master problem that contains the y variables, and (ii) subproblems that contain the x variables. We first note that problem (P) can be written with fixed y variables as follows:

$$(P^{'}) : \min_{\bar{y} \in Y} \{d\bar{y} + \min_{x \geq 0} \{cx : Ax \geq b - B\bar{y}\}\},$$

such that the Benders subproblem is defined by $\phi(\bar{y})$ as follows:

$$(SP) : \phi(\bar{y}) = \min_{x \geq 0} \{cx : Ax \geq b - B\bar{y}\},$$

The Benders subproblem could be feasible, infeasible, or unbounded for a given value of $\bar{y}$. In order to prevent infeasibility, we use the LP-duality theory. Let us denote by (D) the dual formulation associated with (SP) such that:

$$(D) : \phi_D(\bar{y}) = \max_{u \geq 0} \{(b - B\bar{y})^t u : A^t u \leq c\}$$

For the feasible region of (D) we have:

$$F = \{u : u^t A \leq c^t, u \geq 0\}$$

where :

- F represents a polyhedron independent of y.
- F is composed of extreme points $u^p, p = 1, \ldots, P$ and extreme rays $r^q, q = 1, \ldots, Q$

1) In the case where (D) is unbounded this means that the problem (SP) is infeasible. Therefore, there exists an unboundedness direction (extreme ray) r such that:

$$(b - B\bar{y})^t r > 0 \Rightarrow \phi_D(\bar{y}) \rightarrow \infty.$$

In order to eliminate $\bar{y} \in Y$ which causes the Benders subproblem infeasibility, a *Benders feasibility cut* is added to (P):

$$(b - B\bar{y})^t r \leq 0.$$

All infeasible $y \in Y$ can be eliminated by adding constraints

$$(b - By)^t r^q \leq 0, q = 1, \ldots, Q.$$

2) (D) is feasible for $\bar{y} \in Y$, by LP-duality theory we have:

$$\phi(\bar{y}) = \phi_D(\bar{y}) = \max_{u \geq 0}\{(b - B\bar{y})^t u : A^t u \leq c\}.$$

As each problem can be expressed by its extreme points and the optimality is attained at one of them, we have:

$$\phi(\bar{y}) = \phi_D(\bar{y}) = \max_{p \in \{1,\ldots,P\}}\{(b - B\bar{y})^t u^p\}.$$

from which we can derive the *Benders optimality cuts* (see below). Then, (P) can be reformulated by the following program:

$$\begin{aligned} (P): \quad & \min_{y \in Y}\{dy + \max_{p \in \{1,\ldots,P\}} (b - By)^t u^p\} \\ s.t \quad & (b - By)^t r^q \leq 0, q = 1, \ldots, Q \end{aligned}$$

Using an auxiliry variables w to bound the inner maximization problem, the problem (P) is equivalent to the following MIP:

$$\begin{aligned} \min \quad & dy + w \\ s.t. \quad & w \geq (b - By)^t u^p, \quad p = 1, \ldots, P \quad (OptCut) \\ & 0 \geq (b - By)^t u^r, \quad r = 1, \ldots, Q \quad (FeasCut) \\ & y \in Y, w \geq 0 \end{aligned}$$

where *OptCut* represent the Benders optimality cuts and *FeasCut* represent the Benders feasibility cuts.

Since there is, typically, an exponential number of extreme points and extreme rays associated with the dual formulation, the Benders formulation admits an exponential number of constraints. Generating all constraints of type *(OptCut)* and (*FeasCut)* is very time-consuming and implies adding non-efficient constraints to the model. Instead, using the Benders algorithm, Benders cuts are separated at each iteration. We start the algorithm by solving the *relaxed master problem* with a subset of constraints. Let y^* be the optimal solution of this problem. Then we solve the associated dual problem based on y^* in order to generate Benders feasibility and optimality cuts. This procedure is repeated until no further notated Benders cuts are found. This method converges to an optimal solution in a finite number of iterations [59].

There exist two ways to apply the Benders method, old and modern one. In the old version, the relaxed master problem is solved as an MILP after adding cuts. Similarly to the Branch-and-Cut algorithm, the modern version (also called *Branch-and-Benders-cut*) consists in solving, at each node of the branching tree, the relaxed master problem using Benders method.

Some successful applications using Benders method are: large-scale stochastic optimization [132], network design, fixed-charge network design problems [41], linear and quadratic facility location problems [50, 52] and covering location problems [37].

1.1.5 Dantzig-Wolfe decomposition

The Dantzig-Wolfe decomposition definition have been collected from the paper of M.E. Lübbecke and J. Desrosiers [92].

In this subsection, we briefly introduce the Dantzig-Wolfe decomposition developed by George Dantzig and Philip Wolfe in 1960. This decomposition aims to reformulate

an original problem into a master program and one or multiple pricing problems. Let us consider the following linear program representing a compact formulation (original problem):

$$\begin{aligned} (C): \quad \min \quad & c^T x \\ s.t. \quad & Ax \geq b \\ & Dx \geq d \\ & x \geq 0 \end{aligned}$$

Let $P = \{x \in \mathbb{R}^n_+ \mid Dx \geq d\} \neq \emptyset$ be a polyhedron, $\{p_q\}_{q \in Q}$ its extreme points and $\{p_r\}_{r \in R}$ its extreme rays, with Q and R finite. Each $x \in P$ can be written as convex combination of extreme points plus non-negative combination of extreme rays as follows:

$$x = \sum_{q \in Q} p_q \lambda_q + \sum_{r \in R} p_r \lambda_r, \quad \sum_{q \in Q} \lambda_q = 1, \quad \lambda \in \mathbb{R}^{|Q|+|R|}_+$$

By applying the linear transformation $c_j = c^T p_j$ and $a_j = Ap_j, j \in Q \cup R$ we obtain the following formulation:

$$(D-W): \quad \min \quad \sum_{q \in Q} c_q \lambda_q + \sum_{r \in R} c_r \lambda_r \tag{1.1}$$

$$s.t. \quad \sum_{q \in Q} a_q \lambda_q + \sum_{r \in R} a_r \lambda_r \geq b \quad (u) \tag{1.2}$$

$$\sum_{q \in Q} \lambda_q = 1 \quad (v) \tag{1.3}$$

$$\lambda \geq 0 \tag{1.4}$$

The (D-W) model admits a huge number of variables, but it possibly admits less constraints than (C). Constraints (1.3) are called convexity constraints. The compact formulation (C) and the extensive formulation (D-W) are equivalent and they give the same optimal objective function value. Nevertheless, they respective polyhera are not combinatorially equivalent [8,92].

As the (D-W) formulation has a large number of variables, it should be solved using a column generation procedure (see Subsection 1.1.6). First, the model is initialized by a subset of variables, (it is called the *restricted master problem, RMP)*, then the variables

needed to solve its linear relaxation are added at each iteration of the column generation procedure. Given a dual optimal solution $(\bar{u}, \bar{v})$ of the RMP, the subproblem in Dantzig-Wolfe decomposition determines the column j which is optimal for $\min_{j \in Q \cup R} \{c_j - \bar{u}^T a_j - \bar{v}\}$. This objective function representing the minimum reduced cost $\bar{c}^*$, which is, using the linear transformation, equivalent to:

$$\bar{c}^* := \min\{(c^T - \bar{u}^T A)x - \bar{v} | Dx \geq d, x \geq 0\}$$

If $\bar{c}^* \geq 0$, this means that no column with negative reduced cost exists and the column generation procedure terminates. If $\bar{c}^* < 0$, this means that the optimal solution of the pricing problem is either an extreme point or an extreme ray, and the respective column is added to the RMP.

1.1.6 Column generation procedure

Some definitions in this section, Section 1.1.7 and Section 1.2 have been token from the thesis of Y.Magnouche [94].

Generally, compact MILP formulations performance is fairly limited because they often provide a weak linear relaxation. According to Sadykov and Vanderbeck [117], working in an extended variable space allows one to develop tight reformulations for mixed-integer programs. Those reformulations can admit an exponential number of variables which cannot be considered explicitly in the model. In this section, we introduce a method that allows for solving the LP-relaxation of extended formulations. This method is called *Column generation method*.

The column generation method is utilized to solve linear programs admitting an exponential number of variables. Only a small number of variables is considered at the beginning of the procedure. This method was pioneered by Dantzig and Wolfe in 1960 [42] in order to solve problems that could not be managed efficiently (in terms of CPU time and memory consumption) because of their size. Column generation is usually applied either for problems whose structure is suitable for a *Dantzig-Wolfe decomposition* (see Subsection 1.1.5), or for problems with a large number of variables. Gilmore and Gomory [62,63] utilized this method to solve a *cutting stock problem* with a huge number of variables.

The column generation procedure aims to solve a sequence of linear programs with a restricted number of variables (also referred to as columns). The algorithm starts by solving a linear program having a small number of variables, and such that a feasible

solution for the original problem may be identified using this basis. At each iteration of the algorithm, a so-called *pricing problem* is solved with the objective to identify the variables which must enter the current base. A negative reduced cost characterizes these variables. The reduced cost associated with a variable is computed using the dual variables associated with the problem's constraints. Then, the linear program that is obtained by adding the generated variables is solved; this procedure is repeated until no variable with negative reduced cost can be identified by the pricing problem. In this case, the solution obtained from the last restricted program is optimal for the original model. The main step of the column generation procedure is summarized in Algorithm 2.

Algorithm 2: A column generation algorithm [94]

Data : A linear program MP (Master Problem) with a huge number of variables
Output : An optimal solution x^* of MP
1: Consider a linear program RMP (Restricted Master Problem) including only a small subset of variables of the MP;
2: Solve RMP and let x^* be its optimal solution;
3: Solve the pricing problem associated with the dual variables obtained by the resolution of the RMP;
4: **If** there exists a variable x with a negative reduced cost then;
5: add x to RMP.
6: go to 2.
7: **else**
8: x^* is optimal for MP.
9: return x^*.

The column generation procedure can be seen as the dual of the cutting plane method since it adds columns (variables) instead of rows (inequalities) in the linear program. Moreover, the pricing problem may be NP-hard. One can then use heuristic procedures to solve it. For more details on column generation algorithms, the reader is suggested to consult [43, 91, 133].

1.1.7 Branch-and-Price algorithm

The column generation procedure aims to solve the LP-relaxation of an optimization problem, having a huge number of variables. The integrality of the solution provided is not guaranteed. Hence to provide an optimal integer solution for an ILP formulation,

the column generation method has to be merged with a Branch-and-bound algorithm; the resulting algorithm is called *Branch-and-Price algorithm*. This algorithm can be considered as the dual of the Branch-and-Cut algorithm.

The Branch-and-Price algorithm consists of solving each node of the Branch-and-Bound tree using the column generation procedure, all columns with negative reduced cost are added to the model in order to improve its LP-relaxation. When no column with negative reduced cost exists and the current solution is not integer, the algorithm starts the branching phase.

The Branch-and-Price algorithm has been used on various fields to solve large scale integer programming problems, and even real-life problems such as Cutting stock problem [12], Generalized Assignment Problem (GAP) [120], Airline Crew Scheduling [16], Multi-commodity Flow Problems [17], etc.

1.2 Graph theory

The current section is dedicated to introducing basic definitions in graph theory, which will be used throughout the chapters of this dissertation. For more details, we refer the reader to [122].

1.2.1 Undirected graphs

An undirected graph is denoted by $G = (V, E)$ where V is the set of *vertices* or *nodes* and E is the set of *edges*. If e is an edge between two vertices u and v, then u and v are called the *ends* of e, and we write $e = uv$. If u is an end-point of e, then u (resp. e) is said to be *incident* to e (resp. u). Similarly, two vertices u and v forming an edge are said to be adjacent.

Let u and v be two vertices of V. A path p between u and v is an alternating sequence of vertices $(v_0, v_1, v_2, ..., v_{k-1}, v_k)$, where $v_0 = u$, $v_k = v$; $v_{i-1}v_i$ represents an edge connecting v_{i-1} and v_i in the path p for $i = 1, \ldots, k$. p is called *elementary* if it does not visit more than once the same node in G.

1.2.2 Directed graphs

A directed graph is denoted by $D = (N, A)$ where N is the set of nodes and A the set of arcs.

If $a \in A$ is an arc connecting a node u to node v, then u will be called *initial end* and v is called *final end* and we write $a = (u, v)$. We say that a is an *outgoing arc* of node u and an *incoming arc* of node v. The vertices u and v are called ends of a. If v is an end (initial or final) of a, then v (resp. a) is said to be incident to a (resp. v).

1.2.3 Shortest path algorithms

In this subsection, we present the well-known algorithms proposed for finding the shortest path between two nodes in a graph.

1.2.3.1 Dijkstra's algorithm

Given a graph $G = (V, A)$ (resp. $G = (V, E)$), with positive arc (resp. edge) length $l_{uv} \in \mathbb{R}^+$, for each $(u, v) \in A$ (resp. $uv \in E$) the Dijkstra's algorithm aims to find a shortest path to all nodes from a single source in a network [44], [78]. The original algorithm aims to find the shortest paths from a defined source to all nodes in the graph (destinations). This algorithm can be adapted to provide the shortest path between a

source node and one destination node. Dijkstra algorithm is in $O(|E| + |V| \times log|V|)$

Algorithm 3: Dijkstra's algorithm(Graph, source, destination):

```
for each vertex v in the graph G do
    dist[v] := +∞
    previous[v] := ∅
end
dist[source] := 0
Q := the set of all nodes in the graph G
while destination is not reached do
    u := node in Q with smallest distance
    remove u from Q
    for each neighbor v of u do
        dist_uv := dist[u] + luv
        if dist_uv < dist[v] then
            dist[v] := dist_uv
            previous[v] := u
        end
    end
end
return previous
```

1.2.3.2 Moore-Bellman-Ford's algorithm

Given a graph $G = (V, A)$ (resp. $G = (V, E)$), with arc (resp. edge) length $l_{uv} \in \mathbb{R}$, for each $(u, v) \in A$ (resp. $uv \in E$) the Moore-Bellman-Ford's algorithm aims to find a shortest path from a single source to all nodes in a network, without negative length

cycles. The complexity of Moore-Bellman-Ford's algorithm is in $O(|E| \times |V|)$

Algorithm 4: Moore-Bellman-Ford's algorithm(Graph, source):

```
for each vertex v in the graph G \ {s} do
    dist[v] := +∞
    predesessor[v] := ∅
end
dist[source] := 0
for i = 1 to |V| − 1 do
    for each arc (v, w) ∈ A do
        if dist[w] > l_uv + dist[v] then
            dist[w] := l_uv + dist[v]
            predesessor[w] := v
        end
    end
end
return predesessor
```

1.2.3.3 Yen's algorithm

Given a graph $G = (V, A)$, Yen's algorithm aims to find $K-$shortest loopless paths in the graph G between source and destination nodes [138]. Yen's algorithm employs any shortest path algorithm (Dijkstra, for example) in order to find the shortest path, then proceeds to find $K - 1$ deviations of this path, by deleting at each iteration one arc belonging to the shortest path. The complexity of Yen's algorithm based on Djikstra

algorithm is in $O(kn(m + nlogn))$

Algorithm 5: Yen's algorithm(Graph, source, destination, K):

```
SP = Get the shortest path using Dijkstra algorithm(Graph, source,
 destination).
Initialize the set of solutions with the shortest path, Sol = [SP].
Initialize the set of potential paths, PotentialPaths = [ ].
for nbPath = 1 to K do
    for each node i in Sol[nbPath − 1] do
        SpurNode = current node i.
        RootPath = subpath in Sol[nbPath − 1] from the source node to i.
        for each path p in Sol do
            if the same nodes are in path p and the RootPath then
                Remove the edge (i, i + 1) from the graph
            end
        end
        for each node j in the RootPath, with j ≠ SpurNode do
            Remove the node j from the graph
        end
        SpurPath = Dijkstra algorithm(Graph, SpurNode, destination).
        Create the new path TotalPath by concatenating the RootPath and
         the SpurPath: TotalPath = RootPath ∪ SpurPath
        if TotalPath ∉ PotentialPaths then
            Add TotalPath to PotentialPaths
        end
        Restore edges to the graph.
        Restore nodes in RootPath to Graph.
    end
    if PotentialPaths is empty then
        Break
    end
    Sort paths in PotentialPaths from the smaller to the bigger cost.
    Add the lowest path to the set of solutions Sol.
    Delete the path from PotentialPaths.
end
Return the set of solutions Sol.
```

1.3 Telecommunication Networks

Some definitions in this section have been collected from the thesis of A. Tomassilli [129].

This section is dedicated to introducing basic definitions and notations in the telecommunication networks, which will be used throughout the chapters of this dissertation. First, we define the network structure briefly; we present some examples of network functions. Then, we describe the notion of Network Function Virtualization (NFV), we define the Software-Defined Networking (SDN) architecture and explain the Service Functions Chaining (SFC).

1.3.1 Network Structure

The telecommunication network is composed of three parts, as shown in Figure 1.4:

- **Access network** is the part of a telecommunication network that gives the user access to services.
- **Core network** (or backbone) is the part of a network that connects the different parts of the access network. The core network also provides the gateway to other networks.
- **Data center** is a dedicated space within a building, or a group of buildings used to house computer systems and associated components, such as telecommunications and storage systems.

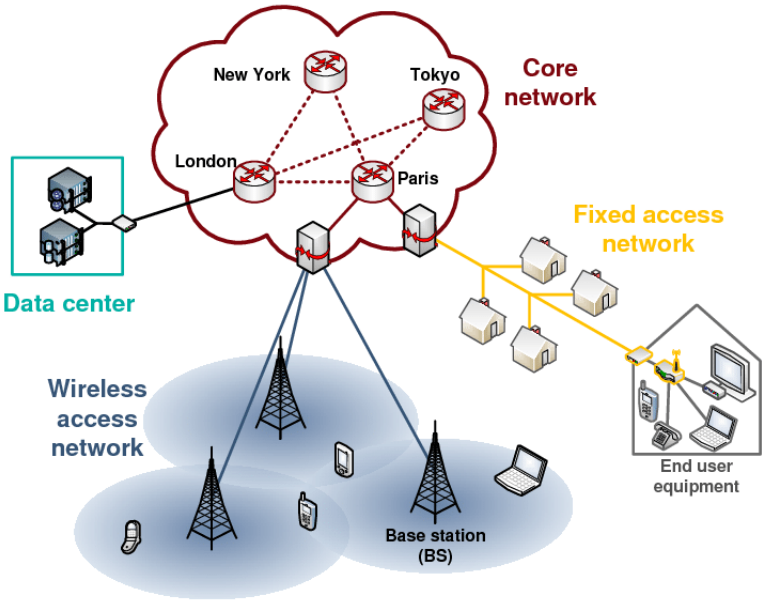


Figure 1.4: Network structure (fixed and wireless access networks, core networks and data centers) [84].

1.3.2 Network devices

Computer Networks consist of a large number of network devices such as routers, switches, and many types of middleboxes and links connecting them. In the following, we define some network nodes.

- **Network interface** is a software or a hardware interface connecting two parts of equipment. It has the ability to process low-level network information.
- **Repeater and hub** is an electronic equipment that receives a network signal, cleans it of undesirable noise, and regenerates it. The signal is re-transmitted at a higher power level, or to the other side of obstruction so that the signal can traverse longer distances without degradation.
- **Bridge** is a network device that connects and filters traffic between two network segments at the data link layer 2 of the OSI [1] model to create a single network.
- **Switch:** is a device that connects several segments (cables or fibers) in a computer and telecommunication network, making it possible to create virtual circuits. Switching is one of two frame transport modes within computer and com-

munication networks. Switches can route data between two connected network nodes only.

- **Router** is a device that aims to determine routes and forwards packets between networks by processing the routing information included in the packets. They have a total view of the network.
- **Modem:** (MOdulator-DEModulator) is a hardware device that transforms data into a suitable format for a transmission medium so that it can be transmitted from one computer to another.
- **Firewall** is a network device for controlling network security and access rules. Firewalls are typically configured to reject access requests from unrecognized sources.

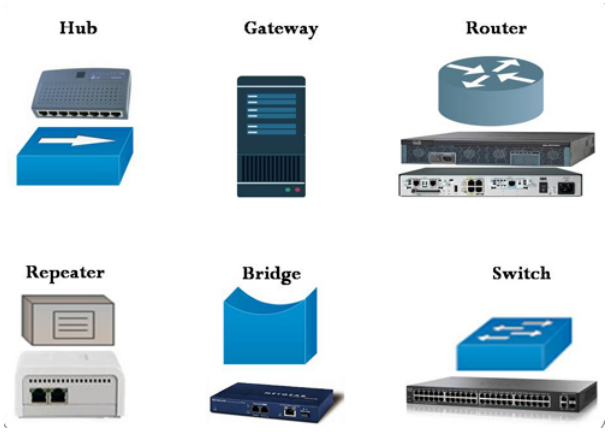


Figure 1.5: Network devices [3]

1.3.3 Network Function (NF)

Network function also called, service function (SF), is a network appliance that runs on the network application layer. SFs are used to treat, manipulate, or store data packages. Typical examples include firewalls, video optimizers, load balancers, parental control, etc. Traditionally, SFs were implemented on physical middleboxes, which are intermediary "devices" (such as applications, functions, machines, etc.) used to treat data packages differently from standard routers. In the following, we will give definitions of some network functions, published in memo [2] by Brian Edward Carpenter.

Network Address Translator (NAT) aims to dynamically assign a global and unique address to a host that does not have one.

Application-level gateway (ALG) needs to keep state for the sessions they are handling, and if the state is lost, the session will normally break irrevocably.

IP Firewall the simplest form of firewall is a router that screens and rejects packets based purely on fields in the IP and transport headers.

Proxies Fielding et al. [49], define a Web proxy as an intermediary program which acts as both a server and a client for the purpose of making requests on behalf of other clients.

1.3.4 Routing schemes

We can distinguish several routing schemes in order to route flow packets through a network. They change in how they transfer messages:

- **Unicast** transmits a message between a sender and a single destination.
- **Broadcast** transfers a message from one sender to all nodes in the network.
- **Multicast** transmits a message from one sender to a group of nodes that have expressed interest in receiving the message.
- **Anycast** transfers a message from one sender to anyone out of a group of nodes.
- **Geocast** delivers a message from a source to a group of nodes in a network based on their geographic location.

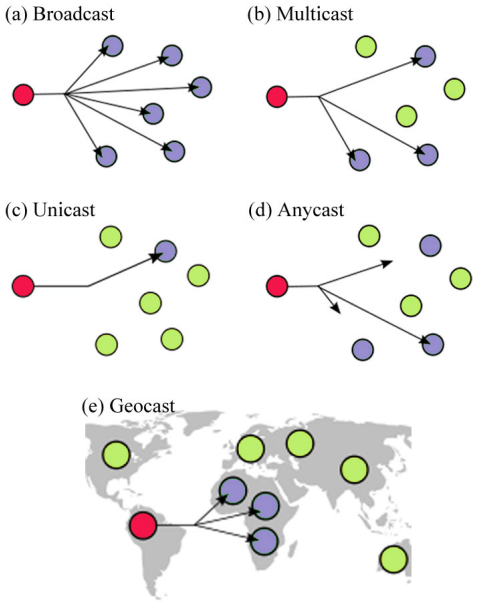


Figure 1.6: Routing Schemes [96]

Unicast is the dominant form of message delivery on the Internet. This thesis focuses on unicast routing scheme.

1.3.5 Network Function Virtualization (NFV)

Typically, telecommunication networks are composed of diverse elements such as switches, routers, and network appliances also called, middleboxes, that aim to transform, investigate, filter, and manage the traffic. The middleboxes are adopted in order to provide network functions. Some examples of middleboxes are: Proxies, Traffic Monitor (TM), Firewalls (FW), Intrusion Detection Systems (IDS), Load Balancers (LB), Network Address Translators (NAT), WAN Optimization Controller (WOC), Intrusion Detection Prevention System (IDPS) and Video Optimization Controller (VOC).

The hardware middleboxes imply large capital expenditure (CAPEX), as well as an operational expenditure (OPEX) [87], caused by the considerable number of these devices in the network which is comparable with the number of switches and routers [80]. Furthermore, the traffic has to be conducted in such a way that traffic flows pass through these devices, which causes long installation delays and high maintenance costs. Moreover, they do not allow to add new features, are energy-intensive, and have short life-cycles [71]. In average and for a large network, the middleboxes cost can reach up a million dollars over five years [124]. Besides, each middlebox provides

a different functionality i.e., there exists a heterogeneous set of these devices in the network, which necessitates a large management team with different expertises.

From 1990 to the year 2000, the network virtualization captivated the attention of network service providers and network operators. Network virtualization can be seen as a representation of one or more logical network topologies, which defines how the data should be transferred, in the same infrastructure. Using virtualization, we can use, for example, multiple logical routers on a single platform, and it also permits resource isolation in terms of CPU and memory.

In addition, network virtualization offers the possibility to transform the way operators plan, use, and maintain their network infrastructures. Its purpose is to deal with a significant number of specific hardware devices deployed in the operator's networks and the very high generated costs.

In network function virtualization technology the network functions are executed as software on commercial off-the-shelf equipment, this allows their instantiation on demand without the installation of new equipment. For instance, an open-source software-based firewall can be run on an x86 platform in a virtual machine [68].

Network Function Virtualization (NFV) makes it possible for NSPs to employ various Virtual Network Functions (VNFs) without installing new equipment [137]. In [35], authors present the history and the state-of-the-art of network virtualization, they present several layers (application, link, network) and levels (link, node) of virtualization and give some examples of virtualized networks. Li and Chen [89] give an overview of the history of NFV, presenting how middleboxes evolve to virtual network functions.

Chiosi et al. in [32] present a technical report in which they introduce the network function virtualization, its benefits, enablers and challenges. They explain how software replaced hardware in the network architecture. They also give some use cases and raise the challenges of the virtualization. Some works in NFV resource allocation emerged in very recent years. Exhaustive surveys are already given in [71] and [58]. Herrera and Botero [71] present a comprehensive state-of-the-art of Network Function Virtualization and Resource Allocation (NFV-RA) problem. They give a classification of the problem by considering all its variants, optimization objectives, solution strategies and application domain. The authors notice that the wide deployment of future network architectures based on NFV will depend largely on the success of resource allocation.

Figure 1.7: NFV substitute the network functions by network applications.

1.3.5.1 Network Function Virtualization architecture

The Network Function Virtualization architecture, defined by the European Telecommunications Standards Institute (ETSI), is composed of 4 layers, as shown in Figure 1.8, presented as follows:

1) **Network Function Virtualization Infrastructure (NFVI)** is the environment that provides the virtual resources required to support the execution of the Virtual Network Functions [31]. It can be physical (servers and switches) or virtual (virtual machines and virtual switches). This layer is divided into three parts:

   - **Hardware resources** it includes computing resources (servers and RAM), storage (disks storage), and networking resources (switches, routers, and firewalls).
   - **Virtualization layer** it decouples virtual from physical resources; it is responsible for abstracting physical resources into virtual resources. It permits the software to grow separately from the hardware.
   - **Software resources** it includes virtual computing resources, virtual storage, and virtual networking resources.

2) **Virtual Network Function (VNF)** it represents the virtualized network elements such that vrouter, vbase station, vfirewall, vIDS, etc. It can be connected or combined together to offer service chains.

3) **Management and Orchestration (MANO)** it includes the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the lifecycle management of VNFs [31]. It is composed of three sotware parts:

   - **Virtualized Infrastructure Manager (VIM)** it is responsible for controlling and managing the NFVI computing, network, and storage resources.
   - **VNF Manager:** it a software that aims to manage the life-cycle of VNF instances, is responsible for initializing, updating, querying, scaling, and terminating one or more VNF instances.
   - **NFV Orchestrator:** it manages the life-cycle of network services, including instantiation, policy management, performance management, and KPI monitoring.

4) **Operation Support System/Business Support System (OSS/BSS):** OSS handles network, fault, configuration, service, and element management. BSS deals with client, operations, order, billing, and revenue management. In the NFV architecture, the current BSS/OSS of an operator may be combined with the NFV Management and Orchestration using standard interfaces.

1.3.6 Software Defined Networking (SDN)

In traditional telecommunication networks, each switch (device) has its own data plane, to forward the traffic, and its own control plane, to decide where to send the data. Also, it has information only about the state of its neighbors to interact with them within the network. No single device has visibility of the whole network. Each switch has to work separately in order to decide where and how to send data, and then all devices synchronize together at a given time slot to route packets through the network.

Supposing that the device uses a command-line interface, this necessitates a manual configuration, which is very costly and time-consuming. Furthermore, having a considerable number of devices implies managing a considerable number of control and data planes. Moreover, if each one has a different operating system and interface, it will be challenging to create one application that can be installed on each of them, allowing us to change its control and data plane. Therefore, network management is very challenging [110]. This leads to increasing costs in terms of CAPEX and OPEX, in order to minimize the QoS (e.g., end-to-end latency), and makes packet forwarding very complicated within a network [48, 111].

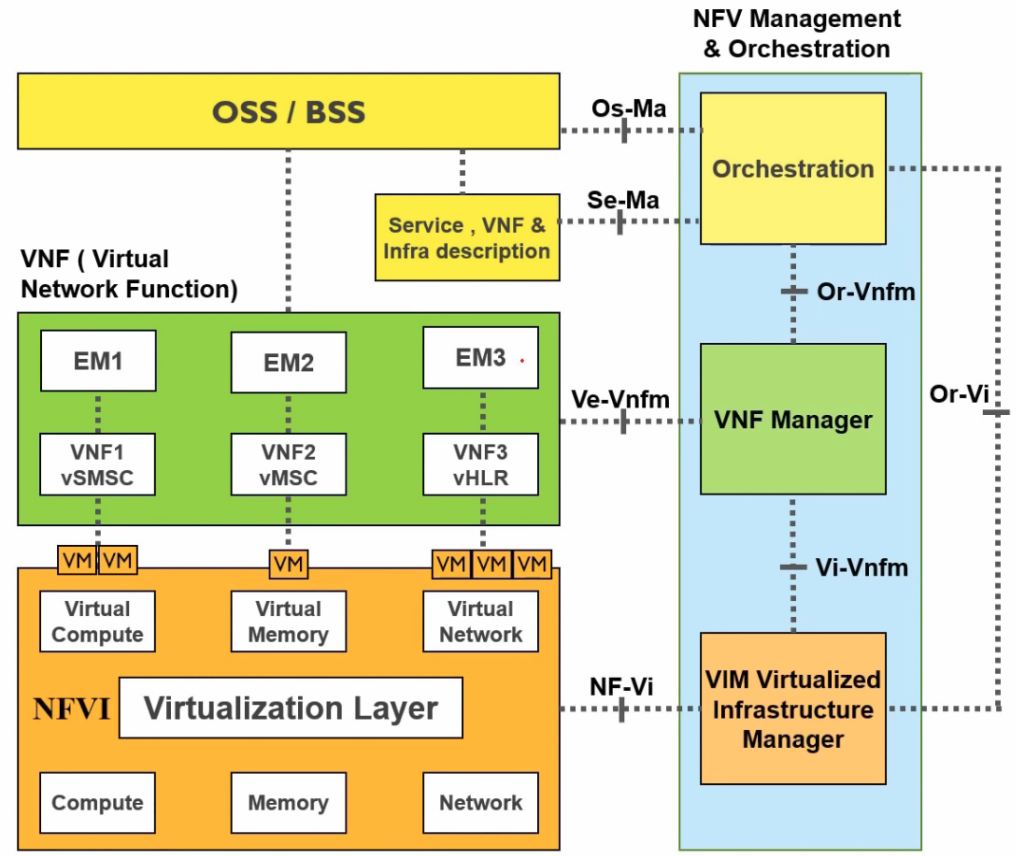


Figure 1.8: NFV Architecture [4]

Software-defined networking is a new paradigm that simplifies network management, and makes the deployment of new services easier by separating the data plane from the control plane. This is done by introducing one controller having a global view of the network state. Several papers were proposed in order to define, explain, and give examples of Software-Defined Networking (SDN) architecture. A software-based controller is responsible for managing the forwarding information of one or more switches [85]. As mentioned by Rao in [114], SDN is a technology that gives the network designer the freedom to refactor the control plane. The author gives the relationship between Network Virtualization, Network Function Virtualization, and Software-Defined Networking, showing how SND plays a significant role in NV and NFV. A list of companies and their SDN products/solutions is presented in [114]. Nunes et al. in [110], explain that in SDN architecture, the network intelligence is logically centralized in software-based controllers (the control plane), and network devices become simple packet forwarding devices (the data plane) that can be programmed via an open interface.

According to authors in [56], Software-Defined Networking provides a new dynamic

network architecture that transforms the traditional networks into service-delivery platforms. They also show the benefits of OpenFlow-Based Software-Defined Networks. OpenFlow is the first standard protocol interface designed explicitly for SDN, providing high-performance and granular traffic control. OpenFlow provides an open protocol to program the flow-table in different switches and routers [100], and allow direct access and manipulation of the forwarding plane of network devices. In [98] authors give a list of the most important technical challenges for the development and deployment of SDN. Guerzoni et al. in [66], present the relationship between NFV and SDN. Li and Chen [89] propose a survey that investigates the development of NFV under the software-defined NFV architecture.

Within an SDN architecture, the traffic engineering mechanisms become much more efficiently implemented with respect to legacy network approaches (e.g., IP, ATM, and MPLS) [129]. Furthermore, all network information are easily retrieved; network elements are dynamically and proactively programmed without having to handle them individually [9]. Moreover, SDN provides various benefits, namely, fast detection time [123], re-routing [21] and shows its capacity to identify and grow from failure under the requirement below 50 ms [14]. With SDN, the network becomes programmable, dynamic, and flexible according to frequent network changes.

Examples of centralized controllers are: Beacon [47], Ryu [128], OpenDayLight [102], and Maestro [28]. Nevertheless, having a single controller in the network signifies having a unique point of failure; if the node (controller) fails, there is no other point that makes decisions and manages the data plane. Accordingly, the notion of a distributed controller is proposed. It represents a set of nodes that can be either centralized or physically distributed. Some distributed controllers are Onix [83], ONOS [21], and DISCO [112].

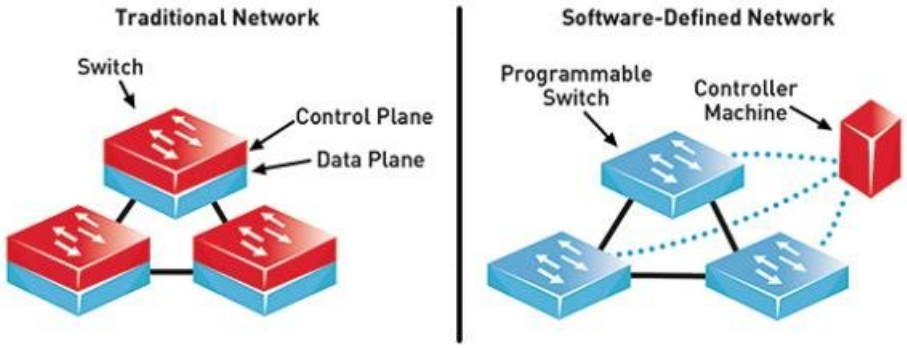


Figure 1.9: Traditional network Vs. SDN network [97]

1.3.6.1 Software Defined Networking architecture

SDN architecture is composed of three main layers (see Figure 1.10): Data Plane, Control Plane, and Application Plane.

- **Application plane** allows network managers to quickly configure, manage, secure and optimize network resources via dynamic automated SDN programs.
- **Control plane** exercises direct control over the data plane using an Application Programming Interface (API), which defines the information exchange between the two planes.
- **Data plane** is a set of network elements such as routers, switches, and middleboxes that offer efficient and programmable packet forwarding devices without any software to make autonomous decisions.

OpenFlow is a protocol represented by an open interface that manages to update the flow table of the device.

1.3.7 Service Functions Chaining (SFC)

Service Function Chain (SFC) is defined as a sequence of network functions that should be traversed by a given data flow in a predefined order [113]. For example, the network administrator may specify a policy that all HTTP traffic should follow the policy chain: "*firewall* $\rightarrow$ *IDS* $\rightarrow$ *proxy*" [87]; Also for instance, the Intrusion Detection System must inspect packets before compressing or encrypting them [121].

For each service, a specific set of required functions needed to handle its packets, and their order is defined by the service function chain; this allows flexible management and classification of services and permits to define the requirements associated with each one [101].

Traditionally, the service function chain represents an ordered sequence of hardware devices that must be traversed by the data flows to support some service. Nevertheless, new hardware devices should be used, placed, and connected to the network elements if a new service is required. This generates additional costs and is time-consuming [89].

Thanks to NFV and SDN, the VNFs could be installed on-demand on network nodes, and the routing paths associated with the Service Function Chains could be computed

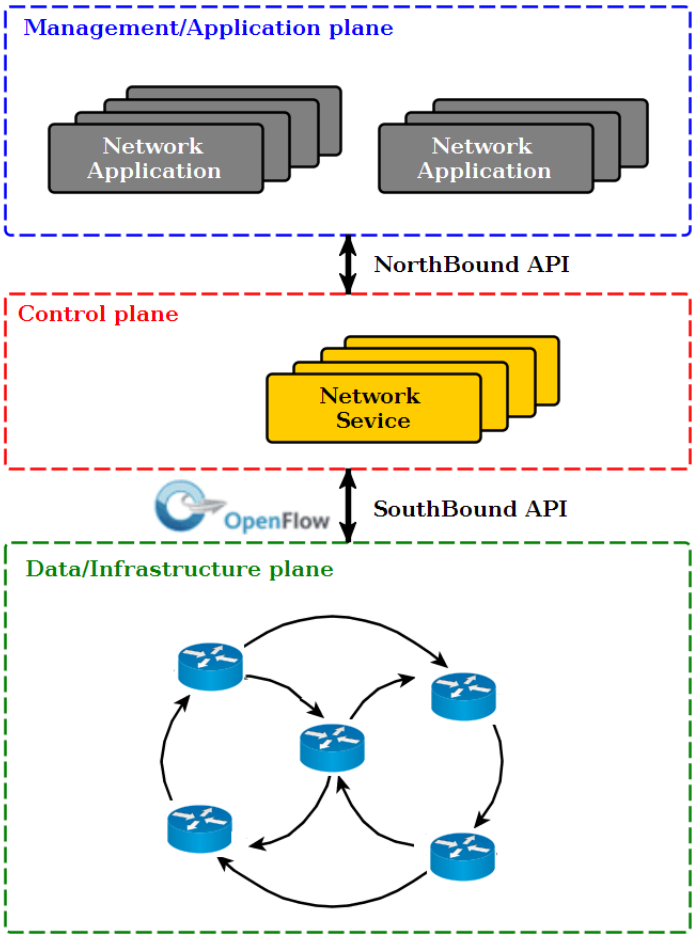


Figure 1.10: SDN architecture

dynamically to encounter the installed VNFs [11]. This new flexibility in network management generates hard decision problems combining the choice of locations to deploy the VNFs, and the computation of efficient routing paths meeting the VNFs in the right order. In the Virtual Network Embedding problem, the service chains can be seen as virtual graphs (paths) to embed into the original network. Each node in the virtual graph represents a VNF, and the links between nodes represent the order. Bhamare et al. in [22] define Service Function Chaining as a mechanism that allows various service functions to be connected to form a service allowing operators to benefit from the software-defined virtualized infrastructure. Medhat et al. in [101], explain that the optimization models are needed for SFC configuration and allocation to achieve optimal network performance, satisfy user demands, accommodate SFCs dynamically, and minimize network costs. Zhang et al. [139] proposes a typical SFC functional framework integrating SDN and NFV including service modeling and resource allocation.

Many works were also proposed for the Service Functions Chaining; for example, a near-optimal solution is provided by a heuristic in [88] for the real-time NFV, aiming to maximize the total number of requests assigned to the cloud for each SFC, while the deadline constraints are satisfied. Hantouti et al. in [69] present different research problems related to SFC; they investigate several key challenges that should be addressed to realize more reliable SFC operations. Examples include SFC path selection, placement of service functions problem, traffic steering problem, Resource allocation problem, SFC orchestration and management problem, maximizing the Quality-of-service, and security problem. Bhamare et al. in [23] present an analytical study of VNF and SFC. Specifically, they study the problem of placing service function chains over the network function virtualized platform in a multi-cloud scenario. The objective is to reduce the total delays to the end-users and the total cost of deployment for service providers in inter-cloud environments. They propose an ILP to model the problem and a novel Affinity-based approach to overcome with limited applicability of the model.

Hantouti et al. [70] present a comprehensive analysis and classification in three categories of the current SDN-based SFC approaches using efficiency criteria such as deployment cost, scalability, and flexibility.

Some studies have been proposed in the literature in order to define how to place an ordered chain of functions in a network. Different objective functions were considered such as: minimizing the number of activated nodes, the end-to-end latency [103], the energy consumption [76], or the bandwidth [75].

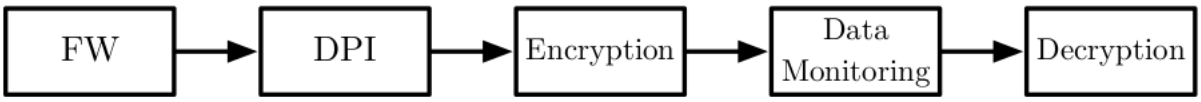


Figure 1.11: Example of Service Function Chain [71]

1.4 Literature review

Although the Virtual Network Functions Placement and Routing Problem appeared relatively recently in the telecommunication context, the problem can be related to a few previously studied combinatorial optimization problems. The problem is composed of two NP-hard problems, namely, the multi-commodity flow problem and the facility location problem, with some additional constraints, such as anti-affinity rules,

chaining or precedence constraints, and linking constraints. The problem is also called in the literature by: *VNFs placement problem*, *VNFs placement and chaining problem*, *VNFs service chaining problem*, or *VNFs placement and path selection with precedence constraints*.

Problem variants

Different objective functions variants In the context of NFV and SDN, the underlying problem consists of finding an optimal installation of VNFs, so that the given traffic requests can be routed within the given network infrastructure while respecting the SFC constraints [5, 67, 103, 105]. Many different variants of the problem have been studied in the recent literature. When it comes to the definition of the objective function, some authors consider the minimization of the number of activated nodes, the VNFs installation costs [11], the end-to-end latency [103], the energy consumption [76], or the bandwidth [75].

Simplest variants Several straightforward variants of the problem were studied, for example, in [7], authors consider the VNF chaining problem with a single type of VNFs for all demands. Furthermore, they suppose that all traffic requests represent exactly the same service. They propose an extension of their work that allows using multiple VNFs and different services. Also, Casado et al. [29] consider the problem with a single type of VNF and present a heuristic algorithm towards solving the placement problem. Basta et al. [18] study the network functions placement problem. They analyze possible placements of virtualized gateways or decomposed gateway functions with respect to delay and impose network load in a mobile core network. The authors propose a path-flow model that minimizes the network load and satisfies the data-plane delay budget. Similarly, Bouet et al. [27], consider the problem of placing the virtualized Deep Packet Inspection in SDN networks. They propose a method based on genetic algorithms, that optimizes the cost of DPI engine deployment, minimizing their number, the global network load and the number of not analyzed flows, while considering arc capacity constraints. Moreover, Mijumbi et al. [104] define the problem of online mapping and scheduling functions in an NFV environment, but they ignore the routing aspect. They consider objective functions involving the total processing time and the cost or revenue of utilized resources. They propose a set of greedy algorithms and tabu search-based heuristic. Furthermore, some variants deal with specific network topology, for example,

Tomassilli [129], proposes two approximation algorithms for the tree network topology to address the SFC placement problem.

Chainingless variants Adding the chaining (precedence) constraints to the problem make it very difficult to solve. In order to deal with the complexity of the problem, some works propose methods that allow them ignoring this family of constraints. For example, Sallam et al. [118] study The Shortest Path and Maximum Flow Problems Under Service Function Chaining constraints for which physical and virtual network functions are considered. The authors solve the SFC-constrained shortest path problem by transforming the graph into a layered graph. This transformation aims to deal with the precedence between VNFs; then, they search for the shortest path in the new graph. They also solve the problem as a fractional multi-commodity maximum flow problem. They propose an ILP formulation to model the VNFs placement from a maximum flow perspective. Similarly, Gouareb et al. [64], propose an MILP to model the Virtual Network Functions Placement and Routing problem while minimizing the delay cost on the arcs. They consider that all VNFs associated to each SFC are located at the same node to deal with precedence constraints, which means that the problem reduces to the optimal routing and location of each service request. Whereas, Cohen et al. [36], provide a near-optimal approximation algorithm to address the problem of placing VNFs on the physical network without order constraints. Their objective is to minimize installation and path costs. Moreover, in order to minimize the number of used network functions, Sang et al. [119] fix the routing paths and focus their attention on the placement of VNFs without considering any chaining constraints.

Closest variants Allybokus et al. [11] consider a generic version of the VNF placement and routing problem in which many features (partial order on the functions, bound on end-to-end latency, conflicts between network functions, resource limitations and processing capacities) are already taken into account. They proposed several MILP formulations, either to minimize the total deployment cost or to minimize the number of rejected demands. A heuristic algorithm based on a continuous relaxation of one formulation is also proposed and appears to be very efficient on instances derived from the Geant topology. However, the MILP models do not seem to be strong enough for direct resolution with the Cplex solver. Similarly, Addis et al. [5] formulate the VNFPRP as an MILP, taking into account flow constraints, latency constraints, node and function capacity constraints, while minimizing the number of used links to route traffic and the number of used nodes to install functions. The authors also propose a math-heuristic approach for the problem. Contrary to our work, they allow for compression/decompression at the VNF nodes.

Furthermore, Mehraghdam et al. [103] showed that the problem could be considered as the Location-Routing Problem [107], that aims to create several paths between different VNFs, and then connect them to get source-destination paths satisfying precedence constraints. The location-routing problem can be seen as a multi-commodity flow problem in which commodities can share some VNFs while minimizing node, edge, or path costs. The authors model the placement of chained VNFs problem as a Mixed Integer Quadratically Constrained Program with three different objectives: maximizing the remaining data rate, minimizing the number of activated nodes or the latency of the paths. They consider first the placement part of the problem; then in order to satisfy the chaining constraints, they create a path between installed VNFs. Their model was tested on small instances and solved using Gurobi Optimizer. Moens and De Turck in [105], study the Virtual Network Functions Placement problem. They propose an ILP model where both physical and virtual resources are allocated to the function chains. The resulting ILP is tested on very small instances using Cplex solver. Contrary to Bari et al. [15] that provide a dynamic programming based heuristic to solve large instances of VNF placement problem.

In order to deal with chaining constraints, Huin [74] uses a layered graph, and based on that he proposed an ILP formulation to model a variant of the VNFPRP. In this variant the routing constraints are defined using flow constraints, and link and latency constraints are considered. The objective function aims to minimize the bandwidth requirements. The proposed formulation does not scale well for medium to large networks. Thus, the author proposes a column generation model.

In other works Tomassilli et al. [131], consider another variant of the problem for which latency, precedence, flow, link and node capacity constraints are considered. Authors propose an ILP and a column generation based model to model the problem.

Variants for multicast service Similar works were proposed for the problem with multicast service, for instance, Kiji et al. in [82], model the VNFPR problem as an ILP for multicast service chaining based on merging multiple service paths. They minimize the cost associated with VNF placement and link usage in providing multicast service chains. They also develop a heuristic algorithm that can find a feasible solution within a reasonable time. Zhang et al. in [140], propose a routing algorithm with approximation factor 2 to solve the Service Function Chain Enabled multicast Routing Problem. In [30], the authors propose a Steiner tree-based algorithm for NFV-enabled multicast communication. The proposed algorithm allows for reducing the number of deployed VNFs and cost minimization. Alhussein et al. in [10] treat the VNF Placement and Multicast Traffic Routing problem, they propose an MILP formulation to

model it while minimizing VNFs and arcs deployment under the physical network resource constraints, flow conservation, and VNF placement constraints. To reduce the computational complexity of the problem, heuristic algorithms are proposed, considering both the single path and multipath routing. In [81], Kiji et al. model the VNF placement and routing model for multicast SC as an ILP problem that allows merging SC paths of different services.

Other variants Some variants propose to place VNFs in specific nodes of the network, for example: Authors in [136], tackle the problem of finding the optimal VNF placement for VNF chaining in the packet/optical data-centers while minimizing the number of performance-optimized data-center an NF chain needs to visit. They propose a binary linear program to model it and a heuristic that demonstrates its effectiveness under various scenarios compared to a simple first-fit algorithm. Also, Nikam et al. [109], study the VNF Service Chaining problem in Optical Data Center Networks. They propose an ILP formulation composed of three sub-problems to model the problem and an efficient heuristic to solve it. The problem of Placing VNFs in the cloud for 5G networks is discussed in [116]. A multi-objective ILP formulation is proposed to model the problem which aims to minimize the number of used VNFs and maximize the Quality-of-Experience. The problem treats the relocation of VNFs. In order to achieve sub-optimal placement of VNFs within polynomial time, an Ant Colony Optimization (ACO) algorithm was proposed. Whereas, In [93], authors formalize the Network Functions Placement and Chaining problem on physical infrastructure as an ILP model. They also propose a heuristic to guide the ILP solver towards feasible and near-optimal solutions efficiently. The objective is to minimize the number of deployed VNFs. Lin et al. [90], study the network function virtualization with end-to-end request realization (NFV-RR) problem; based on a use case proposed in [108], they evaluate the performance of the placement of VNFs in terms of its ability to support end-to-end requests with limited physical resources. They propose an MILP to model the problem.

1.4.1 Overview of the related works

In this subsection, we selected the most relevant papers from the literature related to the VNFPRP that we compare in Table 1.1. The columns in Table 1.1 are defined as follow:

Ref	Reference	PP	Placement problem
RP	Routing Problem	PC	Precedence constraints
FCC	Function capacity constraints	NCC	Node capacity constraints
ACC	Arc capacity constraints	CC	Conflict constraints
LC	Latency constraints	E	Exact method
H	Heuristic method	FC	Minimizing functions costs
NC	Minimizing node costs	AC	Minimizing arc costs
MC	multicast communication		

Ref	PP	RP	PC	FCC	NCC	ACC	CC	LC	E	H	FC	NC	AC	MC
Thesis	x	x	x	x	x		x	x	x	x	x	x		
[5]	x	x		x	x	x		x	x		x	x	x	
[11]	x	x	x	x	x	x	x	x	x	x	x		x	
[30]	x				x	x		x	x		x	x		
[39]	x	x	x	x		x		x	x	x	x		x	
[61]	x	x		x	x	x				x	x		x	x
[64]	x	x	x		x	x		x		x			x	x
[74]	x	x	x		x	x			x			x		
[81]	x	x	x	x	x	x		x	x	x		x		x
[82]	x	x	x	x	x	x		x	x	x	x	x		x
[90]	x	x			x	x				x	x	x		
[93]	x	x	x	x	x	x		x		x	x	x		
[103]	x	x		x	x	x		x	x	x		x	x	
[104]	x	x	x	x	x					x	x	x		
[105]	x			x	x		x		x		x	x		
[127]	x	x		x	x				x		x		x	
[130]	x	x	x		x	x	x		x	x			x	
[131]	x	x	x		x	x		x	x				x	

Table 1.1: Overview of related works

Chapter 2

The Virtual Network Functions Placement and Routing Problem

This chapter is dedicated to the Virtual Network Functions Placement and Routing Problem. We start this chapter by defining the problem, its basic properties, and by giving an illustrative example. We show that the problem is strongly NP-hard even in its simplest version, and propose an MILP compact formulation to model it. We test the proposed model on a set of realistic instances derived from SNDlib.

Contents

2.1 Motivations

The two new technologies, SDN and NFV, have being adopted very fast by most actors in the telecommunication industry: they propose various advantages such as cost and energy minimization, flexibility, and dynamic decisions in new networks. The combined use of both SDN and NFV is gaining the attention of the Network Service Providers (a company that owns, operates and sells access to internet backbone infrastructure and services). Several academic and industrial teams are investigating ways to maximize SDN and NFV benefits. For example, Orange introduced an SDN based coverage offer for 75 countries to help companies immediately provision branch offices with Virtual Network Functions (VNFs) [99]. The leading US Telecommunication Operator (AT&T), has fixed as an objective to virtualize up to 75% of its network by 2020 [126]. Also, Huawei, in the last years, deployed 560 SDN/NFV commercial projects around the world [73]. Furthermore, in 2013, Google declared the use of SDN to interconnect its data centers all over the planet [77]. Using this technology, Google was able to achieve several advantages, including efficient network control, more accessible and faster innovation cycles of networks and services, better network exploitation, and a decrease of both OPEX (Operating Expenditure) and CAPEX (Capital Expenditure).

Both technologies create new challenging problems and on eof them is the Virtual Network Functions Placement and Routing problem. This problem can be decomposed into two *NP-hard* problems, namely, the multi-Commodity Flow (MCF) problem with unsplittable commodity paths and the Facility Location (FL) problem (as shown in Figure 2.1). Further constraints can be added, such as, the service chain and conflict constraints. All these technical constraints make the problem very challenging. Specifically, the challenges are: (i) where to install the required VNFs on network nodes at minimum cost (ii) defining, for each demand, a routing path that traverses suitable nodes in the correct order to satisfy the requirements of a given service chain, and (iii) taking into account network load and other dynamic characteristics when routing through existing VNFs [46].

The methods proposed in the literature to deal with the problem mainly consist in heuristics, or two-stage methods, that solve the routing part and the installation part of the problem separately. Alternatively compact MILP formulations have been proposed that cannot solve large scale instances (see, [5, 11, 64, 72, 127]). Generally, to overcome the fairly limited computational performance of compact formulations, and to improve the capabilities of finding exact solutions for larger instances, reformulation and decomposition methods are recommended in the literature (see [24], [34], [117], [133], [135]).

Motivated by all of the above, the main objective of this thesis is to model and solve the VNFPRP using exact and heuristic methods. To this end, we use some decomposition and reformulation approaches from the literature in order to solve large scale instances.

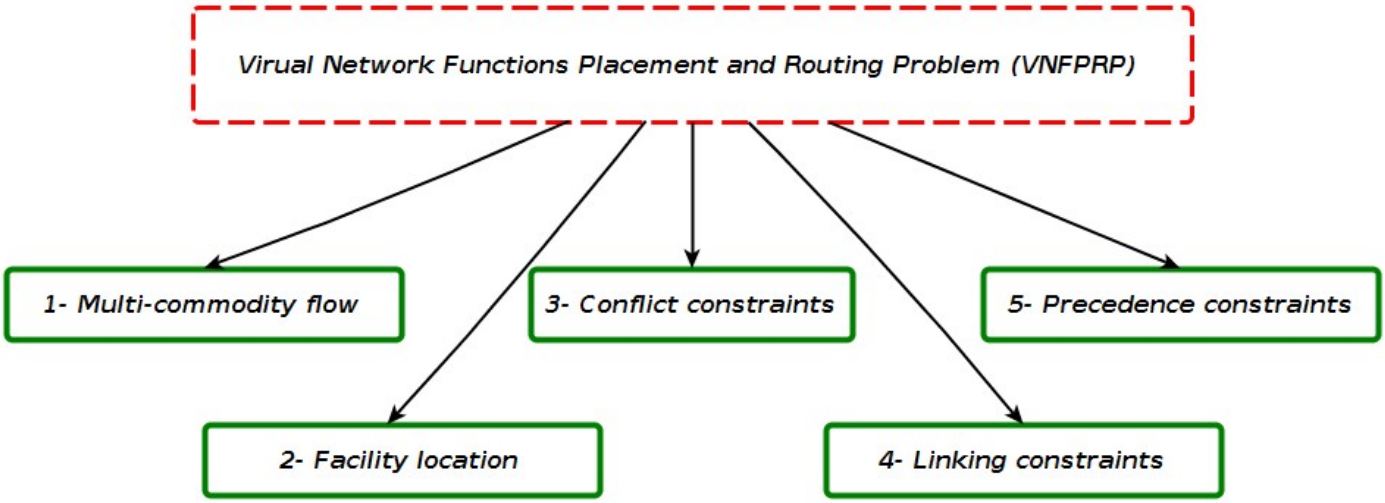


Figure 2.1: VNFPRP is composed of multiple subproblems and constraints.

Outline of the chapter The chapter is organized as follows. In Section 2.2, we define the problem and introduce the main notations. In Section 2.3, we present the problem's properties. An illustrative example is shown in Section 2.4. Section 2.5 is devoted to the problem's complexity analysis. The compact MILP formulation is introduced in Section 2.6. In Section 2.7, we present some computational results and we end the chapter by some conclusions and remarks in Section 2.8.

2.2 Problem definition

2.2.1 Notation

The telecommunication network is modeled as a bi-directed graph $G = (N, A)$. The set of all physical locations equipped with hardware devices allowing VNFs installation is denoted by N and called the set of nodes. The set representing all connections between nodes is called the set of arcs and is denoted by A. At each node $u \in N$ at most c_u

($c_u \in \mathbb{N}$) VNFs can be installed, and an activation cost $\psi_u > 0$ has to be paid. The arc latency $l_{uv} \geq 0$ is defined for each arc $(u, v) \in A$.

Let F denote the set of all virtual network functions. Each VNF $f \in F$ has a limited (bandwidth) capacity $m_f \in \mathbb{N}$ to manage the traffic demand. For installing each single copy of f at a node $u \in N$, an installation cost $\psi_u^f > 0$ has to be paid. For the set of all traffic requests/commodities, denoted by C, each commodity $k \in C$ is characterized by: a source node s_k; a destination node d_k ($s_k \neq d_k$); a bandwidth $b_k > 0$; a maximum latency value $l_k > 0$; a subset of VNFs, $F^k \subseteq F$, and a set of incompatibility constraints $\mathcal{A}^k$ between VNFs. Finally, recall that each commodity requires a specific Service Function Chain to handle its data packages in a specific order. The fact that the VNF f has to be executed before VNF g is expressed by $f \prec_k g$ in the path associated to the commodity k. Table 2.1 summarizes our notation.

Sets		
N	:	Set of all nodes.
A	:	Set of all arcs.
C	:	Set of all commodities (traffic requests).
F	:	Set of all Virtual Network Functions.
$(F^k, \prec_k)$	:	Ordered set of VNFs associated with commodity k, $k \in C$.
$\mathcal{A}^k$	:	Set of pairs of VNFs which are in conflict for commodity k, $k \in C$.
Parameters		
m_f	:	Capacity of VNF f, $f \in F$.
c_u	:	Capacity of node u, $u \in N$.
l_{uv}	:	Latency of links (u, v), $(u, v) \in A$.
ψ_u^f	:	Installation cost of VNF f at node u, $f \in F$, $u \in N$.
ψ_u	:	Activation cost of node u, $u \in N$.
s_k	:	Source node for commodity k, $k \in C$.
d_k	:	Destination node for commodity k, $k \in C$.
l_k	:	Maximum latency delay of commodity k, $k \in C$.
b_k	:	Bandwidth of commodity k, $k \in C$.

Table 2.1: Main notation, parameters and sets.

2.2.2 Problem definition

Definition 1 (VNFPRP) *The Virtual Network Functions Placement and Routing Problem consists of finding for each commodity $k \in C$, the placement of VNFs $f \in F^k$ at nodes, and the routing paths so that the sum of the VNF installation costs ψ_u^f plus*

the sum of node activation costs ψ_u is minimized. Moreover, the following constraints have to be satisfied:

- ***Node capacity constraints*** *each node $u \in N$ has an installation capacity c_u, which means that the number of VNFs installed at u should not exceed c_u.*
- ***VNF capacity constraints*** *each function $f \in F$ has a capacity m_f to manage the amount of data. The sum of bandwidths of all commodities handled by f should be below m_f.*
- ***Routing constraints*** *the $s_k - d_k$ routing path associated with each commodity $k \in C$ should be elementary.*
- ***End-to-end latency constraints*** *the sum of arc latencies belonging to the routing path of each commodity $k \in C$ should not exceed the given upper-bound $l_k \in \mathbb{R}_+$.*
- ***Installation constraints*** *each VNF $f \in F^k$ required for commodity $k \in C$ should be installed at one of the nodes of the routing path, but not at the source node.*
- ***Precedence constraints*** *for each commodity $k \in C$, the VNFs composing its Service Function Chain should be traversed in the right order by the routing path.*
- ***Conflict constraints*** *(also called anti-affinity/incompatibility constraints): For different reasons (e.g. resiliency, privacy) [130] VNFs pairs that are in conflict should not be installed at the same node.*

The VNFPR problem aims to minimize the sum of VNF installation costs ψ_u^f plus the sum of node activation costs ψ_u. A solution satisfying the constraints described above is said to be feasible for the problem. Furthermore, if the associated cost is minimized, then, it is called optimal.

2.3 Proprities of the VNFPRP

In this thesis, we consider the following assumptions:

- In NFV environment, several, heterogeneous and similar virtual network functions can be installed at the same node in the graph.

- Link capacities are omitted, due to the fact that we are not dealing with the *strategic* network design phase (which has to be done much earlier in the planning process), but with a *tactical* planning. The network is designed (by making decisions regarding which links need to be installed, and with which capacity) so that it can accommodate a large amount of traffic. This is a long term process involving expensive investments and possible expansion of capacities (if it turns out that some links are heavily loaded). On a shorter time range, various services are managed to make the best possible use of existing resources, but the transmission/transport network is usually not the bottleneck (as the service demands are not of the same order of magnitude as the link capacity). Typically, network capacities are huge and shared by many different services. Thus, in this thesis we safely assume that there are enough link capacities in the existing network to handle all demands.

- We consider the unicast routing scheme.

- All routing paths should be elementary (without circuit). The use of elementary paths is linked to the protocols used, and to a choice of operators to simplify the supervision of flows.

- Each commodity requires at least one VNF, otherwise, these commodities can be eliminated (the routing problem can be seen as the shortest path problem).

2.4 Illustrative example

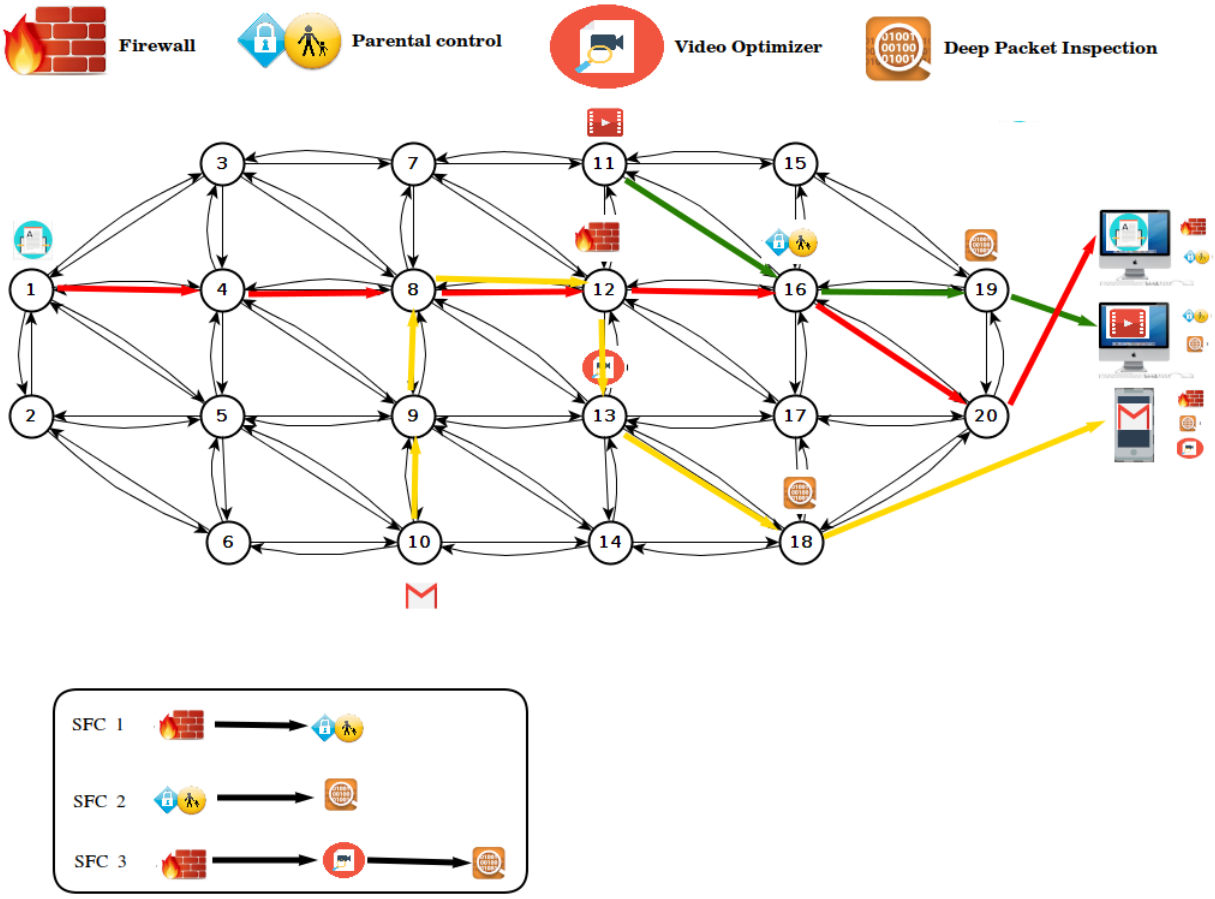


Figure 2.2: Example

The bi-directed graph in the example represents a telecommunication network containing 20 nodes, we suppose that we have a set of four virtual network functions, $F = \{Firewall,\ Parental\ control,\ DPI,\ Video\ optimizer\}$. The example in Figure 2.2, shows that there are three traffic requests representing three different services, such as *RA: "Reading news", WV: "Watching video" and CM: "Checking mailbox"*. Each of these service requires a set of virtual network functions to handle its data packets: *{Firewall, Parental control}, {Parental control, DPI} and {Firewall, DPI, Video optimizer}*, respectively. The associated service functions chain with each service are "$SFC_1 : Firewall \prec_{RA} Parental\ control$", "$SFC_2 : Parental\ control \prec_{WV} DPI$", and "$SFC_3 : Firewall \prec_{CM} Video\ optimizer \prec_{CM} DPI$", respectively.

Pairs of nodes (1, 20), (11, 19), and (10, 18) represent (source, destination) nodes associated with *RA*, *WV* and *CM*, respectively. From Figure 2.2, the red routing path associated with the service *RA*, represents an elementary path passing first through the node 12 on which the VNF *Firewall* is installed first, and then through node 16 at which the VNF *Parental control* is installed. In the same way, the green (resp. yellow) path goes from the source node 11 (resp. 10) to the destination node 19 (resp. 18) and passes through the nodes on which the VNFs are installed in the order satisfying the SFC_2 (resp. SFC_3) constraints. Moreover, we can see that services *RA* and *CM* share the same VNF *Firewall*. Also, services *RA* and *WV* share the VNF *Parental control*. Furthermore, from the example, we observe that the VNFs associated to each service cannot be installed on the source node, while these VNFs can be installed on the destination node.

2.5 Complexity analysis

Problem complexity

In this section we show that the problem under consideration is strongly NP-hard, even for the variant in which no conflicts between functions and no node and VNFs capacities are imposed.

Theorem 2.1 *The Uncapacitated Virtual Network Functions Placement and Routing Problem is strongly NP-hard, even for a single commodity and without latency, conflict, and precedence constraints.*

Proof. The proof is done by a polynomial reduction from the Uncapacitated Facility Location Problem (UFLP), which is an NP-hard problem in a strong sense (see, Theorem 3.1 in [40]).

Given a set of n facilities (sites) and m customers, let ψ_i be the cost of opening the facility i and ψ_i^j the cost of assigning customer j to facility i. We suppose that all facilities have unlimited capacity. The UFLP consists in finding which of the n facilities to open and how to assign the customers to open facilities so that the facility opening cost plus the assignment cost are minimized.

To reduce the UFLP to the uncapacitated VNFPR problem, we consider an arbitrary connected graph $G = (N, A)$ (G can be a complete graph) with $|N| = n$ nodes (with a

one-to-one correspondence between facilities from I and nodes in N) and a set F of m virtual network functions. We use one commodity $|C| = 1$, and let $s_1 = 1$ and $d_1 = n$ be two distinct nodes from N. We set the latency $l_1 = +\infty$ and assume that there are no precedence constraints (i.e., any ordering of functions from F is feasible). The transformation details are given in Table 2.2.

UFLP	1-to-1 corresp.	Uncapacitated VNFPR
–	$\Longleftrightarrow$	$\|C\| = 1$
I	$\Longleftrightarrow$	N
J	$\Longleftrightarrow$	F^1
Facility i	$\Longleftrightarrow$	Node i
Customer j	$\Longleftrightarrow$	Virtual network function j
Open facility i	$\Longleftrightarrow$	Activate node i
Customer j is supplied by facility i	$\Longleftrightarrow$	Function j is installed at node i
ψ_i cost of opening the facility i	$\Longleftrightarrow$	ψ_i activation cost at node i
ψ_i^j cost of assigning cust. j to facility i	$\Longleftrightarrow$	ψ_i^j installation cost of function j at i
–	$\Longleftrightarrow$	G=(N, A), connected graph
–	$\Longleftrightarrow$	$s_1 = 1, d_1 = n, l_1 = +\infty$

Table 2.2: Transformation details

This transformation is polynomial in the number of facilities and customers. An instance of the uncapacitated VNFPR defined in this way consists of installing all virtual network functions from F on a subset of nodes from N, while minimizing costs of installation of VNF and node activation costs. Thus, there is a one-to-one correspondence between an optimal solution of the uncapacitated VNFPR and the optimal solution of the uncapacitated facility location problem with exactly the same solution value. This proves that the Uncapacitated Virtual Network Functions Placement and Routing problem is *NP-hard*. ■

2.6 Compact MILP formulation

In this section, we propose a compact (i.e., polynomial in size) MILP formulation to model the VNFPRP. We fist describe the variables necessary to model it and then the set of constraints.

2.6.1 Decision variables

The set of variables required in our MILP formulation is described in Table 2.3:

Variables		Type
x_u^{fk}	1, if the virtual network function f is installed at or before node u for commodity k; 0, otherwise.	Binary
y_u^{fk}	1, if virtual network function f is installed at node u for commodity k; 0, otherwise.	Binary
w_u	1, if node u is activated; 0, otherwise.	Binary
t_{uv}^k	1, if arc (u, v) is taken in the path associated with commodity k; 0, otherwise.	Binary
z_u^f	number of VNF f installed at node u.	Integer

Table 2.3: Decision variables of the compact MILP formulation

2.6.2 Mathematical model

The VNFPRP can then be modeled as follows:

$$(P): \quad \min \quad \sum_{u\in N}\sum_{f\in F} \psi_u^f \; z_u^f + \sum_{u\in N} \psi_u \; w_u \tag{2.1}$$

$$\sum_{(u,v)\in A} t_{uv}^k - \sum_{(v,u)\in A} t_{vu}^k = \begin{cases} -1 & \text{if } u = d_k, \\ 1 & \text{if } u = s_k, \\ 0 & \text{otherwise.} \end{cases} \qquad k \in C, \quad u \in N \tag{2.2}$$

$$\sum_{(u,v)\in A} t_{uv}^k l_{uv} \leq l_k, \qquad k \in C \tag{2.3}$$

$$\sum_{f\in F} z_u^f \leq c_u \; w_u, \qquad u \in N \tag{2.4}$$

$$\sum_{k\in C} y_u^{fk} b_k \leq m_f z_u^f, \qquad f \in F, \quad u \in N \tag{2.5}$$

$$y_u^{fk} + y_u^{gk} \leq 1, \qquad k \in C, \quad (f,g) \in \mathcal{A}^k, \quad u \in N \tag{2.6}$$

$$(t^k_{uv} - 1) + (x^{fk}_v - x^{fk}_u) \leq y^{fk}_v, \qquad k \in C, f \in F^k, (u,v) \in A \tag{2.7}$$

$$x^{gk}_u \leq x^{fk}_u, \qquad k \in C, f, g \in F^k : f \prec_k g, u \in N \tag{2.8}$$

$$\sum_{u \in N} y^{fk}_u = 1, \qquad k \in C, f \in F^k \tag{2.9}$$

$$y^{fk}_u \leq \sum_{(v,u) \in A} t^k_{vu}, \qquad k \in C, \quad f \in F^k, \quad u \in N \tag{2.10}$$

$$y^{fk}_u \leq x^{fk}_u, \qquad k \in C, f \in F^k, u \in N \tag{2.11}$$

$$x^{fk}_u = \begin{cases} 1, & \text{if } u = d_k \\ 0, & \text{if } u = s_k \end{cases} \qquad k \in C, \quad f \in F^k \tag{2.12}$$

$$y^{fk}_u, x^{fk}_u \in \{0,1\} \qquad u \in N, \quad f \in F, \quad k \in C \tag{2.13}$$

$$w_u \in \{0,1\} \qquad u \in N \tag{2.14}$$

$$t^k_{uv} \in \{0,1\} \qquad (u,v) \in A, \quad k \in C \tag{2.15}$$

$$z^f_u \in \mathbb{N} \qquad u \in N, \quad f \in F \tag{2.16}$$

The objective function (2.1) aims to minimize the sum of the VNF installation costs and the node activation costs. Constraints (2.2) are the standard flow conservation constraints which ensure that one unit of flow is routed from s_k to d_k for each commodity $k \in C$. Constraints (2.3) are the latency constraints: the sum of the arc latency values along the routing path must be less or equal to l_k, for each commodity $k \in C$. Constraints (2.4) represent the node capacity constraints; they guarantee that the number of VNFs installed at each node $u \in N$ is bounded enough by its capacity c_u. Constraints (2.5) are the VNF capacity constraints; they ensure that the volume of data treated by each function $f \in F$ does not exceed its capacity m_f. Constraints (2.6) are the conflict constraints and they guarantee that two VNFs in conflict are not installed at the same node $u \in N$. Constraints (2.7) are needed to link, the node-installation variables (y), the precedence variables (x) and the arc variables (t): the left-hand-side is forced to 1 (implying that the function f is installed at the node v) if and only if (i) the arc (u,v) is taken in the path associated with the considered commodity k and (ii) the function f is installed at or before the node v and it is not installed at or before the node u. Constraints (2.8) impose the VNFs order for each commodity. Constraints (2.9) guarantee that all required functions for commodity $k \in C$ are installed at the graph nodes. Constraints (2.10) ensure that if a VNF $f \in F^k$ is installed at a node u for a given commodity k, then the associated routing path must enter that node. Inequalities (2.11) link the precedence and the installation variables, x and y, and express the fact that if VNF f is installed at node u for the commodity k, then f is installed at or before the node u. Finally, constraints (2.12) guarantee that, for each

commodity $k \in C$, no VNF is installed at or before the source node s_k and all VNFs are installed at or before the destination node d_k. Constraints (2.13)-(2.16) are the integrality constraints.

Theorem 2.2 *The model (P) is valid. In particular, there always exists an optimal solution of (P) such that for each $k \in C$, the flow variables t^k (representing the routing of commodity k) correspond to an elementary path in G.*

Proof. Let $k \in C$ be a given commodity. Flow constraints (2.2) guarantee that s_k and d_k are connected. Furthermore, latency limit l_k cannot be violated due to the constraint (2.3). It only remains to prove that there exists an optimal solution of the compact model whose routing path for k is elementary, which will imply that precedence constraints (2.8) are satisfied (due to constraints (2.7)).

Let $(\hat{x}, \hat{y}, \hat{w}, \hat{t}, \hat{z})$ be an optimal solution of the compact model, and let G_k be the subgraph of G induced by the arcs $(u, v) \in A$ such that $\hat{t}^k_{uv} = 1$. Assume that G_k is not an elementary path connecting s_k to d_k. We distinguish the following two cases:

- Case 1: subgraph G_k is connected, but it contains cycles (see Figure 2.3). Assume that the routing path is composed by an elementary path and (without loss of generality) a single cycle. The nodes in the elementary path are denoted by $u_i, \forall i = 1, \ldots, q+1$ (where $u_{q+1} = d_k$) and all the other nodes are denoted by $v_i, \forall i = 1, \ldots, p$.

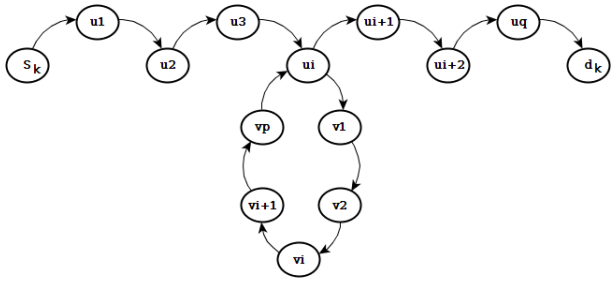


Figure 2.3: Connected subgraph.

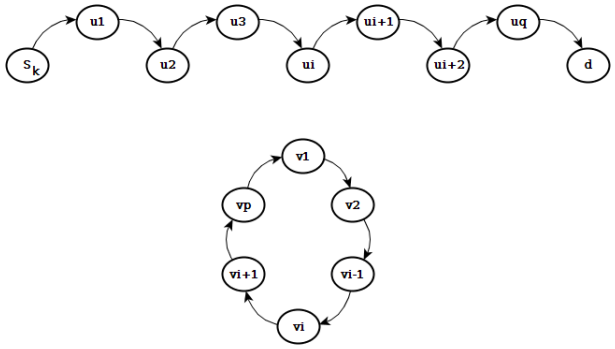


Figure 2.4: Disconnected subgraph.

We will show that all VNFs $f \in F^k$ must be installed along the elementary path, so that the nodes v_i, $i = 1, \ldots, p$ can be removed from the routing path, without violating feasibility and optimality of the solution. Indeed, in that case, removing the cycle from G_k only reduces the latency of the routing path, the flow balance constraints remain valid and the VNFs are properly placed at the remaining nodes.
Let f be an arbitrary VNF from F^k and assume the contrary, i.e., that f is installed at a node v and it is not installed at any node u. Therefore:

$$\hat{y}_{u_i}^{fk} = 0, \quad \forall i = 1, \ldots, q+1.$$

Using the precedence constraints (2.7) $\hat{t}_{uv} - 1 + \hat{x}_v^{fk} - \hat{x}_u^{fk} \leq \hat{y}_v^{fk}$ we can now calculate the values of $\hat{x}_{u_i}^{fk}$, for all $i = 1, \ldots, q+1$.
Let us start with the arc (s_k, u_1), the constraint (2.7) reads as follows:

$$\hat{t}_{s_k u_1}^k - 1 + \hat{x}_{u_1}^{fk} - \hat{x}_{s_k}^{fk} \quad \leq \quad \hat{y}_{u_1}^{fk}$$

Recall that VNFs cannot be installed at the source node, i.e., $\hat{x}_{s_k}^{fk} = 0$ (cf. constraint (2.12)), and that the arc (s_k, u_1) is taken in the path so $\hat{t}_{s_k u_1}^k = 1$. After replacing the variables by the values we obtain:

$$1 - 1 + \hat{x}_{u_1}^{fk} - 0 \quad \leq \quad 0 \qquad \Rightarrow \qquad \hat{x}_{u_1}^{fk} \quad \leq \quad 0.$$

So, it follows that $\hat{x}_{u_1}^{fk} = 0$. By repeating these steps for all subsequent arcs (u_i, u_{i+1}), $i = 1, \ldots, q$, we obtain:

$$\hat{x}_{u_i}^{fk} = 0, \quad \forall i = 2, \ldots, q+1.$$

This contradicts the constraint (2.12), which states that $\hat{x}_{d_k}^{fk} = 1$.

- Case 2: Subgraph G_k is disconnected (see Figure 2.4). In that case, following similar arguments as for the Case 1, we conclude that VNFs from F^k can be installed only along the elementary path in G_k connecting s_k to d_k, so that without loss of feasibility and optimality, the remaining nodes from G_k can be removed (and the values of $\hat{t}$ can be appropriately redefined).

According to this proof, we conclude that we can always find an optimal solution of the compact model which is composed by elementary paths only. ■

2.6.3 Model analysis

In this subsection we analyze the proposed compact formulation in terms of the number of variables and constraints generated by the MILP model.

Number of variables

The total number of variables is:

$$|C| \times |A| + 2 \times |N| \times |C| \times |F^k| + |N| + |N| \times |F| \tag{2.17}$$

$$\iff$$

$$O(|C| \times |A|) + O(|N| \times |C| \times |F^k|) + O(|N| \times |F|)$$

which is equivalent to $O(|C| \times (|A| + |N| \times |F|)$

Number of constraints

The total number of constraints is:

$$\begin{aligned} &|C| \times |N| + |C| + |N| + |N| \times |F| \\ &+|N| \times |C| \times |S^k| + 3 \times |C| \times |F^k| \\ &+|C| \times |A| \times |F^k| + |N| \times |C| \times |F^k|^2 \\ &+3 \times |N| \times |C| \times |F^k| \end{aligned} \tag{2.18}$$

which is equivalent to $O(|C| \times |F| \times (|A| + |N| \times |F|))$

2.7 Computational results

In this section, we test the capability and the performance of the compact MILP formulation on a set of realistic instances derived from the SND library in terms of CPU time, and final gaps. The instances creation is detailed in the next chapter. All the experiments described in this section were made using a computer with Intel(R)Xeon(R) CPU E5-2650 v2 processor clocked at 2.60GHz, 32 cores, two threads per core and 252GB RAM, under Linux operating system. The MILP compact formulation is implemented using the Python API for CPLEX, which is run in single-thread mode and a default memory limited to 20GB. All CPLEX parameters were set to their default values. A default time limit of one hour is set for each tested instance.

In the following, we denote by `C`, The compact MILP formulation.

Figure 2.5 shows that the number of variables increases with the number of commodity, which this is caused by the variables indexed by k (x, y and t variables). The number of constraints increases accordingly with the density of the graph: for example, instances *Pdh* having 24 commodities and 68 bidirected arcs, the number of generated variables is small compared to the number of variables. We can also observe that the number of constraints generated by the compact formulation is in general very large compared to the number of variables.

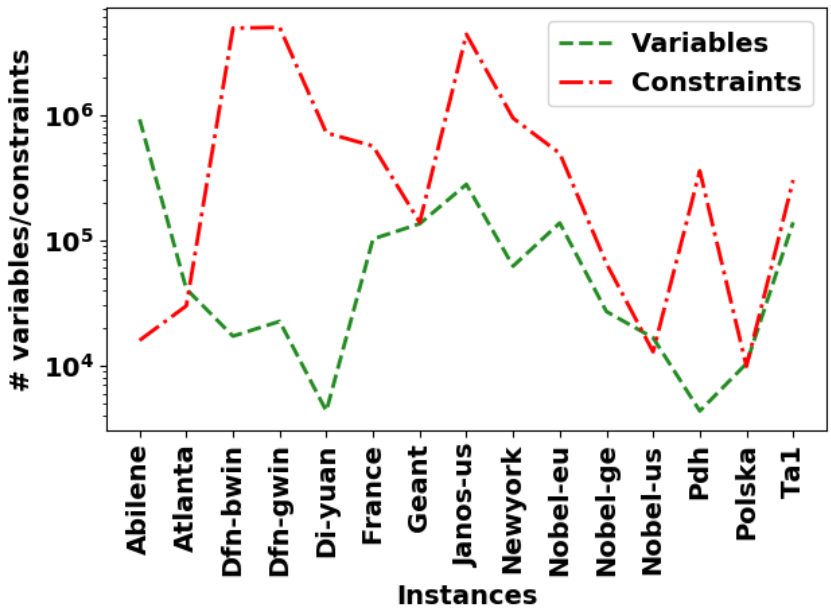


Figure 2.5: Number of variables and constraints generated by the compact formulation.

Table 2.4 summarizes the more detailed results which are given in Tables 2.6-2.9 (see Subsection 2.7.1). Tables show the obtained results by solving 15 SNDlib instance types with different graph topologies and different number of commodities, namely, "Abilene", "Atlanta", "Dfn-bwin", "Dfn-gwin", "Di-yuan", "France", "Geant", "Janos-us", "Newyork", "Nobel-eu", "Nobel-germany", "Nobel-us", "Pdh", "Polska" and "Ta1". For each instance type, ten instances are generated randomly. From the tables, we observe that only two "Abilene" instances and two "Pdh" instances are solved to optimality. Also, we remark that no feasible solution is provided by the model for four "Nobel-eu" instances and for one instance "Ta1".

Table 2.4 represents the average value for the CPU time, final Gap, cost, and relaxation value of the ten instances associated with each instance type. A value in the column CPU_time(s) represents the average of the CPU time of the instances solved to optimality without exceeding the time limit, whereas TL in this column illustrates the fact that the compact formulation needs more than 1 hour to find an optimal solution for all instances. From the table, we observe that for some instance types, finding a good feasible solution is easier than for others, for example, for instance types "Abilene", "Dfn-bwin", "Dfn-gwin", "Janos-us" and "Pdh" the final gaps are below 10%, while, for "Atlanta", "France", "Geant", "Newyork", "Nobel-eu", "Nobel-germany", "Nobel-us", "Polska" and "Ta1" the final gaps are between 20% and 74%.

Instance_name	CPU_time(s)	GAP(%)	Costs($)	Relaxation value
Abilene	1688.92	4.85	79990.30	58929.67
Atlanta	TL	58.37	278836.30	92640.77
Dfn-bwin	TL	1.58	70656.50	48731.37
Dfn-gwin	TL	8.79	131783.70	100959.17
Di-yuan	TL	20.49	35104.00	25435.27
France	TL	67.97	651469.10	187780.30
Geant	TL	45.80	451038.30	109636.63
Janos-us	TL	4.10	13775446.70	13204901.84
Newyork	TL	50.11	342506.20	141291.17
Nobel-eu	TL	74.45	600379.83	139001.24
Nobel-germany	TL	45.11	124062.90	58908.65
Nobel-us	TL	38.93	118642.50	63906.55
Pdh	1587.34	1.20	54561.60	38874.36
Polska	TL	31.30	137285.30	88343.00
Ta1	TL	58.86	376160.33	78467.77

Table 2.4: Average of CPU time, Gap, Cost and relaxation value.

2.7.1 Detailed results

In this subsection, we report detailed results obtained by solving the SNDlib instances using the compact MILP formulation.

Abbreviations	Description
CPU_C	CPU time in seconds.
RC	LP-relaxation value.
LB_C	Best known lower bound, provided by Cplex.
Gap_C	Relative gap, provided by Cplex.
UB_C	Best known global upper bound, provided by cplex.

Table 2.5: Description of abbreviations used on Tables 2.6-2.9.

The final gap is calculated as GAP_C $= (UB - LB)/LB * 100\%$, where UB denotes the best feasible solution, and LB the global lower bound found in each run.

Instance	CPU_C	RC	LB_C	Gap_C	UB_C
Abilene	*TL*	96349.60	121732	11.05	108280.35
	TL	70104.38	102227	17.51	84325.84
	TL	50815.42	68999	1.29	68106.14
	TL	48503.40	68227	0.34	67997.68
	1533.09	40452.10	55712	0.00	55712.00
	TL	57305.83	74926	2.58	72989.17
	TL	89779.21	111873	7.24	103773.29
	TL	50181.90	67942	4.64	64787.33
	1844.75	40260.73	58900	0.00	58900.00
	TL	45544.12	69365	3.86	66684.41
Atlanta	*TL*	187685.36	455555	55.87	201047.36
	TL	65595.14	171546	53.71	79415.00
	TL	75171.64	270340	67.91	86748.61
	TL	65463.21	140690	42.68	80637.43
	TL	79399.97	244878	61.30	94777.48
	TL	60813.42	152840	53.12	71651.54
	TL	98220.61	473321	76.86	109548.74
	TL	84582.53	361504	73.80	94728.71
	TL	64238.76	128957	39.33	78239.26
	TL	145237.07	388732	59.10	158977.90
Dfn-bwin	*TL*	48543.97	72194	1.71	70960.00
	TL	43190.17	66269	2.36	64703.00
	TL	51086.81	73695	1.26	72765.00
	TL	42190.82	62340	2.76	60618.34
	TL	59611.15	82931	2.45	80896.00
	TL	42820.20	65148	0.60	64758.00
	TL	43048.82	63340	2.22	61934.45
	TL	60745.40	84250	0.41	83908.00
	TL	52554.58	73317	0.50	72951.62
	TL	43521.76	63081	1.50	62137.64
Dfn-gwin	*TL*	93821.72	129047	11.44	114280.35
	TL	118367.32	151463	11.40	134192.26
	TL	77911.29	113279	11.66	100072.27
	TL	72339.77	105326	11.79	92910.44
	TL	105110.36	132226	6.90	123096.30
	TL	101920.72	130431	6.19	122360.65
	TL	105783.60	133469	6.79	124406.07
	TL	123237.29	152082	4.48	145263.03
	TL	86216.46	116757	10.54	104454.14
	TL	124883.21	153757	6.69	143469.36

Table 2.6: Obtained results for the compact MILP formulation.

Instance	CPU_C	RC	LB_C	Gap_C	UB_C
Di-yuan	TL	23367.81	30114	17.55	24829.23
	TL	32295.12	43227	22.46	33518.40
	TL	24334.12	33484	19.14	27076.03
	TL	23803.91	31854	18.16	26069.27
	TL	22808.12	34687	29.56	24431.85
	TL	25128.40	37195	25.33	27773.67
	TL	29156.42	40179	16.12	33700.80
	TL	25619.04	39170	27.21	28510.55
	TL	25561.37	32548	13.88	28031.58
	TL	22278.40	28582	15.51	24148.23
France	TL	177774.80	673370	69.42	205891.57
	TL	129220.07	646723	77.15	147748.06
	TL	174563.06	704671	72.75	191988.23
	TL	181826.41	637928	68.07	203678.70
	TL	139979.59	690638	76.90	159531.84
	TL	235734.13	743322	66.02	252556.79
	TL	224804.23	693888	64.66	245227.88
	TL	230949.85	609195	59.66	245733.54
	TL	118290.50	463475	68.98	143772.28
	TL	264660.39	651481	56.08	286114.53
Geant	TL	77238.42	147568	26.64	108253.33
	TL	90112.74	167340	25.19	125190.60
	TL	188864.38	1012275	78.06	222085.94
	TL	86437.31	157715	23.96	119922.21
	TL	111260.20	191529	25.49	142712.27
	TL	123382.34	1090981	85.85	154365.54
	TL	100105.48	676904	81.02	128477.31
	TL	147328.25	778519	76.54	182635.56
	TL	78866.47	142442	22.11	110949.97
	TL	92770.69	145110	13.17	125993.68
Janos-us	TL	10887359.23	11382718	4.28	10895750.90
	TL	11342337.18	13070200	13.13	11353640.23
	TL	14005192.67	15379304	8.88	14014082.08
	TL	11873470.30	13822127	14.00	11887417.16
	TL	15534868.92	15570677	0.16	15545076.64
	TL	14555386.83	14580575	0.10	14565999.76
	TL	12625202.51	12648938	0.10	12636608.73
	TL	12848111.78	12870288	0.09	12858854.98
	TL	12537209.83	12570924	0.16	12550414.28
	TL	15839879.13	15858716	0.06	15849306.31

Table 2.7: Obtained results for the compact MILP formulation.

Instance	CPU_C	RC	LB_C	Gap_C	UB_C
Newyork	*TL*	149337.54	466578	64.35	166338.93
	TL	211270.08	460553	49.65	231888.21
	TL	87961.14	359078	70.03	107602.64
	TL	155144.28	456137	59.71	183782.67
	TL	131163.4	233563	36.01	149449.93
	TL	90660.41	184876	41.27	108584.56
	TL	199250.03	417816	48.54	215000.02
	TL	149434.14	391109	58.14	163720.47
	TL	91031.13	184684	38.11	114305.83
	TL	147659.6	270668	35.3	175110.97
Nobel-eu	*TL*	108330.18	–	–	–
	TL	131278.09	–	–	–
	TL	257646.79	–	–	–
	TL	266694.08	–	–	–
	TL	88728.3	411905	74.4	105456.84
	TL	130476.89	639078	77.27	145238.05
	TL	104054.56	628717	80.6	121959.23
	TL	94211.05	475501	76.38	112290.76
	TL	265906.49	753040	62.29	283984.67
	TL	150630.18	694038	75.78	168085.82
Nobel-ger	*TL*	55711.09	104103	37.05	65532.57
	TL	44959.19	86285	34.8	56257.18
	TL	69743.46	134963	40.8	79896.42
	TL	62029.7	130569	44.75	72136.98
	TL	74763.61	150460	46.96	79805.52
	TL	66893.27	150787	51.24	73518.01
	TL	46931.61	119016	53.42	55433.86
	TL	51239.35	112900	47.72	59023.63
	TL	72577.69	138111	41.28	81097.6
	TL	44237.53	113435	53.1	53199.58
Nobel-us	*TL*	51971.67	79287	23.87	60359.93
	TL	44398.81	97496	46.5	52164.02
	TL	103875.95	155983	26.39	114818.67
	TL	68501.25	129906	41.59	75877.57
	TL	117886.87	193246	33.18	129125.75
	TL	46824.9	101846	40.4	60697.28
	TL	47960.09	97320	41.07	57346.84
	TL	62032.18	125415	44.64	69431.48
	TL	43797.41	101285	50.18	50457.68
	TL	51816.35	104641	41.49	61224.9

Table 2.8: Obtained results for the compact MILP formulation.

Instance	CPU_C	RC	LB_C	Gap_C	UB_C
Pdh	2141.91	38805.94	56518	0	56518
	TL	45729.75	66862	0.49	66537.25
	TL	38932.42	51989	1.05	51442.24
	TL	36016.09	49517	1.5	48772.25
	TL	30850.73	44116	3.33	42647.65
	TL	32338.46	47046	1.6	46291.68
	1032.78	39393.92	55876	0.0	55876
	TL	37821.69	50606	1.82	49684.82
	TL	38734.6	57071	1.64	56135.3
	TL	50119.97	66015	0.54	65660
Polska	*TL*	82914.01	137978	37.29	86522.85
	TL	106159.63	142278	21.88	111142.83
	TL	85717.67	132934	31.35	91255.38
	TL	85426.18	146856	35.6	94570.84
	TL	81055.23	130412	32.72	87744.08
	TL	94797.85	143218	28.64	102198.21
	TL	95199.63	135530	26.05	100218.98
	TL	91039.84	148651	34.62	97190.71
	TL	92574.06	132398	26.11	97832.97
	TL	68545.87	122598	38.7	75147.37
Ta1	*TL*	82715.84	–	–	–
	TL	73074.28	148201	38.87	90592.97
	TL	63090.38	126016	29.69	88602.56
	TL	65060.97	207975	57.13	89154.76
	TL	71865.36	294566	66.63	98300.08
	TL	87691.33	306170	63.5	111764.07
	TL	100768.98	925598	85.98	129798.82
	TL	96286.92	960568	87.46	120413.5
	TL	80692.01	250241	56.22	109551.25
	TL	67679.75	166108	44.24	92618.86

Table 2.9: Obtained results for the compact MILP formulation.

2.8 Conclusions

In this chapter, we have presented the Virtual Network Functions Placement and Routing Problem, its properties, and an compact MILP formulation to model it. We have tested the proposed compact MILP formulation on a set of realistic instances derived from the SND library. The results showed that the model could find an optimal solution within 1 hour for some easy instances. However, it is no strong enough to find a good feasible solution for harder instances.

Chapter 3

MILP-based Heuristic

In this chapter, we propose a path-based MILP formulation to model the VNFPR problem. We also demonstrate how to efficiently use it to derive high-quality heuristic solutions in a reasonable computational time. The study is conducted on a set of realistic telecommunication instances derived from the SND library. To test the efficiency of our approach, we also compare the obtained results with the compact MILP formulation introduced in Chapter 2. We vary the problem's parameters such as the node and VNF capacities, the commodities bandwidth and latency, and we discuss the trade-offs between saved costs and (in)feasibility. The purpose of this chapter is to provide an empirical study and evaluate the viability of the MIP-based heuristic, as an alternative to the compact MILP formulation presented on Chapter 2.

Contents

In Chapter 2, we have proposed an MILP compact formulation for the Virtual Network Functions Placements and Routing Problem. Computational results showed that the model could provide feasible solutions for some easy instances. Unfortunately, the model cannot find any feasible solution or provides huge final gaps for some other instances. This is due to the large number of variables and constraints in the formulation, making its computational performance fairly limited.

To tackle this, some papers divide the problem into two parts and treat the VNFs placement and the routing problems separately. A large body of the literature deals with heuristics, in order to handle each part of the problem, or for their relaxed versions. In this chapter, we propose a computationally effective path-based MILP formulation to model the Virtual Network Functions Placement and Routing Problem. The Path-based Formulation (PF) is established using Yen's algorithm, which aims to find a fixed number of elementary shortest paths between two nodes. Based on that, the latency-constrained routing paths associated with each commodity are obtained.

In the case where all feasible paths associated with each commodity are taken into account in the model, the path formulation represents an exact method; thus, it provides optimal solutions. Alternatively, if only a subset of feasible paths is included in the model, the path-based formulation provides heuristic solutions. We demonstrate the path-based formulation's effectiveness by comparing it with the MILP compact formulation presented in Chapter 2 on a set of realistic benchmark instances derived from the SNDLib. In addition, we vary the problem's parameters, such as node and VNF capacities and commodities latency and bandwidth, and analyze the computational behavior and the cost saving achieved by enlarging the capacities.

Outline of the chapter. Section 3.1 is devoted to the path formulation. In Section 3.2, we present computational results and a sensitivity analysis in which we vary the input parameters and discuss the results. In Section 3.3 we provide some concluding remarks.

3.1 Path-based MILP Formulation

In this section, we present the path-based MILP formulation to model the VNFPRP. We first describe the set of decision variables, and then the constraints defining the model.

3.1.1 Decision variables

The path formulation is characterized by five families of variables described in Table 3.1. We keep the variables x, y, z and w already defined for the compact MILP formulation in Chapter 2 and we define a new family of variables representing the routing variables. For each commodity $k \in C$, let $\mathcal{P}_k$ denote the set of all shortest s_k-d_k-paths whose total length does not exceed l_k, where the arc latency is used as the length measure. For each commodity k, we suppose that all feasible paths in $\mathcal{P}_k$ are given. Let t_{uv}^{pk} be the parameter that is equal to 1 if arc (u, v) belongs to path p for commodity k; and equal to 0 otherwise.

Variables		Type
λ_p^k	1, if path $p \in \mathcal{P}_k$ associated with commodity k is chosen; 0, otherwise.	Binary
x_u^{fk}	1, if the virtual network function f is installed at or before node u for commodity k; 0, otherwise.	Binary
y_u^{fk}	1, if virtual network function f is installed at node u for commodity k; 0, otherwise.	Binary
w_u	1, if node u is activated; 0, otherwise.	Binary
z_u^f	number of VNF f installed at node u.	Integer

Table 3.1: Decision variables of the path-based formulation

3.1.2 Linear constraints

The constraints containing x, y, z and w variables are the same for the path-based MILP formulation as in the compact formulation. As we replace the arc variables t in the compact formulation, by the routing variables λ in the path formulation, the flow (2.2) constraints, the precedence constraints (2.7), and the linking constraints (2.10), will be replaced by the following constraints:

Routing constraints

$$\sum_{p \in \mathcal{P}_k} \lambda_p^k = 1, \qquad k \in C \tag{3.1}$$

For each commodity k, exactly one $s_k - d_k$ elementary routing path $p \in \mathcal{P}_k$ satisfying the latency constraints is chosen.

Precedence constraints

$$\begin{aligned}(\sum_{\substack{p\in\mathcal{P}_k\\(u,v)\in p}} t_{uv}^{pk}\ \ \lambda_p^k - 1) + (x_v^{fk} - x_u^{fk}) \quad &\leq y_v^{fk}, \\ k \in C, \quad f \in F^k, \quad (u,v) \in A \end{aligned} \tag{3.2}$$

Inequalities (3.2) represent the linking constraints between routing variables (λ), installation variables (y) and precedence variables (x), ensuring that for each commodity k: *(i)* if the routing path p passes through the arc (u,v), and *(ii)* the VNF f is installed at or before the node v and *(iii)* the VNF f is not installed at or before the node u, then the left-hand-side is forced to be equal to 1 (imposing the installation of the VNF f at the node v).

Linking constraints

$$y_u^{fk} \quad \leq \quad \sum_{(v,u)\in A}\sum_{p\in\mathcal{P}_k} t_{vu}^{pk}\ \ \lambda_p^k, \quad k \in C, \quad f \in F^k, \quad u \in N \tag{3.3}$$

Constraints (3.3) impose that the chosen path for each commodity k passes through the nodes where the required functions are installed.

3.1.3 Mathematical model

The Path-based formulation (PF) then reads as follows:

$$\text{(PF): min} \quad \sum_{u\in N}\sum_{f\in F} \psi_u^f\ \ z_u^f + \sum_{u\in N} \psi_u\ \ w_u$$

$$(\lambda, x, y, z, w) \qquad \text{satisfy} \quad \text{(3.1) - (3.3), (2.4)-(2.6), (2.8)-(2.9), (2.11)-(2.12)}$$

$$\begin{aligned} \lambda_p^k &\in \{0,1\} && k \in C, \quad p \in \mathcal{P}_k \\ x_u^{fk}, y_u^{fk} &\in \{0,1\} && k \in C, \quad u \in N, \quad f \in F \\ w_u &\in \{0,1\} && u \in N \\ z_u^f &\in \mathbb{N} && u \in N, \quad f \in F \end{aligned}$$

This model contains a possibly exponential number of variables. We therefore consider a heuristic approach in which only a subset of the most promising feasible paths is selected per each commodity.

Theorem 3.1 *If Yen's algorithm generates all elementary latency-constrained paths for each commodity $k \in C$, then the path-based formulation gives optimal solutions; otherwise it provides a heuristic solution for VNFPRP.*

3.1.4 Getting the routing paths

In order to generate routing paths associated with each commodity k, we use Yen's algorithm [138]. This algorithm aims to find a limited number of loopless shortest paths between a pair of nodes. Yen's algorithm is based on Dijkstra's algorithm [44]. For reasonably small instances, instead of recovering a fixed number of paths per commodity, we run Yen's algorithm to get all $s_k - d_k$ paths.

In order to consider the latency constraints 2.3 in the path formulation model using Yen's algorithm, we keep only paths whose length is less or equal to l_k, $k \in C$.

Remark: We used NetworkX graph library of Python to get all the paths associated with each commodity. The used Python function is called "Shortest_simple_paths", which returns a list of all simple paths between a source node and a destination node, ordered from the shortest to the longest one.

To fix the number of generated paths to κ we use the following Python command:

$$list(islice(nx.shortest\_simple\_paths(G, source, destination, weight = weight), \kappa))$$

We adapted the Python function "Shortest_simple_paths" to get only the latency constrained paths. In what follows in this manuscript, when we say: we get all the paths using Yen algorithm and we keep only the latency constrained one, we mean that we use this adapted Python function "Shortest_simple_paths".

By definition of the sets $\mathcal{P}_k$, only elementary paths are considered in this model, and hence, the formulation is correct. Unfortunately, the number of elementary paths can be exponential, which makes this MILP formulation intractable, unless it is embedded within a Branch-and-Price procedure. Nevertheless, this formulation can be efficiently used as an MILP heuristic, considering only a subset (polynomial in size) of elementary paths in the model.

3.1.5 Linear relaxation of path variables λ

Constraints $\lambda_p^k \in \{0,1\}$ are the integrality constraints guaranteeing that the latency-constrained path cannot be split. Together with constraints (3.1), they ensure that there is exactly one path used to route the flow for each commodity. In the following, we show that $\lambda_p^k \in \{0,1\}$ can be relaxed. Let PF' denote the model PF for which the integrality constraints associated with variables λ are replaced by: $\lambda_p^k \geq 0, \quad k \in C, p \in \mathcal{P}_k$.

Proposition 3.2 *If the relaxed path formulation PF' has an optimal solution with fractional λ values, then it must necessarily admit an integer solution with the same objective value.*

Proof. Let $\hat{X} = (\hat{\lambda}, \hat{x}, \hat{y}, \hat{z}, \hat{w})$ denotes the optimal solution of (PF') with $\hat{\lambda}$ fractional, i.e., there exists at least one commodity $k \in C$ for which the path variable $\hat{\lambda}_p^k$ is fractional. The proof aims to show that from the optimal fractional solution, $\hat{X}$ a totally integer solution can be constructed. Let $\bar{X} = (\bar{\lambda}, \hat{x}, \hat{y}, \hat{z}, \hat{w})$ be the integer solution which is also feasible for (PF').

Variables $\hat{y}$ and $\hat{w}$ appear in the objective function with strictly positive cost and are the same for both solutions $\hat{X}$ and $\bar{X}$; thus, the two solutions will have the same objective value.

Let $k \in C$ be the fractional commodity for which the associated vector $\hat{\lambda}$ contains fractional values in $\hat{X}$. Let $\mathcal{P}'_k = \{p_1, p_2, \ldots, p_Q\}$ be the set of fractional paths associated with commodity k, satisfying the path constraints (3.1). In that case, by setting an arc upper bounds corresponding to constraints (3.2) and node capacity lower bounds corresponding to constraints (3.3) as follows:

$$\hat{cap}_{uv} := \min_{f \in F^k} \{\hat{y}_v^{fk} + 1 - \hat{x}_v^{fk} + \hat{x}_u^{fk}\} \qquad (u,v) \in A \tag{3.4}$$

$$\hat{cap}_u := \max_{f \in F^k} \{\hat{y}_u^{fk}\} \qquad u \in N \tag{3.5}$$

we obtain the following linear inequality system:

$$0 \leq \sum_{p \in \mathcal{P}_k} \lambda_p^k \; t_{vu}^{pk} \quad \leq \quad \hat{cap}_{uv} \qquad (u,v) \in A \tag{3.6}$$

$$\sum_{(v,u) \in A} \sum_{p \in \mathcal{P}_k} \lambda_p^k \; t_{vu}^{pk} \quad \geq \quad \hat{cap}_u \qquad u \in N \tag{3.7}$$

For each node $u \in N$ such that $\hat{cap}_u = 1$, by the feasibility of $\hat{\lambda}$, all paths $p \in \mathcal{P}'$ must pass through the node u. Hence, to satisfy constraints (3.6) and (3.7), one could take any of the paths $p \in \mathcal{P}'_k$ as a feasible binary solution.

From the above, amid the fractional paths constituting $\hat{\lambda}$, there exists at least one (integer) path which can be used to replace $\hat{\lambda}$. The same procedure can be reiterated for each fractional commodity that admits a fractional $\hat{\lambda}^k_p$ in the solution $\hat{X}$, keeping the same structure of the binary component $(\hat{x}, \hat{y}, \hat{z}, \hat{w})$, due to the fact that the routing paths are chosen separately for each commodity. ■

Corollary 3.3 *Without loss of generality, constraints $\lambda^k_p \in \{0, 1\}$, for all $k \in C, p \in \mathcal{P}_k$, can be replaced by $\lambda^k_p \geq 0$ in the Path-based formulation.*

3.2 Computational results

The purpose of this section is to test the efficiency and the sensitivity of the proposed path formulation. We present some computational results to compare the path formulation with the compact MILP formulation proposed in Chapter 2. To compare both models, we focus on the CPU time, the quality of the obtained solutions, and the final gaps between the global lower and the best known upper bound. We vary the problem input parameters and conduct a sensitivity analysis to determine the most relevant parameters that affect the model's empirical performance.

All the experiments described in this section were made using a computer with Intel(R)Xeon(R) CPU E5-2650 v2 processor clocked at 2.60GHz, 32 cores, 2 threads per core and 252GB RAM, under Linux operating system. The path formulation is implemented using the Python API for CPLEX, which is run in single-thread mode and a default memory limited to 20GB. All CPLEX parameters were set to their default values. A default time limit of one hour is set for each tested instance.

3.2.1 Benchmark instances

Our benchmark instances are generated using the SND library [125] (Survivable Network Design), which is a repository of realistic telecommunication network design instances. The number of nodes, arcs, and demands varies for each instance. SNDlib provides the graph topology, along with the node coordinates and a set of demands

with the associated source node, destination node, and a bandwidth. The remaining parameters required for our setting are generated as follows:

To calculate the distance between two nodes u and v in km having only their longitude φ_u, φ_v and latitude ς_u, ς_v, we use the Spherical Law of Cosine [106]. First we use the following formula to calculate the angular distance in radians:

$$S_{uv} = arc\cos(\cos\varsigma_u \cos\varsigma_v + \sin\varsigma_u \sin\varsigma_v \cos d\varphi)$$

with, $d\varphi = \varphi_v - \varphi_u$.

The distance in km is then obtained as:

$$d_{uv} = R \times S_{uv},$$

where R is earth's radius ($R = 6378137km$). The fiber propagation delay per km is roughly equal to $10\mu s/km$, see, e.g., [33, 86]. To define the latency l_{uv} of an arc (u, v), we multiply the distance between u and v by the fiber propagation delay, and we set $l_{uv} = l_{vu}$.

We consider the following set of six Virtual Network Functions, typically employed in service function chaining [75, 121] to construct our VNFs set, namely $F = \{NAT,\ FW,\ TM,\ WOC,\ IDPS,\ VOC\}$. A detailed description of these VNFs is given in Table 3.2. The maximum capacity rate associated with each VNF is given as an integer number in $[a, b]$, where a *(resp. b)* is the minimum *(resp. maximum)* bandwidth of all demands in the same set of demand (i.e. $a = \min_{k \in C} b_k$ and $b = \max_{k \in C} b_k$), the capacity is measured on $Mbits/s$. The functions installation cost $\psi_u^f \in \mathbb{N}$ is chosen randomly as integer in $[50, 1000]$, and is expressed in dollars \$.

id-VNF	VNF name	Capacity	Cost
NAT	Network Address Translator	$[\min_{k \in C} b_k, \max_{k \in C} b_k]$	$[50, 1000]$
FW	Firewall		
TM	Traffic Monitor		
WOC	WAN Optimization Controller		
IDPS	Intrusion Detection Prevention System		
VOC	Video Optimization Controller		

Table 3.2: Virtual Network Functions

The total number of demands varies for each SNDlib instance. For each commodity $k \in C$ the source node s_k, destination node d_k and the bandwidth b_k in $Mbits/s$ are given.

We divide the set of demands in five categories: *Online Gaming*, *Video Streaming*, *Voice over IP*, *Web Services*, and *Other Services*. Each category is characterized by a latency value and a set of chained service functions, as depicted in Table 3.3.

To define the latency value l_k and the set of chained VNFs associated with each commodity k, the demands are first assigned to exactly one category. To do so, we calculate the shortest path SP_k between s_k and d_k, which is equal to the sum of the arc latencies composing it. Based on its length, we randomly assign our demands in one of the five categories, and so we set the value of l_k and the set of chained VNFs. For example, if the length of the shortest path is equal to $l(SP_k) = 40ms$, then we can assign the demand to any of the five categories, and we randomly choose one.

Latency value	Service	SFC
$\leq 60\ ms$	Online Gaming (O-G)	NAT-FW-TM-WOC-IDPS
$\leq 100\ ms$	Video Streaming (V-S)	NAT-FW-TM-VOC-IDPS
	VoIP	NAT-FW-TM-FW-NAT
$\leq 500\ ms$	Web Services (W-S)	NAT-FW-TM-WOC-IDPS
$\leq \infty\ ms$	Other services (O-S)	NAT-FW-TM-WOC-VOC

Table 3.3: Five services and their respective SFC and latency value.

In order to generate feasible instances, the node capacity value is obtained based on the number of demands and the number of VNFs per commodity, which is equal to 5, $\forall k \in C$. Thus, the node capacity is an integer chosen randomly $c_u \in [\frac{|C| \times 5}{|N|}, 2 \times \frac{|C| \times 5}{|N|}]$, $\forall u \in N$. Node activation cost $\psi_u \in \mathbb{N}$ is chosen uniformly at random from $[3000, 5000]$ [65].

For each commodity $k \in C$ there is at most one anti-affinity constraint (AAC) between VNFs. We suppose that we cannot install Firewall and Network Address Translator at the same node. The maximum number of AAC is fixed to five per instance because adding multiple anti-affinity constraints renders some instances infeasible.

Instance_type	$\|N\|$	$\|A\|$ / bi-directed	$\|C\|$	$\|F\|$	# AAC
Abilene	12	15/ 30	132	6	5
Atlanta	15	22/ 44	210	6	4
Dfn-bwin	10	45/ 90	90	6	0
Dfn-gwin	11	47/ 94	110	6	0
Di-yuan	11	42/ 84	22	6	0
France	25	45 / 90	300	6	5
Geant	22	36/ 72	462	6	1
Janos-us	26	84/ 168	650	6	1
Newyork	16	49/ 98	240	6	0
Nobel-eu	28	41/ 82	378	6	0
Nobel-Germany	17	26/ 52	121	6	5
Nobel-us	14	21/ 42	91	6	0
Pdh	11	34/ 68	24	6	0
Polska	12	18/ 36	66	6	1
Ta1	24	55/ 110	396	6	0

Table 3.4: Details about the instances. $|N|$: the number of nodes, $|A|$: the number of arcs, $|C|$: the number of demands, $|F|$: the number of functions, # AAC: the number of anti affinity constraints.

To test the efficiency of the proposed path formulation, we consider fifteen different graphs from SNDlib, each of them corresponding to one "instance-type" in our benchmark. Table 3.4 summarizes the details about the graphs used in this study. In order to vary the demand types per instance, we generated ten instances for each instance type. Table 3.5 summarizes the average demand per service for each instance type.

Instance_type	O-G	V-S	VOIP	W-S	O-S
Abilene	4.2	6.2	7.4	57.6	55.6
Atlanta	1.7	6.1	4.9	98.7	97.6
Dfn-bwin	0.0	22.1	21.4	21.9	23.6
Dfn-gwin	0.0	23.6	23.5	31.0	30.9
Di-yuan	0.6	1.6	1.2	8.1	9.5
France	2.5	11.0	8.8	140.1	136.6
Geant	0.8	20.9	17.5	208.1	213.7
Janos-us	10.5	24.2	22.5	302.9	288.9
Newyork	4.5	12.8	11.3	105.6	104.8
Nobel-eu	0.3	10.8	9.4	168.9	187.6
Nobel-Germany	0.0	6.6	5.7	52.0	55.7
Nobel-us	1.6	5.3	6.2	39.0	37.9
Pdh	0.0	6.3	4.9	5.5	6.3
Polska	0.0	5.0	4.1	27.5	28.4
Ta1	6.5	20.5	20.1	169.2	178.7

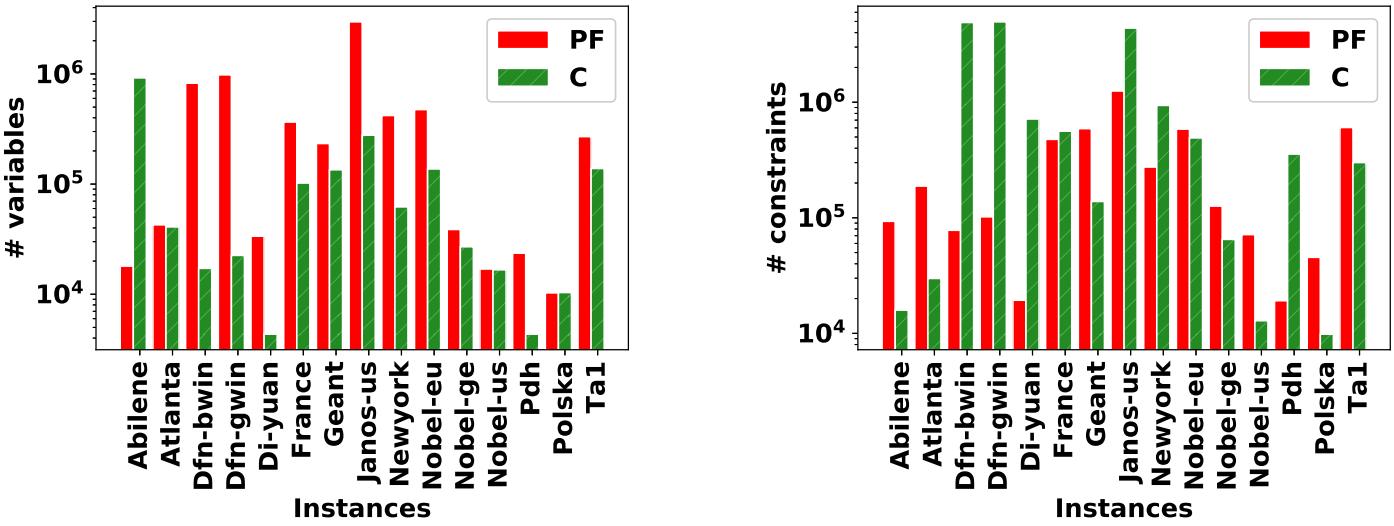
Table 3.5: Average demands per service for each instance type.

3.2.2 Models analysis

In this subsection, we analyze the proposed path-based formulation in terms of the number of variables and constraints. We compare them to the ones of the compact MILP formulation introduced in Chapter 2. In the following, we show only the difference in terms of number of variables and constraints between both models.

Variables: Both formulations have practically the same set of variables, except for the routing part. For the path formulation there are $|C| \times |\mathcal{P}_k|$ path variables, for the compact MILP formulation there are $|C| \times |A|$ arc variables. The set $\mathcal{P}_k$ may contain an exponential number of elements.

Constraints: The number of constraints in the path formulation is smaller than the one in the compact MILP formulation. Accordingly, the compact MILP formulation has $|C| \times |N|$ more constraints.



(a) # Variables

(b) # Constraints

Figure 3.1: Comparison between the path-based MILP formulation and the compact MILP formulation with respect to the number of variables and constraints.

Figure 3.1 shows the average number of variables and constraints per instance-type. We observe that the number of variables in the path formulation is, on average, more significant than the number of variables of the compact MILP formulation. This is due to the number of feasible paths associated with each commodity. We can also observe that the number of constraints in the compact MILP formulation is, on average higher than the one in the path formulation – this number depends on the number of nodes and demands.

3.2.3 Obtained results

The following two settings are compared in our computational study:

- `C` : The compact MILP formulation proposed in Chapter 2.
- `PF` : The path-based formulation proposed in Section 3.1.

Table 3.7 shows the results obtained by solving 10 instances for each instance type. The *CPU* time, *GAP*, and the costs of each instance type are obtained by calculating the average over all ten instances for which a feasible solution was found, otherwise these

instances are not considered. We fixed the maximum number of latency-constrained paths generated by Yen's algorithm to 5000 paths per commodity. In order to define the nature of the path formulation, for each instance type, we pre-calculate the maximum number of latency constrained paths over all commodities. This number is shown in column #paths in Table 3.6. Based on that, the second column denoted by PF(E/H), describes the nature of the path formulation: exact "E" means that the maximum number of feasible paths was below our threshold of 5000 paths per commodity; or heuristic "H" method, meaning that not all feasible paths were taken into consideration by the path formulation, which is the case only for two instance types: "Dfn-bwin" and "Dfn-gwin".

In Table 3.6, we also report the number of instances solved to optimality by the two models. The path formulation solves eight instances to optimality within one hour, whereas C manages to find the optimum for only 4 instances. Moreover, there are five instances for which the compact MILP formulation cannot provide any feasible solution within the given time limit. On the contrary, feasible solutions are found for all instances, using the path formulation.

Table 3.7 summarizes the results obtained by solving 15 SNDlib instance types with different graph topologies and different number of commodities. Columns one and two in the table represent the average of the CPU time calculated from instances solved to optimality by PF and C, respectively. We observe that for instances solved to optimality, PF is outperforming C in terms of CPU time. However, for most of the instances, none of the two methods manages to provide an optimal solution within one hour. Notion "TL" in the CPU time column means that the time limit was reached. When the optimal solution is not found after reaching the time limit, we report the average final gaps, calculated as $GAP(\%) = ((UB - LB)/LB) * 100$, where UB represents the best found feasible solution and the LB represent the global lower bound. The global upper bound is shown in column Costs($), and the value of the LP-relaxation of the two models is given in the last two columns.

Detailed results of this study are shown in Tables 3.9-3.13 in Subsection 3.2.4.

Instance_type	# paths	PF(E/H)	# optimal		no feasible solution found	
			PF	C	PF	C
Abilene	16	E	6	2	0	0
Atlanta	53	E	0	0	0	0
Dfn-bwin	9061	H	0	0	0	0
Dfn-gwin	8801	H	0	0	0	0
Di-yuan	1419	E	0	0	0	0
France	969	E	0	0	0	0
Geant	286	E	0	0	0	0
Janos-us	4316	E	0	0	0	0
Newyork	1588	E	0	0	0	0
Nobel-eu	977	E	0	0	0	4
Nobel-Germany	149	E	0	0	0	0
Nobel-us	45	E	0	0	0	0
Pdh	872	E	2	2	0	0
Polska	35	E	0	0	0	0
Ta1	440	E	0	0	0	1

Table 3.6: Nature of the path formulation (exact or heuristic) and a comparison between both formulations with respect to the number of instances solved to optimality (#optimal) and the number of instances for which no feasible solution is found.

Instance_name	CPU_time(s)		GAP(%)		Costs($)		LP-Relaxation value	
	PF	C	PF	C	PF	C	PF	C
Abilene	**1123.64**	1688.92	**0.61**	4.85	**77158.50**	79990.30	**67547.38**	58929.67
Atlanta	*TL*	*TL*	**27.03**	58.37	**156865.60**	278836.30	**103478.55**	92640.77
Dfn-bwin	*TL*	*TL*	–	**1.58**	70717.00	**70656.50**	**63927.16**	48731.37
Dfn-gwin	*TL*	*TL*	–	8.79	**131284.00**	131783.70	**116131.98**	100959.17
Di-yuan	*TL*	*TL*	**8.81**	20.49	**30759.70**	35104.00	**26154.09**	25435.27
France	*TL*	*TL*	**54.87**	67.97	**497767.20**	651469.10	**213059.66**	187780.30
Geant	*TL*	*TL*	**37.50**	45.80	**284847.80**	451038.30	**127580.37**	109636.63
Janos-us	*TL*	*TL*	**0.04**	4.10	**13221546.20**	13775446.70	13204901.84	13204901.84
Newyork	*TL*	*TL*	**32.10**	50.11	**248954.50**	342506.20	**157088.98**	141291.17
Nobel-eu	*TL*	*TL*	**55.86**	74.45	**430106.33**	600379.83	**152708.92**	139001.24
Nobel-germany	*TL*	*TL*	**8.66**	45.11	**77152.50**	124062.90	**65700.13**	58908.65
Nobel-us	*TL*	*TL*	**21.91**	38.93	**97874.20**	118642.50	**70360.64**	63906.55
Pdh	**1279.91**	1587.34	**1.17**	1.20	**54508.10**	54561.60	**50051.47**	38874.36
Polska	*TL*	*TL*	**15.67**	31.30	**116640.40**	137285.30	**90635.47**	88343.00
Ta1	*TL*	*TL*	**41.52**	58.86	**217968.22**	376160.33	**94323.52**	78467.77

Table 3.7: Average CPU time, Gap, Cost and relaxation value comparison between PF and C.

In Table 3.7 no final Gaps are shown for PF for instance types "Dfn-bwin" and "Dfn-gwin" because PF is a heuristic for them, we are not generating all feasible paths for these instances.

For all instances for which PF is an exact method, the final gap provided (CPLEX exit gap) by PF is consistently better than the final gap provided by C (see Figure 3.2).

Each coordinate (x, y) in Figure 3.3 indicates that for y instance types, the average final gap provided was bellow x. From Figure 3.3, we can observe that all instance types solved by PF as an exact method (we exclude instance types "Dfn-bwin" and "Dfn-gwin") are solved within a gap less than 55%, while the same instances type are solved within a final gaps up to 75% by the compact MILP formulation.

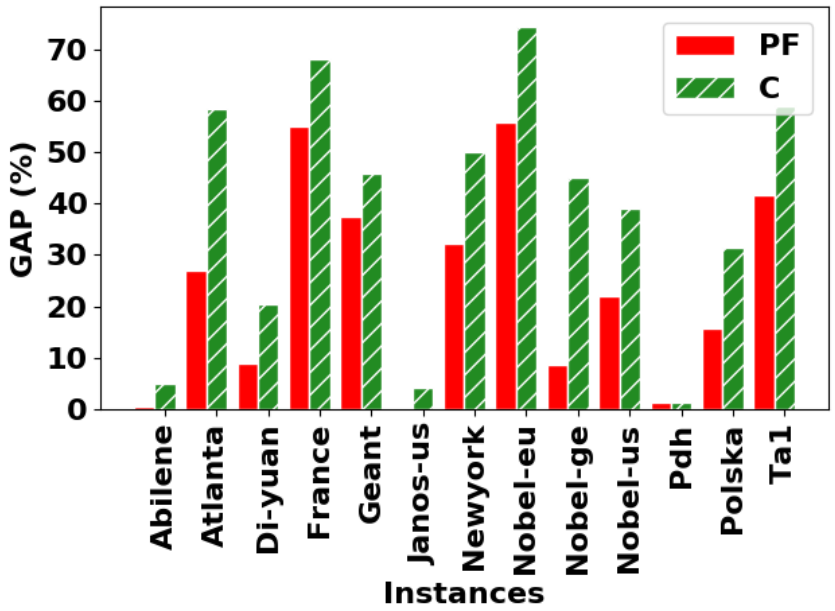


Figure 3.2: GAP comparison between PF and C.

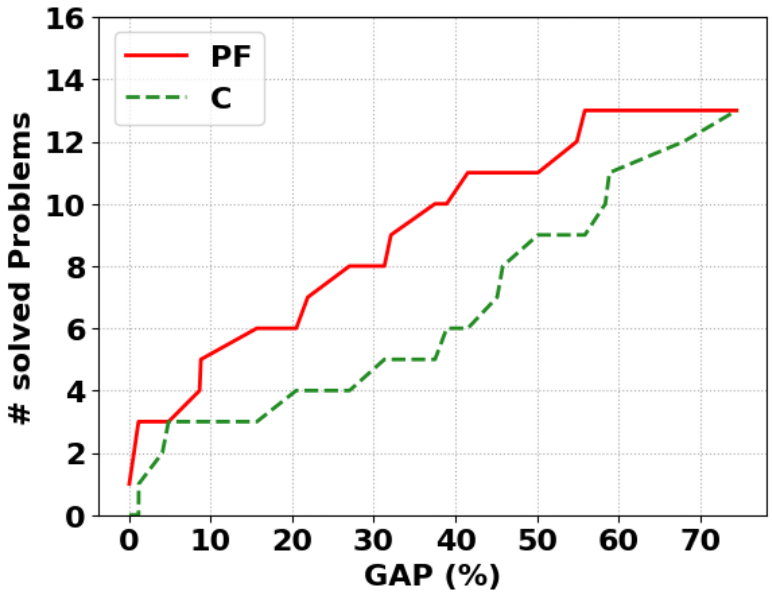


Figure 3.3: GAP comparison between PF and C.

Similarly, from Figure 3.4 we observe that PF consistently provides solutions of a significantly higher quality when compared with solutions given by the compact MILP formulation. The difference between the two methods in terms of costs is very visible.

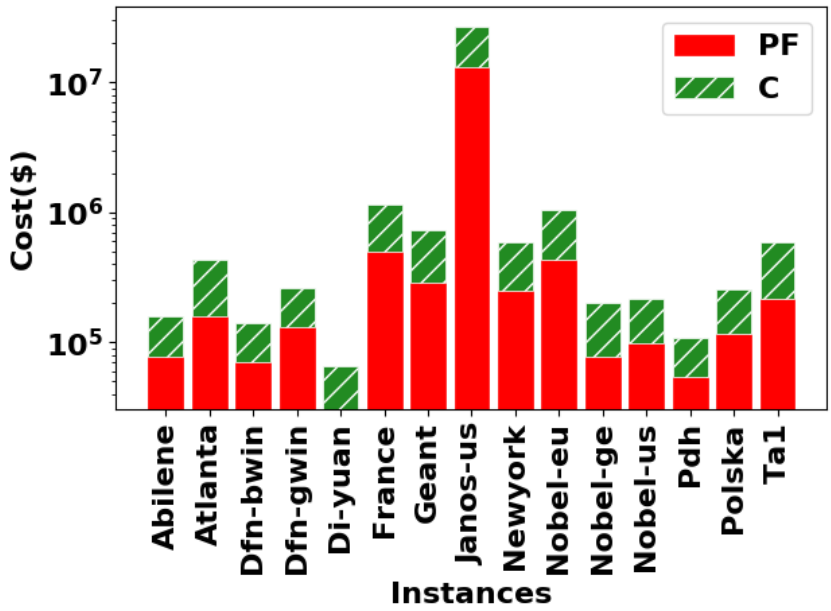


Figure 3.4: Costs comparison between PF and C.

Figure 3.5 shows the relative improvement of the LP-relaxation value provided by PF with respect to C. We notice that the path formulation provides a much better relaxation bound compared with the compact MILP formulation. In Figure 3.5 we observe that for all solved instances the value of the relaxation at the root node provided by the PF formulation is always greater than or equal to the given LP-relaxation of the compact MILP formulation, so closer to the optimal solution.

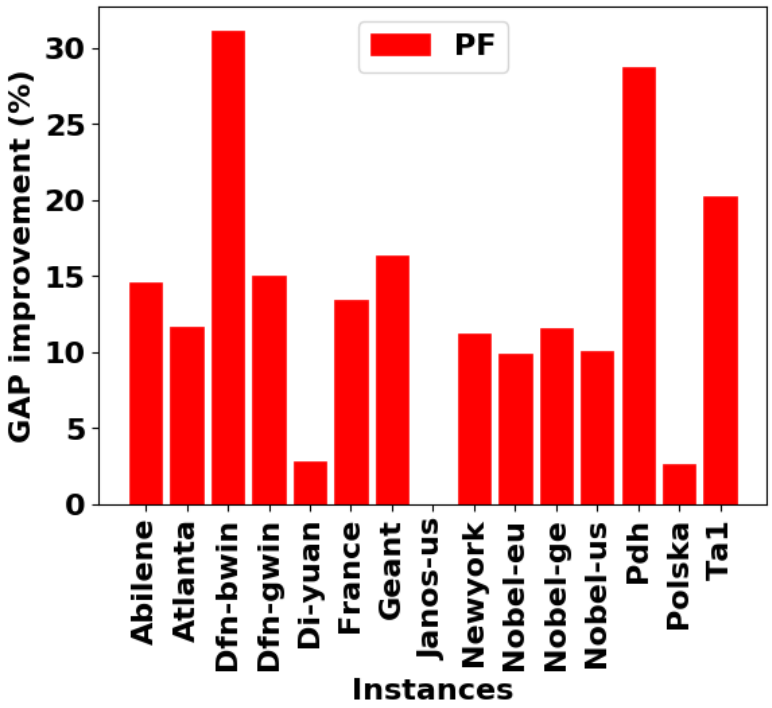


Figure 3.5: LP-relaxation improvement by the path formulation.

3.2.3.1 Sensitivity analysis

For the sensitivity analysis we vary the instances parameters such as: latency value l_k, bandwidth b_k, node capacities c_u and functions capacities m_f and observe how the algorithm will behave, by putting $l_k = \alpha \times l_k$, $b_k = \alpha \times b_k$, $c_u = \alpha \times c_u$ and $m_f = \alpha \times m_f$ with $\alpha \in \{1.0, 1.5\}$. The aim of these tests is to show the parameters that affect directly the solution, and so the costs.

Figures 3.6 and 3.7 show the obtained results by varying the number of paths generated by Yen algorithm, with $max\_paths \in \{5000, 500, 100, 50\}$ and $\alpha = 1.0$. Fixing the number of paths for Yen's algorithm means that the demands latency are tightened. We compare the values of the best solutions found within one hour (averaged over 10 graphs per instance type) for each of these settings.

Instances used in Figure 3.6 need at most 500 paths to be solved to optimality. For these instances, we varied the number of max_paths in $\{50, 100, 500\}$. We observe that adding a small number of paths permits `PF` to converge quickly to a very good solution for "Geant" and "Ta1" for which the number of required paths exceeds 200 paths. Conversely, for instances "Atlanta" and "Nobel-germany", which required only

53 and 149 paths, respectively, to be solved to optimality, 50 paths were insufficient to find a high-quality solution. This can be explained by the sparsity of their graphs and the number of demands.

Instances used in Figure 3.7 require more than 500 paths to be solved to optimality. We notice that for instances "Janos-us", the PF formulation was struggling to find a good quality solution. For Géant and Ta1, the increasing of costs is explained by the fact that Cplex reaches the time limit for all instances. Moreover, when there are too many paths (variables), Cplex needs more time to find a good solution, while it is faster with a fewer number of variables. We observe that in this case there is a trade-off between the quality of heuristic solution and the size of the underlying formulation.

For all other instances we can see that the number of fixed paths does not affect a lot the behavior of PF.

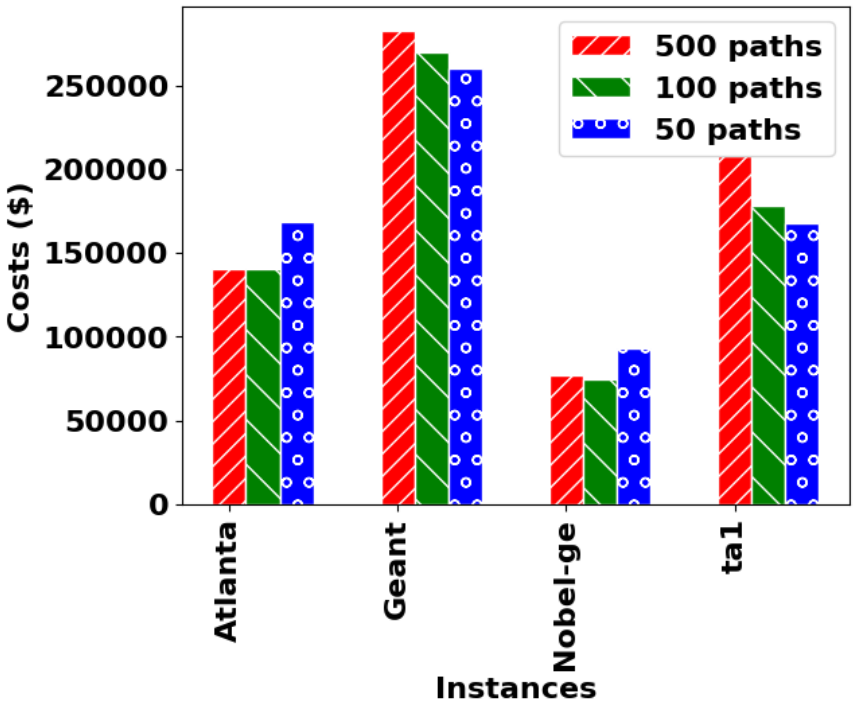


Figure 3.6: Costs comparison between path formulation with 500, 100 and 50 paths.

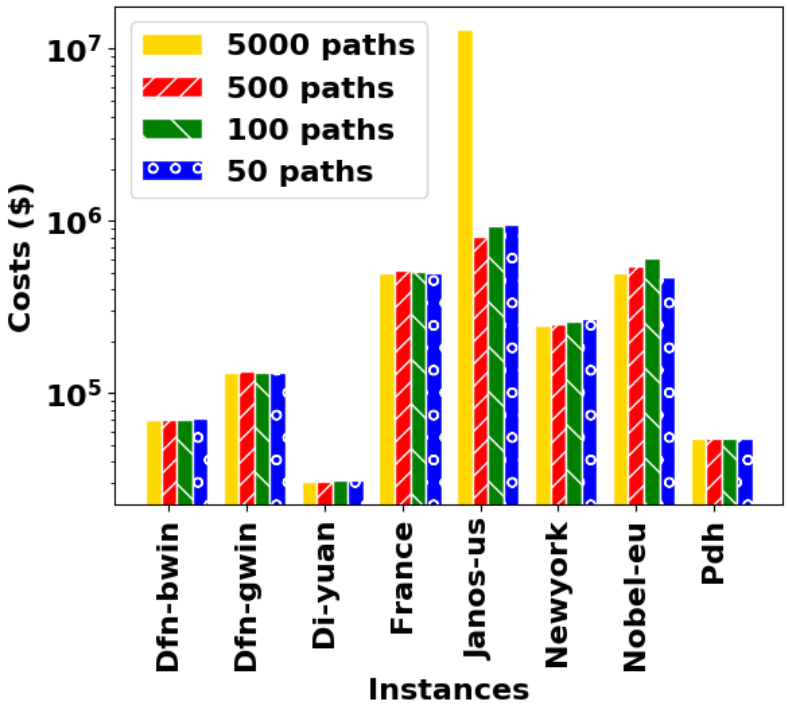


Figure 3.7: Costs comparison between path formulation with 5000, 500, 100 and 50 paths.

Figure 3.8 shows that increasing the function capacities allows for a significant cost reduction for all instances by -25,96 %. A similar effect can be achieved by increasing node capacities up to -2,78 %.

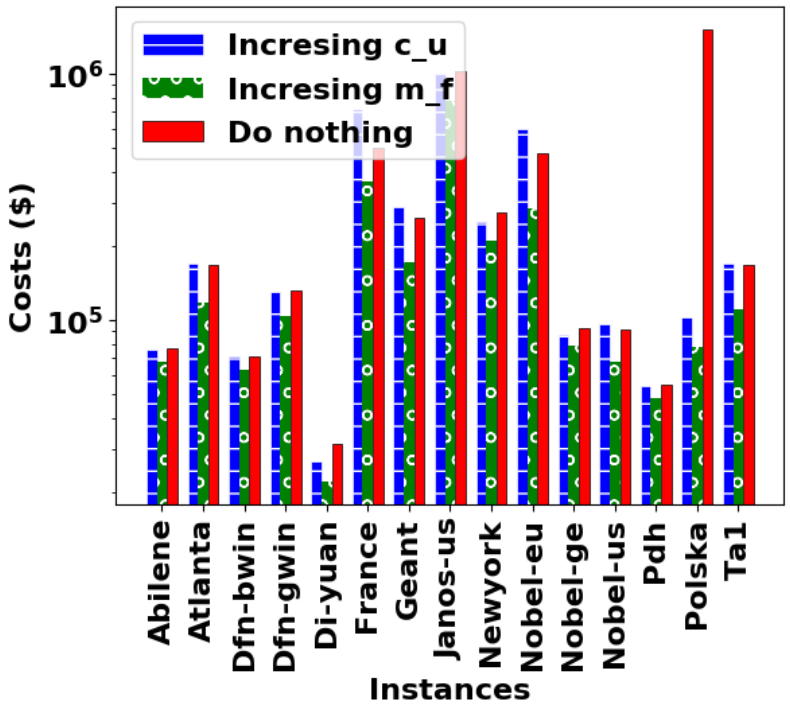


Figure 3.8: Costs comparison between path formulations with 50 path and with/without increasing node and functions capacities.

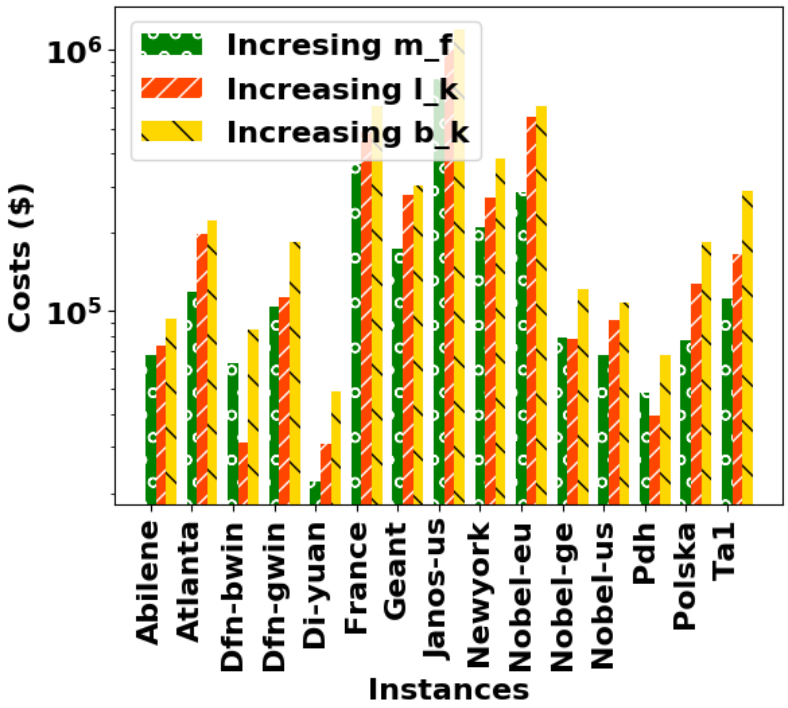


Figure 3.9: Costs comparison between the path formulations with 50 path and with increasing function capacities, bandwidth and latency

From Figure 3.9 we notice that increasing bandwidth of demands increases costs for all solved instances up to 46,34 %. On the other hand, increasing latency for "Dfn-bwin" and "Pdh" allows cost reduction, both instances have almost the same sparse graph topology. Increasing the bandwidth value for some instances like "Polska" and "Dfn-gwin" makes them infeasible. Conversely increasing the latency value makes the problem easy to solve, so we can have an optimal solution very quickly. Increasing the latency increase the cost by +29,96 %.

3.2.4 Detailed results

Abbreviations	Description
CPU_H	Heuristic CPU time in seconds.
RH	Heuristic LP-relaxation value.
LB_H	Heuristic best known lower bound, provided by Cplex.
Gap_H	Heuristic relative gap, provided by Cplex.
UB_C	Heuristic best known global upper bound, provided by Cplex.
CPU_C	Compact formulation CPU time in seconds.
RC	LP-relaxation value of the compact formulation.
LB_C	Best known lower bound, provided by Cplex for the compact formulation.
Gap_C	Relative gap, provided by Cplex for the compact formulation.
UB_C	Best known global upper bound, provided by Cplex for the compact formulation.

Table 3.8: Description of Tables 3.9-3.13 abbreviations

Notation "'TL" in tables means that the time limit is exceeded and the sign "$-$" means that no feasible solution is found by the model.

Instance	CPU_H	RH	LB_H	Gap_H	UB_H	CPU_C	RC	LB_C	Gap_C	UB_C
Abilene	TL	105520.02	114675	3.87	110238.19	TL	96349.60	121732	11.05	108280.35
	TL	78797.52	89999	1.71	88463.96	TL	70104.38	102227	17.51	84325.84
	TL	59169.54	68811	0.27	68623.06	TL	50815.42	68999	1.29	68106.14
	648.89	57451.58	68227	0.00	68227.00	TL	48503.40	68227	0.34	67997.68
	314.00	46198.09	55712	0.00	55712.00	1533.09	40452.10	55712	0.00	55712.00
	2650.30	66227.75	73696	0.00	73696.00	TL	57305.83	74926	2.58	72989.17
	TL	96449.21	107282	0.29	106969.36	TL	89779.21	111873	7.24	103773.29
	1168.32	58401.24	66164	0.00	66164.00	TL	50181.90	67942	4.64	64787.33
	736.15	49122.61	58900	0.00	58900.00	1844.75	40260.73	58900	0.00	58900.00
	1224.18	58136.28	68119	0.00	68119.00	TL	45544.12	69365	3.86	66684.41
Atlanta	TL	201710.88	342809	36.97	216085.25	TL	187685.36	455555	55.87	201047.36
	TL	76950.20	134316	38.79	82215.03	TL	65595.14	171546	53.71	79415.00
	TL	83414.38	139618	36.08	89241.70	TL	75171.64	270340	67.91	86748.61
	TL	76452.75	98392	16.53	82132.51	TL	65463.21	140690	42.68	80637.43
	TL	92908.94	112936	12.16	99207.01	TL	79399.97	244878	61.30	94777.48
	TL	72291.24	112766	29.97	78972.24	TL	60813.42	152840	53.12	71651.54
	TL	106620.36	157292	27.04	114763.16	TL	98220.61	473321	76.86	109548.74
	TL	92200.14	124327	19.77	99747.22	TL	84582.53	361504	73.80	94728.71
	TL	75368.65	99427	19.67	79870.87	TL	64238.76	128957	39.33	78239.26
	TL	156867.94	246773	33.30	164590.25	TL	145237.07	388732	59.10	158977.90
Dfn-bwin	TL	64276.96	71938	1.42	70918.29	TL	48543.97	72194	1.71	70960.00
	TL	58813.46	66658	2.98	64674.06	TL	43190.17	66269	2.36	64703.00
	TL	66776.57	73804	1.53	72674.66	TL	51086.81	73695	1.26	72765.00
	TL	55042.21	62550	3.15	60580.74	TL	42190.82	62340	2.76	60618.34
	TL	75887.37	82946	2.53	80851.30	TL	59611.15	82931	2.45	80896.00
	TL	58328.00	65148	0.64	64729.00	TL	42820.20	65148	0.60	64758.00
	TL	56155.05	63674	2.81	61882.64	TL	43048.82	63340	2.22	61934.45
	TL	78460.89	84036	0.25	83829.45	TL	60745.40	84250	0.41	83908.00
	TL	67966.66	73426	0.96	72720.00	TL	52554.58	73317	0.50	72951.62
	TL	57564.46	62990	1.47	62062.69	TL	43521.76	63081	1.50	62137.64

Table 3.9: Results comparison between MILP-Based Heuristic and compact MILP formulation.

Instance	CPU_H	RH	LB_H	Gap_H	UB_H	CPU_C	RC	LB_C	Gap_C	UB_C
Dfn-gwin	TL	110630.85	124536	8.19	114330.46	TL	93821.72	129047	11.44	114280.35
	TL	130506.55	157038	14.57	134159.45	TL	118367.32	151463	11.40	134192.26
	TL	95595.68	111488	10.19	100128.50	TL	77911.29	113279	11.66	100072.27
	TL	87882.56	102521	9.44	92845.32	TL	72339.77	105326	11.79	92910.44
	TL	118597.77	129043	4.66	123032.84	TL	105110.36	132226	6.90	123096.30
	TL	117281.23	135755	9.95	122240.80	TL	101920.72	130431	6.19	122360.65
	TL	120269.45	132721	6.13	124582.58	TL	105783.60	133469	6.79	124406.07
	TL	141079.29	154456	5.90	145343.16	TL	123237.29	152082	4.48	145263.03
	TL	100273.40	113761	8.11	104530.45	TL	86216.46	116757	10.54	104454.14
	TL	139203.06	151521	5.32	143458.98	TL	124883.21	153757	6.69	143469.36
Di-yuan	TL	23710.31	26434	5.24	25047.54	TL	23367.81	30114	17.55	24829.23
	TL	32425.34	37218	9.88	33541.80	TL	32295.12	43227	22.46	33518.40
	TL	25468.11	28716	3.81	27621.74	TL	24334.12	33484	19.14	27076.03
	TL	24355.02	29774	12.44	26069.70	TL	23803.91	31854	18.16	26069.27
	TL	23446.32	28300	12.15	24862.35	TL	22808.12	34687	29.56	24431.85
	TL	26091.33	29122	2.97	28258.20	TL	25128.40	37195	25.33	27773.67
	TL	31165.59	35617	5.40	33695.40	TL	29156.42	40179	16.12	33700.80
	TL	25824.91	34325	18.23	28067.04	TL	25619.04	39170	27.21	28510.55
	TL	25982.10	31075	10.62	27775.33	TL	25561.37	32548	13.88	28031.58
	TL	23071.91	27016	7.40	25017.75	TL	22278.40	28582	15.51	24148.23
France	TL	209682.80	503447	56.48	219100.12	TL	177774.80	673370	69.42	205891.57
	TL	146961.89	455088	65.99	154793.98	TL	129220.07	646723	77.15	147748.06
	TL	199678.93	585092	64.39	208350.04	TL	174563.06	704671	72.75	191988.23
	TL	211555.63	456530	51.24	222612.17	TL	181826.41	637928	68.07	203678.70
	TL	158866.78	553185	70.00	165951.80	TL	139979.59	690638	76.90	159531.84
	TL	266668.07	615896	55.29	275382.73	TL	235734.13	743322	66.02	252556.79
	TL	250292.21	428400	39.60	258762.19	TL	224804.23	693888	64.66	245227.88
	TL	250355.76	542799	52.38	258499.64	TL	230949.85	609195	59.66	245733.54
	TL	144357.63	326375	52.76	154171.06	TL	118290.50	463475	68.98	143772.28
	TL	292176.86	510860	40.61	303416.51	TL	264660.39	651481	56.08	286114.53

Table 3.10: Results comparison between MILP-Based Heuristic and compact MILP formulation.

Instance	CPU_H	RH	LB_H	Gap_H	UB_H	CPU_C	RC	LB_C	Gap_C	UB_C
Geant	TL	91476.97	162198	33.26	108245.50	TL	77238.42	147568	26.64	108253.33
	TL	111167.38	185942	32.27	125937.30	TL	90112.74	167340	25.19	125190.60
	TL	207991.33	885567	74.92	222087.22	TL	188864.38	1012275	78.06	222085.94
	TL	103714.90	163215	26.85	119397.16	TL	86437.31	157715	23.96	119922.21
	TL	127883.50	270154	47.19	142671.28	TL	111260.20	191529	25.49	142712.27
	TL	141890.02	329112	53.13	154248.17	TL	123382.34	1090981	85.85	154365.54
	TL	118231.64	360569	63.90	130179.11	TL	100105.48	676904	81.02	128477.31
	TL	167243.54	215315	15.10	182807.91	TL	147328.25	778519	76.54	182635.56
	TL	95413.91	134712	17.55	111063.69	TL	78866.47	142442	22.11	110949.97
	TL	110790.52	141694	10.82	126360.57	TL	92770.69	145110	13.17	125993.68
Janos-us	TL	10887359.23	10904806	0.08	10896593.30	TL	10887359.23	11382718	4.28	10895750.90
	TL	11342337.18	11360428	0.05	11354528.77	TL	11342337.18	13070200	13.13	11353640.23
	TL	14005192.67	14019188	0.03	14014765.63	TL	14005192.67	15379304	8.88	14014082.08
	TL	11873470.30	11892690	0.03	11888550.07	TL	11873470.30	13822127	14.00	11887417.16
	TL	15534868.92	15551362	0.04	15545762.85	TL	15534868.92	15570677	0.16	15545076.64
	TL	14555386.83	14572514	0.04	14566875.56	TL	14555386.83	14580575	0.10	14565999.76
	TL	12625202.51	12643229	0.05	12637097.00	TL	12625202.51	12648938	0.10	12636608.73
	TL	12848111.78	12862499	0.02	12859573.33	TL	12848111.78	12870288	0.09	12858854.98
	TL	12537209.83	12555350	0.03	12551172.35	TL	12537209.83	12570924	0.16	12550414.28
	TL	15839879.13	15853396	0.02	15849940.58	TL	15839879.13	15858716	0.06	15849306.31
Newyork	TL	162178.15	255618	34.44	167590.28	TL	149337.54	466578	64.35	166338.93
	TL	226266.77	425812	45.77	230902.26	TL	211270.08	460553	49.65	231888.21
	TL	104449.65	156045	30.78	108019.14	TL	87961.14	359078	70.03	107602.64
	TL	177077.07	233906	21.39	183865.69	TL	155144.28	456137	59.71	183782.67
	TL	146316.71	187556	19.16	151619.43	TL	131163.40	233563	36.01	149449.93
	TL	104367.65	142370	22.06	110956.44	TL	90660.41	184876	41.27	108584.56
	TL	210317.85	300595	28.43	215135.32	TL	199250.03	417816	48.54	215000.02
	TL	160831.54	258889	35.76	166321.06	TL	149434.14	391109	58.14	163720.47
	TL	109303.66	307996	62.74	114744.64	TL	91031.13	184684	38.11	114305.83
	TL	169780.71	220758	20.45	175613.29	TL	147659.60	270668	35.30	175110.97

Table 3.11: Results comparison between MILP-Based Heuristic and compact MILP formulation.

Instance	CPU_H	RH	LB_H	Gap_H	UB_H	CPU_C	RC	LB_C	Gap_C	UB_C
Nobel-eu	TL	120670.98	770133	83.42	127684.52	TL	108330.18	–	–	–
	TL	152736.31	708663	77.72	157873.93	TL	131278.09	–	–	–
	TL	273774.44	368822	23.96	280450.66	TL	257646.79	–	–	–
	TL	279553.91	537904	47.32	283378.73	TL	266694.08	–	–	–
	TL	99610.93	308532	65.35	106914.14	TL	88728.30	411905	74.40	105456.84
	TL	143465.19	943711	84.19	149216.55	TL	130476.89	639078	77.27	145238.05
	TL	116307.21	239790	48.81	122757.78	TL	104054.56	628717	80.60	121959.23
	TL	111621.79	331527	64.13	118910.39	TL	94211.05	475501	76.38	112290.76
	TL	279722.97	506774	42.21	292869.90	TL	265906.49	753040	62.29	283984.67
	TL	165525.45	250304	30.49	173986.45	TL	150630.18	694038	75.78	168085.82
Nobel-ger	TL	62615.07	71490	6.97	66505.18	TL	55711.09	104103	37.05	65532.57
	TL	52706.33	58160	3.25	56270.01	TL	44959.19	86285	34.80	56257.18
	TL	78947.90	89974	6.50	84127.03	TL	69743.46	134963	40.80	79896.42
	TL	69514.62	86168	14.41	73754.76	TL	62029.70	130569	44.75	72136.98
	TL	78449.10	105203	22.72	81300.58	TL	74763.61	150460	46.96	79805.52
	TL	73723.82	83586	7.12	77633.60	TL	66893.27	150787	51.24	73518.01
	TL	54285.12	60091	4.12	57615.22	TL	46931.61	119016	53.42	55433.86
	TL	56841.39	63807	4.03	61237.71	TL	51239.35	112900	47.72	59023.63
	TL	78594.93	94609	12.16	83107.32	TL	72577.69	138111	41.28	81097.60
	TL	51323.03	58437	5.28	55351.94	TL	44237.53	113435	53.10	53199.58
Nobel-us	TL	57866.99	64908	5.37	61419.61	TL	51971.67	79287	23.87	60359.93
	TL	50563.16	65188	18.11	53380.29	TL	44398.81	97496	46.50	52164.02
	TL	111129.96	141625	17.58	116730.95	TL	103875.95	155983	26.39	114818.67
	TL	74329.68	134983	42.32	77858.98	TL	68501.25	129906	41.59	75877.57
	TL	124607.33	164645	20.40	131053.25	TL	117886.87	193246	33.18	129125.75
	TL	57692.81	100274	38.07	62098.96	TL	46824.90	101846	40.40	60697.28
	TL	54553.80	74501	21.53	58459.39	TL	47960.09	97320	41.07	57346.84
	TL	66320.66	96427	25.07	72255.63	TL	62032.18	125415	44.64	69431.48
	TL	48062.51	64788	17.59	53392.11	TL	43797.41	101285	50.18	50457.68
	TL	58479.55	71403	13.06	62079.35	TL	51816.35	104641	41.49	61224.90

Table 3.12: Results comparison between MILP-Based Heuristic and compact MILP formulation.

Instance	CPU_H	RH	LB_H	Gap_H	UB_H	CPU_C	RC	LB_C	Gap_C	UB_C
Pdh	1407.39	51627.24	56518	0.00	56518.00	2141.91	38805.94	56518	0.00	56518.00
	TL	61181.17	66862	0.50	66526.00	*TL*	45729.75	66862	0.49	66537.25
	TL	48768.24	51817	0.72	51442.00	*TL*	38932.42	51989	1.05	51442.24
	TL	45046.45	49451	1.60	48661.72	*TL*	36016.09	49517	1.50	48772.25
	TL	39659.06	43913	2.94	42623.36	*TL*	30850.73	44116	3.33	42647.65
	TL	43576.62	47046	1.64	46272.47	*TL*	32338.46	47046	1.60	46291.68
	1152.43	52391.94	55876	0.00	55876.00	1032.78	39393.92	55876	0.00	55876.00
	TL	45912.24	50606	2.25	49465.00	*TL*	37821.69	50606	1.82	49684.82
	TL	51435.68	57071	1.59	56161.00	*TL*	38734.60	57071	1.64	56135.30
	TL	60916.07	65921	0.49	65600.00	*TL*	50119.97	66015	0.54	65660.00
Polska	*TL*	84198.70	108842	17.67	89608.52	*TL*	82914.01	137978	37.29	86522.85
	TL	107945.27	125051	9.88	112697.10	*TL*	106159.63	142278	21.88	111142.83
	TL	88444.81	108731	12.88	94725.86	*TL*	85717.67	132934	31.35	91255.38
	TL	89805.46	116229	13.23	100855.09	*TL*	85426.18	146856	35.60	94570.84
	TL	82866.71	133347	29.39	94154.34	*TL*	81055.23	130412	32.72	87744.08
	TL	97408.93	129216	18.00	105961.46	*TL*	94797.85	143218	28.64	102198.21
	TL	96297.61	114620	11.06	101947.38	*TL*	95199.63	135530	26.05	100218.98
	TL	93893.16	112466	10.43	100732.87	*TL*	91039.84	148651	34.62	97190.71
	TL	94575.33	126803	20.57	100714.17	*TL*	92574.06	132398	26.11	97832.97
	TL	70918.74	91099	13.59	78714.83	*TL*	68545.87	122598	38.70	75147.37
Ta1	*TL*	100868.31	148197	23.73	113030.26	*TL*	82715.84	–	–	–
	TL	80932.27	142626	36.31	90845.49	*TL*	73074.28	148201	38.87	90592.97
	TL	79953.34	141155	36.98	88960.19	*TL*	63090.38	126016	29.69	88602.56
	TL	80115.91	150423	40.70	89202.02	*TL*	65060.97	207975	57.13	89154.76
	TL	88812.96	133654	25.60	99437.24	*TL*	71865.36	294566	66.63	98300.08
	TL	104637.53	334228	66.49	111984.62	*TL*	87691.33	306170	63.50	111764.07
	TL	118075.41	198805	34.81	129607.03	*TL*	100768.98	925598	85.98	129798.82
	TL	113621.32	583691	79.11	121925.50	*TL*	96286.92	960568	87.46	120413.50
	TL	98045.52	139050	21.00	109849.84	*TL*	80692.01	250241	56.22	109551.25
	TL	84717.46	138082	32.68	92953.90	*TL*	67679.75	166108	44.24	92618.86

Table 3.13: Results comparison between MILP-Based Heuristic and compact MILP formulation.

3.3 Conclusions

In this chapter, we have proposed a path-based MILP formulation for the VNFPRP. With this formulation we were able to provide optimal solutions for additional instances and to improve the gaps obtained from the compact formulation. We then derived an MILP-based heuristic that we have compared to the compact MILP formulation, in terms of the CPU time and the quality of the obtained solution. We have also tested the path formulation with different values of problem parameters and discussed the difference in terms of costs and (in)feasibility.

The proposed Path-based formulation has shown its effectiveness for finding high-quality solutions. In the next chapter, we will apply a column generation algorithm to the path formulation to solve its linear relaxation and derive a Branch-and-Price algorithm to solve it.

Chapter 4

Extended formulations

In this chapter, we present two column generation approaches to tackle the Virtual Network Functions Placement and Routing problem. First, we propose two extended MILP formulations to model it, for which we then derive branch-and-price algorithms. For each formulation, we present the pricing problem, its resolution method, computation of the Lagrangian bound, and the branching scheme. Both approaches are empirically compared, and experiments are conducted on realistic instances to show their efficiency.

Contents

Previous studies related to the simultaneous placement of ordered VNFs and routing of traffic demands mainly rely on compact MILP formulations, whose computational performance is fairly limited. To enhance the capabilities of finding exact solutions for larger instances of practical relevance, we focus in this part of the thesis, on the extended formulations with an exponential number of variables. Several studies and works were proposed to deal with MILP compact formulations performance limitation. According to Sadykov and Vanderbeck [117], working in an extended variable space allows one to develop tight reformulations for mixed-integer programs. Ford and Fulkerson [55] are the first that use an optimization sub-problem to price out an exponential number of non-basic variables, which allow them to add only basic variables and reduce the model size. Bilde and Krarup [24] show that the extended facility location reformulation for uncapacitated lot-sizing was integral, this means that solving the linear relaxation of the reformulation can give an integer optimal solution without any branching. Chopra et al. [34] propose an extended formulation for the Convex Recoloring problem on a tree. They show that the LP-relaxation of the extended formulation provides an integer optimum for all considered problem instances, which indicates that the LP-relaxation of the extended formulation is a very good approximation of the integer polytope. Vanderbeck and Wolsey [135] survey the main reformulations based on decomposition methods, such as Lagrangian relaxation, Dantzig-Wolfe, and the resulting branch-and-price algorithms and Benders' reformulation, they also discuss in detail extended formulations.

in this part of the thesis, we propose two *extended MIP formulations* for the VNF-PRP. In both cases, we consider an extended variable space admitting an exponential number of variables which allows us to develop tighter MIP reformulations.

In the first reformulation, we separate the VNF placement problem, which is treated at the master level, from the routing problem, which is solved separately for each commodity in the pricing problem. The second extended formulation arises from an alternative Dantzig-Wolfe decomposition approach. The problem is decomposed per commodity in such a way that: the master problem ensures that exactly one path with its associated VNF installations is chosen for each commodity, and that node and VNFs capacity constraints are satisfied. The routing and VNF placement constraints associated for each commodity are managed in the pricing problems.

In order to improve the LP-relaxation bounds of our formulations, new families of valid inequalities are also proposed. All these elements are combined in two efficient Branch-and-Price algorithms. A detailed computational comparison of the Branch-and-Price algorithms against the compact MIP formulation and the automatic Benders decomposition approach by the commercial solver Cplex is given.

Outline of the chapter The chapter is organized as follows. In Sections 4.1 and 4.2, we present the two extended formulations. We discuss the associated pricing problems, detail the computation of the Lagrangian bound and present branching schemes. In Section 4.3, we provide a new set of valid inequalities that aim to strengthen the LP-relaxation bounds. In Section 4.4 we provide implementation details of Branch-and-Price (B&P) algorithms for each formulation. In Section 4.5, we provide a theoretical result proving that the Dantzig-Wolfe formulation provides better LP-relaxation bounds than the path formulation. In Section 4.6, we discuss the obtained numerical results, and conclude with some remarks and perspectives in Section 4.7.

4.1 First extended formulation: the model PF

In this section, we present the first extended MILP formulation, denoted by PF (which stands for "path-based formulation") to model the VNFPRP. The formulation has been introduced in Chapter 3 to generate heuristic solutions, by considering a compact model obtained from choosing a small but promising number of columns. In this chapter instead we focus on developing an exact method for solving the path formulation, based on a Branch-and-Price procedure. In this model we use latency-constrained elementary path variables associated to each commodity to model routing decisions. In this section we discuss theoretical properties of this model, along with a derivation of a valid Lagrangian bound, whereas the details related to the B&P implementation are given in Section 4.4. The master problem aims to find the optimal VNFs installation for a given routing path for each traffic request. One pricing problem is defined per commodity $k \in C$, and it consists on determining an $s_k - d_k$ latency-constrained elementary routing path.

4.1.1 Decision variables

The set of variables required for the path formulation is already defined in Chapter 3, Table 3.1. Similarly, the constraints modeling the path formulation are presented in Chapter 3. We recall the path formulation variables and MILP model in order to derive the associated dual formulation.

Let us denote by $\mathcal{P}_k$ the set of all latency-constrained elementary paths associated with commodity $k \in C$. We assume that the set $\mathcal{P}_k$ is given and that, for each chosen path, the arcs composing it are known. Let t_{uv}^{pk} be the parameter that is equal to 1 if arc (u, v) belongs to path $p, p \in \mathcal{P}_k$, and equal to 0 otherwise.

Variables		Type
λ_p^k	1, if path $p \in \mathcal{P}_k$ associated with commodity k is chosen; 0, otherwise.	Binary
x_u^{fk}	1, if the virtual network function f is installed at or before node u for commodity k; 0, otherwise.	Binary
y_u^{fk}	1, if virtual network function f is installed at node u for commodity k; 0, otherwise.	Binary
w_u	1, if node u is activated; 0, otherwise.	Binary
z_u^f	number of VNF f installed at node u.	Integer

Table 4.1: Decision variables of the path formulation

$$t_{uv}^{pk} = \begin{cases} 1 & \text{if arc } (u,v) \in A \text{ belongs to path } p \in \mathcal{P}_k \text{ associated to commodity } k \in C, \\ 0 & \text{otherwise.} \end{cases}$$

4.1.2 The master problem formulation

The VNFPRP can be modeled as follows:

$$(PF): \quad \min \quad \sum_{u \in N} \sum_{f \in F} \psi_u^f z_u^f + \sum_{u \in N} \psi_u w_u \tag{4.1}$$

$$\sum_{p \in \mathcal{P}_k} \lambda_p^k = 1 \qquad k \in C \quad (\alpha_k) \tag{4.2}$$

$$\sum_{f \in F} z_u^f \leq c_u w_u \qquad u \in N \quad (\beta_u) \tag{4.3}$$

$$\sum_{k \in C} y_u^{fk} b_k \leq m_f z_u^f \qquad f \in F, u \in N \quad (\gamma_u^f) \tag{4.4}$$

$$y_u^{fk} + y_u^{gk} \leq 1 \qquad k \in C, (f,g) \in \mathcal{A}^k, u \in N \quad (\delta_u^{(f,g)k}) \tag{4.5}$$

$$\Big(\sum_{\substack{p \in \mathcal{P}_k \\ (u,v) \in p}} t_{uv}^{pk} \lambda_p^k - 1 \Big) + (x_v^{fk} - x_u^{fk}) \leq y_v^{fk} \quad k \in C, f \in F^k, (u,v) \in A \quad (\eta_{uv}^{fk}) \tag{4.6}$$

$$x_u^{gk} \leq x_u^{fk} \qquad k \in C, f, g \in F^k : f \prec_k g, u \in N \quad (\varphi_u^{(f,g)k}) \quad (4.7)$$

$$y_u^{fk} \leq x_u^{fk} \qquad k \in C, f \in F^k, u \in N \quad (\theta_u^{fk}) \quad (4.8)$$

$$y_u^{fk} \leq \sum_{(v,u)\in A} \sum_{p\in\mathcal{P}_k} t_{vu}^{pk} \lambda_p^k \qquad k \in C, f \in F^k, u \in N \quad (\pi_u^{fk}) \quad (4.9)$$

$$\sum_{u\in N} y_u^{fk} \geq 1 \qquad k \in C, f \in F^k \quad (\vartheta^{fk}) \quad (4.10)$$

$$x_u^{fk} = \begin{cases} 0, & u = s_k \\ 1, & u = d_k \end{cases} \qquad k \in C, f \in F^k \quad (\sigma_{d_k}^{fk}) \quad (4.11)$$

$$(\lambda, x, y, w) \text{ are binary} \quad (4.12)$$

$$z \text{ is integer} \quad (4.13)$$

Constraints (4.2) are the path constraints which ensure that exactly one elementary latency-constrained path $p \in \mathcal{P}_k$ is chosen for each commodity $k \in C$. Constraints (4.3) represent the node capacity constraints, which guarantee that the number of VNFs installed at each node $u \in N$ is bounded by its capacity c_u. Constraints (4.4) are the VNF capacity constraints. They ensure that the volume of data treated by each function $f \in F$ should not exceed its capacity m_f. Constraints (4.5) are the conflict constraints and they guarantee that two VNFs in conflict are not installed at the same node $u \in N$. Constraints (4.6) are needed to link node installation variables (y), precedence variables (x) and path variables (λ): the left-hand-side is forced to 1 (implying that the function f is installed at the node v) if and only if (i) the path p passing through the arc (u, v) is chosen for the considered commodity k and (ii) the function f is installed at or before the node v and it is not installed at or before the node u. Constraints (4.7) impose the VNFs order for each commodity. Inequalities (4.8) link the precedence and the installation variables, x and y, and express the fact that if VNF f is installed at node u for the commodity k, then f is installed at or before the node u. Constraints (4.9) ensure that if a VNF $f \in F^k$ is installed at a node u for a given commodity k, then the associated routing path p must enter that node. Constraints (4.10) guarantee that all required functions for commodity $k \in C$ are installed at the graph nodes. Finally, constraints (4.11) guarantee that, for each commodity $k \in C$, no VNF is installed at or before the source node s_k and all VNFs are installed at or before the destination node d_k. Model (4.1)-(4.13) admits an exponential number of path variables; thus, a column generation (CG) procedure is needed to solve its continuous relaxation. In the following, we describe the pricing problem and discuss three possible procedures for its resolution.

The LP-relaxation of this model is obtained by replacing constraints (4.12)-(4.13) with $\lambda \geq 0$, $0 \leq x \leq 1$, $y \geq 0$, $0 \leq w \leq 1$ and $z \geq 0$.

4.1.3 The dual of the master problem

Let DPF denote the dual formulation of the LP-relaxation of the model PF. The number of routing paths associated with each commodity can be exponential. Thus, the number of constraints associated to the path variables is exponential in the dual. The associate dual variables are shown next to each constraint from the formulation given in Subsection 4.1.2, in particular, we associate α, η, and π to constraints (4.2), (4.6) and (4.9), respectively.

The dual of the master problem is given by the following linear program:

$$(DPF): \quad \max \sum_{k\in C} \alpha_k - \sum_{k\in C}\sum_{u\in N}\sum_{(f,g)\in S^k} \delta_u^{(f,g)k} - \sum_{k\in C}\sum_{f\in F^k}\sum_{(u,v)\in A} \eta_{uv}^{fk} + \sum_{k\in C}\sum_{f\in F^k} \vartheta^{fk} + \sum_{k\in C}\sum_{f\in F^k} \sigma_{d_k}^{fk} - \sum_{k\in C}\sum_{f\in F^k}\sum_{u\in N} \Delta_u^{fk} - \sum_{u\in N} \Omega_u$$

$$\alpha_k - \sum_{(u,v)\in A}\sum_{f\in F^k} t_{uv}^{pk}\ \eta_{uv}^{fk} + \sum_{u\in N}\sum_{v\in\Gamma^-(u)}\sum_{f\in F^k} t_{vu}^{pk}\ \pi_u^{fk} \le 0 \qquad k\in C, p\in\mathcal{P}_k$$

$$c_u\ \beta_u - \Omega_u \le \psi_u \qquad u\in N$$

$$-\beta_u + m_f\gamma_u^f \le \psi_u^f \qquad f\in F, u\in N$$

$$-b_k\gamma_u^f - \sum_{\substack{g\in F^k\\(f,g)\in S^k}} \delta_u^{(f,g)k} - \sum_{\substack{g\in F^k\\(g,f)\in S^k}} \delta_u^{(g,f)k} \sum_{v\in\Gamma^+(u)} \eta_{uv}^{fk}$$
$$-\theta_u^{fk} - \pi_u^{fk} + \vartheta^{fk} \le 0 \qquad k\in C,\quad f\in F^k,\quad u\in N$$

$$-\sum_{v\in\Gamma^-(u)} \eta_{vu}^{fk} + \sum_{v\in\Gamma^+(u)} \eta_{uv}^{fk} - \sum_{\substack{g\in F^k\\(f,g)\in S^k}} \varphi_u^{(f,g)k} + \sum_{\substack{g\in F^k\\(g,f)\in\mathcal{A}^k}} \varphi_u^{(g,f)k} + \theta_u^{fk}$$
$$-\Delta_u^{fk} \le 0 \qquad k\in C,\quad f\in F^k,\quad u\in N\setminus\{s_k,d_k\}$$

$$-\sum_{v\in\Gamma^-(s_k)} \eta_{vs_k}^{fk} + \sum_{v\in\Gamma^+(s_k)} \eta_{s_kv}^{fk} - \sum_{\substack{g\in F^k\\(f,g)\in S^k}} \varphi_u^{(f,g)k}$$
$$+\sum_{\substack{g\in F^k\\(g,f)\in\mathcal{A}^k}} \varphi_{s_k}^{(g,f)k} + \theta_{s_k}^{fk} + \sigma_{s_k}^{fk} - \Delta_{s_k}^{fk} \le 0 \qquad k\in C,\quad f\in F^k$$

$$
\begin{aligned}
&- \sum_{v \in \Gamma^-(d_k)} \eta^{fk}_{vd_k} + \sum_{v \in \Gamma^+(d_k)} \eta^{fk}_{d_k v} - \sum_{\substack{g \in F^k \\ (f,g) \in S^k}} \varphi^{(f,g)k}_{d_k} \\
&+ \sum_{\substack{g \in F^k \\ (g,f) \in \mathcal{A}^k}} \varphi^{(g,f)k}_{d_k} + \theta^{fk}_{d_k} + \sigma^{fk}_{d_k} - \Delta^{fk}_{d_k} \quad \leq \quad 0 \qquad\qquad k \in C, \quad f \in F^k
\end{aligned}
$$

$$
\beta, \gamma, \delta, \eta, \varphi, \theta, \pi, \vartheta, \sigma, \Delta, \Omega \quad \geq \quad 0, \qquad \alpha_k \quad \in \quad \mathbb{R}^{|C|}
$$

Then, for each $k \in C$ and $p \in \mathcal{P}_k$, the dual constraint associated with the path variable λ^k_p is given as follows:

$$
\alpha_k - \sum_{f \in F^k} [\sum_{(u,v) \in A} t^{pk}_{uv} \eta^{fk}_{uv} + \sum_{u \in N} \sum_{v \in \Gamma^-(u)} t^{pk}_{vu} \pi^{fk}_u] \quad \leq \quad 0, \qquad k \in C, p \in \mathcal{P}_k \tag{4.14}
$$

$$
\iff \quad \alpha_k + \sum_{(u,v) \in A} \sum_{f \in F^k} (\pi^{fk}_v - \eta^{fk}_{uv}) t^{pk}_{uv} \quad \leq \quad 0, \qquad k \in C, p \in \mathcal{P}_k \tag{4.15}
$$

4.1.4 The pricing problem

As customary in column generation, the master problem is initialized with a subset of λ variables (resulting in the so-called *restricted master problem*), and then the additional variables necessary to solve the LP-relaxation of the model are generated on the fly by separating the associated dual constraints (4.15). The *pricing problem* then consists of finding for each commodity k, a path $p \in \mathcal{P}_k$ with negative reduced costs, i.e., a path p such that:

$$
\alpha^*_k + \sum_{(u,v) \in p} \sum_{f \in F^k} (\pi^{*fk}_v - \eta^{*fk}_{uv}) \quad > \quad 0, \tag{4.16}
$$

where $(\alpha^*, \eta^*, \pi^*)$ refers to a sub-vector of an optimal dual solution of the restricted master problem. This dual solution will be used for defining the pricing problems and computing the Lagrangian bound (cf. Section 4.1.5).

Thus, for each commodity $k \in C$, a separate pricing problem is defined in order to find an $s_k - d_k$ latency-constrained elementary path of minimum cost. Cost per each arc is defined as

$$
\tilde{c}_{uv} = \sum_{f \in F^k} (\eta^{*fk}_{uv} - \pi^{*fk}_v), \quad (u,v) \in A. \tag{4.17}
$$

If we find a path $p \in P_k$ such that $\sum_{(u,v)\in p} \tilde{c}_{uv} - \alpha_k^* < 0$, the associated variable λ_p^k will be inserted in the restricted master problem. Based on the dual solution, the values defined by (4.17) may be negative. Therefore the pricing problem consists of finding an elementary shortest path satisfying latency constraints on a graph that may contain negative cycles. Hence, the pricing problem is *strongly* NP-hard [45]. Three different methods are proposed and computationally evaluated for solving this pricing problem (cf. Sections 4.4.3 and 4.6, respectively).

4.1.5 Lagrangian bound

In the following, we describe how to derive the Lagrangian bound of the model PF which can be used to minimize the number of iterations in the column generation procedure. Let DRPF denote the LP-dual of the restricted master problem of the PF model, let Υ^* be its optimal solution, Z_{DRPF} the associated objective value, and let DPF denote the dual of the model PF (including all columns). We will slightly abuse the notation and focus only on the (α, π, η) components of the vector Υ.

Υ^* satisfies all constraints of DRPF, but not necessarily all constraints of DPF, i.e., Υ^* is optimal for DRPF, but not necessarily feasible for DPF, as the constraints associated with the columns that are left out from the master are missing.

In what follows, we will show how to construct a feasible solution for DPF during the column generation phase. Let Z_k^* be defined as:

$$Z_k^* = \min_{p\in\mathcal{P}_k}[-\alpha_k^* - \sum_{(u,v)\in p}\sum_{f\in F^k}(\pi_v^{*f} - \eta_{uv}^{*f})] = -\alpha_k^* - \max_{p\in\mathcal{P}_k}\sum_{(u,v)\in p}\sum_{f\in F^k}(\pi_v^{*f} - \eta_{uv}^{*f}).$$

So, Z_k^* is the minimum reduced cost associated with commodity k (over all paths $p \in \mathcal{P}_k$). Therefore, for each commodity $k \in C$ and each path $p \in \mathcal{P}_k$, it holds:

$$Z_k^* \leq -\alpha_k^* - \sum_{(u,v)\in p}\sum_{f\in F^k}(\pi_v^{*f} - \eta_{uv}^{*f})$$

Consequently, constraints (4.15) in the DPF can be written as:

$$Z_k^* + \alpha_k^* + \sum_{(u,v)\in p}\sum_{f\in F^k}(\pi_v^{*f} - \eta_{uv}^{*f}) \leq 0 \quad k \in C, p \in \mathcal{P}_k. \tag{4.18}$$

The feasible solution Υ_D for the DPF is build based on Υ^*, and by using the following variable change: $\bar{\alpha}_k = \alpha_k^* + Z_k^*,\ k \in C$. It follows that for all $k \in C$ and for all $p \in \mathcal{P}_k$:

$$(4.18) \iff \bar{\alpha}_k + \sum_{(u,v)\in p} \sum_{f\in F^k} (\pi_v^{*f} - \eta_{uv}^{*f}) \quad \leq \quad 0$$

Thus, $\Upsilon_D = (\bar{\alpha}^*, \eta^*, \pi^*)$ is a feasible solution for DPF. Correspondingly, $LB = Z_{DRPF} + \sum_{k\in C} Z_k^*$ is the objective value of the DPF defining a valid lower bound for the model PF.

The path formulation can be obtained by applying Dantzig-Wolfe decomposition on the compact formulation proposed in Chapter 2. The pricing problem could generate cyclic graphs (columns). Therefore, these columns will not appear in any feasible solution of the master problem because of precedence constraints. In the path formulation, considered in this thesis, we have added sub-tour elimination cuts in the pricing problem in such a way that it generates only elementary paths. The later formulation cannot be obtained from Dantzig-Wolfe decomposition of the compact model given in Chapter 2. This is due to the fact that sub-tour elimination cuts are needed in the ILP pricing problem.

However, the model PF can be seen as a Dantzig-Wolfe problem reformulation obtained from the compact model studied in Chapter 3 after enhancing the latter with subtour elimination constraints imposed on the flow variables. The lower bound derived in this section corresponds to the Lagrangian bound of this Dantzig-Wolfe problem reformulation (we leave out further details and refer the reader to e.g., [92], for the general theory of Dantzig-Wolfe decomposition).

Corollary 3.3 defined in Chapter 3 indicates that, when implementing a B&P procedure to solve the model PF, we can sidestep branching on the exponential variables λ and apply the regular branching scheme defined only for x, y, z, and w variables. In particular, this means that (apart from the change of the coefficients in the objective function), the pricing problem will not be affected by branching decisions.

4.2 Second extended formulation: the model DW

In this section, we present an alternative extended formulation for the VNFPRP, to which we refer as the model DW (where DW stands for Dantzig-Wolfe). First, we introduce the master problem and then, based on the master problem's dual solution,

we describe the pricing problem and show how to calculate the value of the Lagrangian bound. At the end of this section, a branching scheme is proposed.

In the following, we use the term *path-installation* to indicate a path p with pre-installed VNFs satisfying latency, conflict, and precedence constraints. The master problem aims to choose one path-installation per each commodity k while respecting node and function capacity constraints.

4.2.1 Decision variables

Let us denote by $\mathcal{T}_k$ the set of all path-installations associated with commodity k. To create our master problem, we need three families of variables, as described in Table 4.2.

Variables		Type
τ_p^k	1, if path-installation p associated with commodity k is chosen; 0, otherwise.	Binary
w_u	1, if node u is activated; 0, otherwise.	Binary
z_u^f	number of VNF f installed at node u.	Integer

Table 4.2: Decision variables of the Dantzig-Wolfe formulation

The set $\mathcal{T}_k$ associated with each commodity k containing all feasible path-installations is supposed to be known. Thus, the placement of each VNF for each path-installation at network nodes is uniquely defined. Let us denote by a_u^{fpk} the parameter that is equal to 1 if the VNF f is used at node u for the path-installation p associated with commodity k; and that is equal to 0 otherwise.

$$a_u^{fpk} = \begin{cases} 1, & \text{if VNF } f \text{ is used at node } u \text{ for the path-installation } p \text{ associated} \\ & \text{with commodity } k, \\ 0, & \text{otherwise.} \end{cases}$$

4.2.1.1 ILP formulation

The model DW is then given as:

$$(DW): \quad \min \quad \sum_{u\in N}\sum_{f\in F}\psi_u^f z_u^f + \sum_{u\in N}\psi_u w_u \tag{4.19}$$

$$\sum_{p\in\mathcal{T}_k}\tau_p^k = 1 \qquad k\in C \quad (\alpha_k) \tag{4.20}$$

$$\sum_{f\in F} z_u^f \leq c_u\, w_u \qquad u\in N \quad (\beta_u) \tag{4.21}$$

$$\sum_{k\in C}\sum_{p\in\mathcal{T}_k} a_u^{fpk}\tau_p^k b_k \leq m_f z_u^f \qquad f\in F,\quad u\in N \quad (\gamma_u^f) \tag{4.22}$$

$$\tau_p^k \in \{0,1\} \qquad k\in C,\quad p\in\mathcal{T}_k \tag{4.23}$$

$$w_u \in \{0,1\} \qquad u\in N \quad (\eta_u) \tag{4.24}$$

$$z_u^f \in \mathbb{N} \qquad u\in N,\quad f\in F \tag{4.25}$$

Constraints (4.20) represent the routing constraints ensuring that one path-installation is chosen for each commodity $k \in C$. Inequalities (4.21) represent the nodes capacity constraints. Constraints (4.22) are the VNF-capacity constraints.

The LP-relaxation of this model is obtained by replacing (4.23) by $\tau_p^k \geq 0$, (4.24) by $0 \leq w_u \leq 1$, and (4.25) by $z_u^f \geq 0$

4.2.2 The dual of the master problem

The dual of the master problem is given by the following linear program:

$$\max \quad \sum_{k\in C}\alpha_k - \sum_{u\in N}\eta_u$$

$$\alpha_k - \sum_{u\in N}\sum_{f\in F^k} a_u^{fpk} b_k \gamma_u^f \leq 0 \qquad k\in C,\quad p\in\mathcal{T}_k$$

$$c_u\,\beta_u - \eta_u \leq \psi_u \qquad u\in N$$

$$-\beta_u + m_f\gamma_u^f \leq \psi_u^f \qquad f\in F,\quad u\in N$$

$$\beta,\gamma,\eta \geq 0, \quad \alpha\in\mathbb{R}^{|C|}$$

Inequalities (4.19)-(4.25) constitute the master problem which admits an exponential number of variables. Therefore, a column generation procedure is needed to solve its

continuous relaxation. The master problem is initialized with a subset of columns (called the restricted master problem), and the missing variables necessary to solve its linear relaxation are generated by separating the following dual constraints:

$$\alpha_k - \sum_{u \in N} \sum_{f \in F^k} a_u^{fpk} b_k \gamma_u^f \quad \leq \quad 0 \qquad k \in C, \quad p \in \mathcal{T}_k, \tag{4.26}$$

where we associate variables α and γ to constraints (4.20) and (4.22), respectively. The separation of constraints (4.26) represents the pricing problem. Let (α^*, γ^*) be components of the dual solution of the restricted master problem, the pricing problem consists of finding a commodity k and a path $p \in \mathcal{T}_k$ such that:

$$\alpha_k^* - \sum_{u \in N} \sum_{f \in F^k} a_u^{fpk} b_k \gamma_u^{*f} \quad > \quad 0.$$

4.2.3 The pricing problem

For each commodity k, we have one pricing problem that aims to find a path-installation. The left-hand-side in inequalities (4.26) characterizes the objective function of the pricing problem. The set of variables required in the pricing problem is described in Table 4.3.

Variables		Type
d_u^f	1, if virtual network function f is installed at or before node u; 0, otherwise.	Binary
h_u^f	1, if virtual network function f is installed at node u; 0, otherwise.	Binary
n_{uv}	1, if arc (u, v) belongs to the routing path; 0, otherwise.	Binary

Table 4.3: Decision variables of the pricing problem for the model DW

The MILP formulation of the pricing problem is given as follows:

$$\max \quad \alpha_k^* - \sum_{u \in N} \sum_{f \in F^k} h_u^f b_k \gamma_u^{*f}$$

$$\sum_{(u,v)\in A} n_{uv} - \sum_{(v,u)\in A} n_{vu} = \begin{cases} -1 & \text{if } u = d_k, \\ 1 & \text{if } u = s_k, \\ 0 & \text{otherwise.} \end{cases} \qquad u \in N \tag{4.27a}$$

$$\sum_{(u,v)\in A} n_{uv} l_{uv} \leq l_k \tag{4.27b}$$

$$h_u^f + h_u^g \leq 1 \qquad (f,g) \in \mathcal{A}^k, \quad u \in N \tag{4.27c}$$

$$(n_{uv} - 1) + (d_v^f - d_u^f) \leq h_v^f \qquad f \in F^k, \quad (u,v) \in A \tag{4.27d}$$

$$d_u^g \leq d_u^f \qquad f, g \in F^k : f \prec_k g, u \in N \tag{4.27e}$$

$$h_u^f \leq d_u^f \qquad f \in F^k, \quad u \in N \tag{4.27f}$$

$$h_u^f \leq \sum_{(v,u)\in A} n_{vu} \qquad f \in F^k, \quad u \in N \tag{4.27g}$$

$$\sum_{u\in N} h_u^f \geq 1 \qquad f \in F^k \tag{4.27h}$$

$$d_{s_k}^f = 0 \qquad f \in F^k \tag{4.27i}$$

$$d_{d_k}^f = 1 \qquad f \in F^k \tag{4.27j}$$

$$(d, h, n) \text{ is binary} \tag{4.27k}$$

Constraints (4.27a) are the flow-preservation constraints ensuring that the path goes from the source node s_k to the destination node d_k. Inequalities (4.27b) represent the latency constraints. Inequalities (4.27c) are the anti-affinity constraints which ensure that two VNFs f and g in conflict are not installed at the same node u. Constraints (4.27d), (4.27e), (4.27i) and (4.27j) represent the precedence constraints. Constraints (4.27f) are the linking constraints between variables d and h, the right-hand-site is forced to 1 in order to ensure that if VNF f is used at node u, then it is used at or before u. Inequalities (4.27g) link variables h and n, they guarantee that the routing path enter all nodes at which VNFs are installed. Finally, constraints (4.27h) guarantee that all required VNFs for the current commodity are installed at nodes.

Proposition 4.1 *The binary constraints imposed on the arc variables n in the pricing problem (4.27a)-(4.27k) can be relaxed and replaced by $n_{uv} \geq 0$, for all $(u,v) \in A$.*

Proof. Observe that variables n do not appear in the objective function, and that the location variables h (which basically determine the value of the solution) remain binary.

Therefore, if there exists a feasible solution of the pricing problem with fractional n values, there also exists a latency constrained path (corresponding to binary n values) which satisfies all the constraints (4.27a)-(4.27k). To show the latter result, one has to follow similar arguments as those given in the proof of Proposition 3.2. ■

4.2.4 Lagrangian bound

In this subsection, we provide details on how we calculate a Lagrangian bound for the DW formulation. Let DDW be the dual of the model DW and let DRDW be the restricted master problem of the model DW. The Lagrangian bound computation is made in the same way as for the model PF, i.e., by constructing a feasible solution for the DDW from the optimal solution of the DRDW. In the following, we show how to construct this feasible solution during the column generation procedure. Let $\Upsilon^* = (\alpha^*, \gamma^*)$ be the relevant component of the dual vector obtained by solving DRDW, Z_{DRDW} the associated objective value, and Z_k^* the optimal value of the pricing problem (see previous section) associated with commodity k with respect to Υ^*.

Υ^* is optimal for DRDW but not necessarily feasible for the DDW; this means that there exists at least one constraint (4.26) in DDW which is violated by this solution. For a fixed $k \in C$ we have:

$$Z_k^* = -\alpha_k^* + \min_{p \in \mathcal{T}_k} \{ \sum_{u \in N} \sum_{f \in F^k} a_u^{fpk} b_k \gamma_u^{*f} \}$$

As Z_k^* represents the minimum reduced cost (over all path installations $p \in \mathcal{T}_k$), the dual constraint (4.26) in DDW can be written as follows:

$$Z_k^* \leq -\alpha_k^* + \sum_{u \in N} \sum_{f \in F^k} a_u^{fpk} b_k \gamma_u^{*f} \iff Z_k^* + \alpha_k^* - \sum_{u \in N} \sum_{f \in F^k} a_u^{fpk} b_k \gamma_u^{*f} \leq 0 \qquad (4.28)$$

To construct a feasible solution Υ_D for the DDW, we can use the following transformation: $\bar{\alpha}_k = \alpha_k^* + Z_k^*$, for all $k \in C$. It follows that for all $k \in C$ and for all $p \in \mathcal{T}_k$:

$$(4.28) \iff \bar{\alpha}_k - \sum_{u \in N} \sum_{f \in F^k} a_u^{fpk} b_k \gamma_u^{*f} \leq 0$$

Thus, $\Upsilon_D = (\bar{\alpha}^*, \gamma^*)$ is a feasible solution for DDW and the associated (Lagrangian) bound is calculated as $LB = Z_{DRDW} + \sum_{k \in C} Z_k^*$.

Given that, any value of LB during the column generation procedure represents a lower bound for the relaxed master problem, in our code, at each iteration of the column generation procedure, the value of the LB is saved and the maximum overall its values is compared to the objective value of the restricted master problem, which we call Primal bound in Figures 4.1 and 4.2. The CG procedure is stopped if the difference between both values is smaller than $\varepsilon = 10^{-4}$, i.e., if $|LB - Z_{DRDW}| < \varepsilon$ holds. Figure 4.1 refers to the evolution of LB if the column generation is initialized using an artificial column, and Figure 4.2 illustrates the improvement in the quality of LB that can be achieved if columns building a heuristic solutions are used to initialize the CG procedure. Moreover, both figures indicate that at least 1/3 of CG iterations can be saved and the pricing can be prematurely stopped when $|LB - Z_{DRDW}| < \varepsilon$ stopping condition is met.

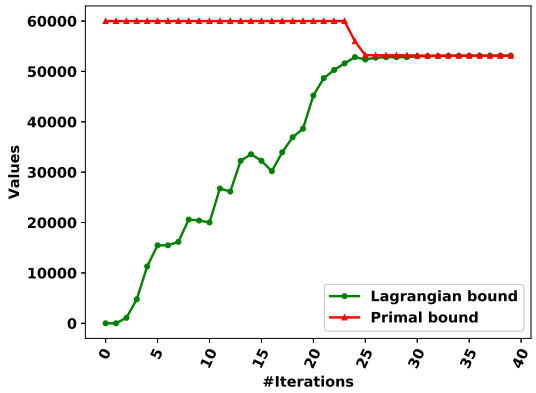


Figure 4.1: Example of the Lagrangian bound evolution during the column generation procedure for the Dantzig-Wolfe formulation on "Pdh_1" instance without heuristic solution and with valid inequalities.

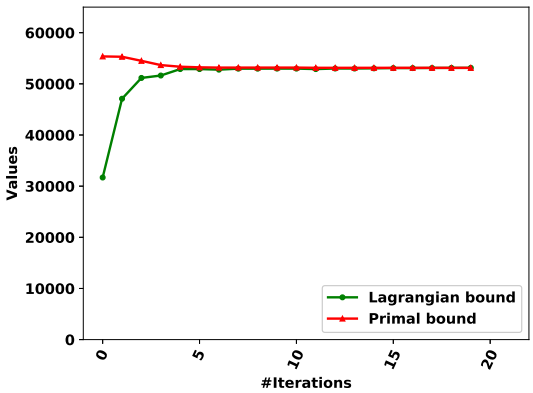


Figure 4.2: Example of the Lagrangian bound evolution during the column generation procedure for the Dantzig-Wolfe formulation on "Pdh_1" instance with heuristic solution and valid inequalities.

4.2.5 Branching on τ variables

The LP-relaxation of the Dantzig-Wolfe formulation solved by column generation procedure is not necessarily integral. Furthermore, applying the Branch-and-Bound algorithm on the restricted master problem with only the generated columns at the root node will not guarantee a feasible solution and so an optimal solution. Moreover, at each branching node, there may exist new columns with a negative reduced cost which

should be added to the master problem. Therefore, in order to find an optimal integer solution, we should generate columns at each branching node.

Various branching schemes, specific (like the one proposed below) or generic (see e.g., [134]), can be used to generate integer solutions using column generation procedure embedded within the Branch-and-Bound algorithm. The resulting algorithm is called *Branch-and-Price*.

In the following, a commodity $k \in C$ is called *fractional* if it admits a fractional τ variable. For a path-installation $p \in \mathcal{T}_k$, we use the notation $u \in p$ to indicate that the path-installation p passes through the node u. Furthermore, the notation $a_u^{fpk} = 1$ is used to indicate that the path-installation p passes through the node u on which the VNF f is installed for commodity k.

Proposition 4.2 *For any given LP-solution of the (restricted) master problem with fractional τ variables, at least one of the following cases is valid for each fractional commodity $k \in C$:*

Case 1. *There exist two nodes $u, v \in N \setminus \{s_k\}$ satisfying:*

$$0 < \sum_{\substack{p \in \mathcal{T}_k \\ u \in p, v \in p}} \tau_p^k < 1 \tag{4.29}$$

Case 2. *There exist a function $f \in F^k$ and a node $u \in N \setminus \{s_k\}$ satisfying:*

$$0 < \sum_{\substack{p \in \mathcal{T}_k \\ a_u^{fpk} = 1}} \tau_p^k < 1 \tag{4.30}$$

Proof. Let us suppose that k is a fractional commodity but neither **Case 1** nor **Case 2** holds. Hence, we have:

$$\sum_{\substack{p \in \mathcal{T}_k \\ u \in p, v \in p}} \tau_p^k \in \{0, 1\} \qquad u, v \in N \tag{4.31}$$

$$\sum_{\substack{p \in \mathcal{T}_k \\ a_u^{fpk} = 1}} \tau_p^k \in \{0, 1\} \qquad u \in N, f \in F^k. \tag{4.32}$$

From (4.32) we can distinguish two cases: (a) there exists a node $u \in N$ and a function $f \in F^k$ such that $\sum_{\substack{p \in \mathcal{T}_k \\ a_u^{fpk} = 1}} \tau_p^k = 1$, or (b) $\sum_{\substack{p \in \mathcal{T}_k \\ a_u^{fpk} = 1}} \tau_p^k = 0$, for each $u \in N$ and $f \in F^k$.

(a) By constraints (4.20), and because k is a fractional commodity, all path-installations of $\mathcal{T}_k$ in the solution are fractional. Let $p_1 \in \mathcal{T}_k$ be a fractional path-installation passing through node u on which VNF f is installed (i.e., $0 < \tau^k_{p_1} < 1$). As $\sum_{\substack{p\in\mathcal{T}_k \\ a_u^{fpk}=1}} \tau^k_p = 1$, there must exist another fractional path-installation $p_2 \in \mathcal{T}_k \setminus \{p_1\}$ passing through node u on which VNF f is installed. Since $p_1 \neq p_2$, we have two cases:

(I) p_1 and p_2 pass through the same nodes but with different function installations, i.e., there exists at least one VNF $g \in F^k \setminus \{f\}$ installed on a different node. Let denote by v (resp. w, $v \neq w$) the node belonging to the path-installation p_1 (resp. p_2) on which g is installed. As the hypothesis (4.32) is valid for any VNF in F^k and any node in $N \setminus \{s_k\}$, $\sum_{\substack{p\in\mathcal{T}_k \\ a_v^{gpk}=1}} \tau^k_p \in \{0,1\}$ must hold. Given that $\tau^k_{p_1} > 0$, then (1) $\sum_{\substack{p\in\mathcal{T}_k \\ a_v^{gpk}=1}} \tau^k_p > 0$. Thus $\sum_{\substack{p\in\mathcal{T}_k \\ a_v^{gpk}=1}} \tau^k_p = 0$ cannot hold. Accordingly, $\sum_{\substack{p\in\mathcal{T}_k \\ a_v^{gpk}=1}} \tau^k_p = 1$. We know that $0 < \tau^k_{p_2} < 1$ and that g is not installed on v for p_2, we will have: (2) $\sum_{\substack{p\in\mathcal{T}_k \\ a_v^{gpk}=1}} \tau^k_p < 1$. Therefore, (1) and (2) contradict hypothesis (4.32).

(I) p_1 and p_2 pass through at least one different node, i.e., there should exist another node $v \neq u$ belonging to p_2 and not to p_1. We notice that VNF f is installed only on node u for both p_1 and p_2, and that another VNF $g \neq f$ can or not be installed on node v belonging to the path-installation p_2. Since p_2 contains u and v and $\tau^k_{p_2} > 0$, this implies: (i) $\sum_{\substack{p\in\mathcal{T}_k \\ u\in p, v\in p}} \tau^k_p > 0$. Moreover, as $\sum_{\substack{p\in\mathcal{T}_k \\ a_u^{fpk}=1}} \tau^k_p = 1$ and $\tau^k_{p_1} > 0$ and we know that path-installation p_1 does not pass through node v, then the value $\tau^k_{p_1}$ can be deleted from the following sum: $\sum_{\substack{p\in\mathcal{T}_k \\ u\in p, v\in p}} \tau^k_p$, which implies that (ii) $\sum_{\substack{p\in\mathcal{T}_k \\ u\in p, v\in p}} \tau^k_p < 1$. Therefore, (i) and (ii) contradict hypothesis (4.31).

(a) Recall that for each commodity k, we assume that $F^k \neq \emptyset$ (otherwise the commodity can be pre-processed and eliminated). This, together with constraints (4.20), contradicts the assumption that for all functions $f \in F^k$, no path-installation $p \in \mathcal{T}_k$ is chosen such $a_u^{fpk} = 1$ (i.e., it contradicts the hypothesis

$$\sum_{\substack{p\in\mathcal{T}_k \\ a_u^{fpk}=1}} \tau_p^k = 0).$$

Therefore, the result holds. ■

`

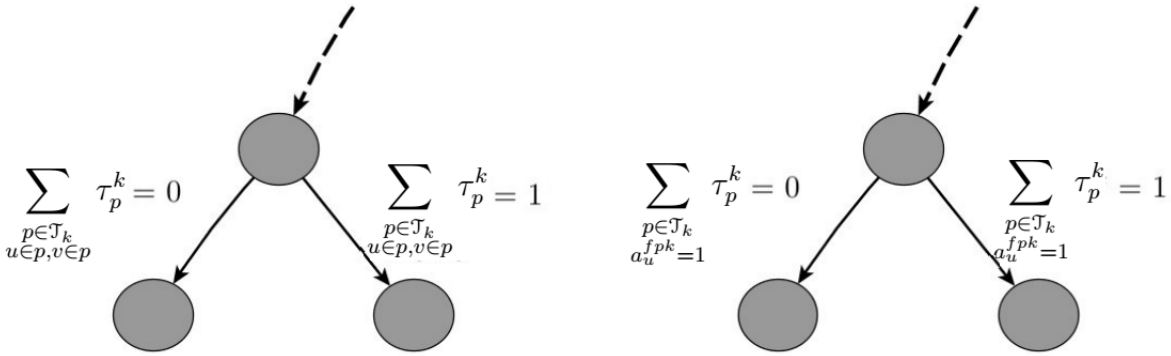


Figure 4.3: Branching scheme for Dantzig-Wolfe formulation.

A branching scheme is said to be *complete*, if it can generate any feasible solution. From Proposition 4.2 we conclude that our branching scheme proposed for the model DW is complete, as at least one of the two cases should hold for any fractional commodity.

4.3 Strengthening inequalities

In this section, we derive several families of valid inequalities that can strengthen the LP-bounds of both proposed extended formulations. We first present inequalities that can be used to directly enhance the model PF. We then present inequalities that are valid for both models and that can be exploited in case a function's capacity is smaller than the respective traffic demand. We close this section by explaining how some of inequalities proposed for the model PF can be used within the Dantzig-Wolfe decomposition to strengthen the model DW.

4.3.1 Valid inequalities for the model PF

Proposition 4.3 *Inequalities* (4.33) *are valid for the VNFPRP:*

$$y_u^{fk} \leq w_u, \quad u \in N, \quad k \in C, \quad f \in F^k. \tag{4.33}$$

Proof. For a given commodity $k \in C$, if the VNF $f \in F^k$ is installed at node u, then node u should be activated. ■

Proposition 4.4 *Inequalities* (4.34) *are valid for the VNFPRP:*

$$y_u^{fk} + y_u^{gk} \leq w_u, \quad u \in N, \quad k \in C, \quad (f,g) \in \mathcal{A}^k. \tag{4.34}$$

Proof. If two VNFs f and g are in conflict for a commodity $k \in C$, then, both functions cannot be installed at the same node u, if u is activated. ■

Let $D_k = (F^k, E)$ be the conflict graph associated with commodity k, $k \in C$, where nodes in D_k represent the VNFs $f \in F^k$. An edge $e \in E$ between two nodes f and g in D_k represents the fact that f and g are in conflict. Let $\mathcal{D}_k$ denote the set of all maximal cliques in D_k and let $\omega(D_k)$ be the *clique-number* (i.e., the size of the maximum clique) in the conflict graph.

Proposition 4.5 *Inequalities* (4.35) *are valid for the VNFPRP:*

$$\sum_{f \in Q} y_u^{fk} \leq 1, \quad u \in N, \quad k \in C, \quad Q \in \mathcal{D}_k. \tag{4.35}$$

Proof. For a given commodity $k \in C$, nodes in Q represent the set of VNFs in conflict, i.e., they cannot be installed at the same node. Therefore, only one VNF in Q can be installed at node u as otherwise the conflict constraints (4.5) are violated. ■

Linear inequalities (4.34) and (4.35) can be combined and generalized for each clique Q in D_k.

Proposition 4.6 *Inequalities* (4.36) *are valid for the VNFPRP, and they also dominate inequalities* (4.35).

$$\sum_{f \in Q} y_u^{fk} \leq w_u, \quad u \in N, \quad k \in C, \quad Q \in \mathcal{D}_k. \tag{4.36}$$

Proof. Trivial. ■

Given a commodity $k \in C$ and a node $u \in N$, if there is a unique path p going from s_k to u in G, let A_p be the arcs belonging to the path p.

Proposition 4.7 *For a given commodity $k \in C$, and a node $u \in N$, if there exists a unique path from s_k to u in G, then inequalities (4.37) are valid for the VNFPRP:*

$$\sum_{f \in Q} x_u^{fk} \leq |A_p|, \quad u \in N, \quad k \in C, \quad Q \in \mathcal{D}_k \ : \ |A_p| < |Q|. \tag{4.37}$$

Proof. The proof is given for a fixed commodity k and is valid for all commodities. Let u be the node for which we have a unique path p going from s_k to u, and let Q be a clique in the graph D_k, such that $|A_p| < Q$. The number of VNFs in conflict installed at or before node u should be less than or equal the number of arcs in the path p; otherwise, two or more functions in conflict need to be installed at the same node, which leads to an infeasible solution. ■

Proposition 4.8 *Inequalities (4.38) are valid for the VNFPRP.*

$$\sum_{u \in N} w_u \geq \max\{1, \max_{k \in C} \omega(D_k)\}. \tag{4.38}$$

Proof. Recall that each commodity requires at least one VNF. As all required VNFs should be installed at graph nodes, at least one node in the graph must be activated. Furthermore, if there is a conflict between VNFs associated with one commodity k, then the number of activated nodes should be at least equal to the maximum number of VNFs in conflict, which is the clique number of D_k. ■

Proposition 4.9 *Inequalities (4.39) are valid for the VNFPRP.*

$$\sum_{u \in N} \sum_{f \in Q} y_u^{fk} \geq |Q|, \quad k \in C, \quad Q \in \mathcal{D}_k. \tag{4.39}$$

Proof. From inequalities (4.10), all VNFs required for each commodity $k \in C$ should be installed at graph nodes, thus the number of nodes necessary to install VNFs in conflict should be at least equal to the number of VNFs in conflict, which is equal to the size of the cliques from $\mathcal{D}_k, k \in C$. ■

Let $C^u \subseteq C$ be a subset of commodities for which there exists at least one latency-constrained path visiting node u, (i.e., if $k \notin C^u$, this means that all paths associated with k do not enter the node u). Let N^k be the set of nodes belonging to at least one latency-constrained path associated with commodity $k \in C$ (this can be checked in polynomial time using a min-cost flow algorithm for example).

Proposition 4.10 *The following inequalities are valid for the VNFPRP.*

$$\sum_{k \in C \setminus C^u} \sum_{f \in F^k} y_u^{fk} = 0, \quad u \in N, \tag{4.40}$$

$$\sum_{k \in C \setminus C^u} \sum_{f \in F^k} x_u^{fk} = 0, \quad u \in N. \tag{4.41}$$

Moreover, inequalities (4.42) are valid and dominate inequalities (4.10);

$$\sum_{u \in N^k} y_u^{fk} \geq 1, \quad k \in C, \quad f \in F^k, \tag{4.42}$$

$$\sum_{f \in F} z_u^f = 0, \quad u \in N \setminus \{\cup_{k \in C} N^k\}, \tag{4.43}$$

$$\sum_{u \in N \setminus \{\cup_{k \in C} N^k\}} w_u = 0. \tag{4.44}$$

Proof. If there exists a node $u \in N$ which is not visited by any commodity $k \in C$, then no function $f \in F^k$ can be installed at u (4.40), thus at or before u (4.41). Therefore, VNFs associated with commodity $k \in C$ can be installed only at nodes in N^k (4.42). In consequence, u cannot be activated (4.44), so the number of VNFs installed on it is equal to zero (4.43). ■

Proposition 4.11 *Inequalities (4.45) are valid for the VNFPRP:*

$$z_u^f \geq \lceil \frac{b_k}{m_f} \rceil y_u^{fk}, \quad k \in C, \quad f \in F^k, \quad u \in N. \tag{4.45}$$

Proof. The number of VNFs installed at node u is at least equal to the number of VNFs needed to handle one commodity $k \in C$. ■

Proposition 4.12 *Inequalities (4.46) are valid for the VNFPRP:*

$$x_u^{fk} \leq 1 - y_{d_k}^{fk}, \qquad k \in C, \quad u \in \Gamma^-(d_k), \quad f \in F^k, \tag{4.46}$$

where $\Gamma^-(d_k)$ denotes all incoming neighbors of d_k.

Proof. If a VNF f associated with commodity $k \in C$ is installed at the destination node, then, this function cannot be installed at or before any predecessor of d_k. ■

Inequalities (4.46) can be generalized by the following inequalities (4.47). Let us consider a cut separating the source nodes s_k from the destination node d_k in G. We denote by N_{s_k} (*resp.* N_{d_k}) the component containing s_k (*resp.* d_k) in G. Suppose that arcs in the cut go from N_{s_k} to N_{d_k}. The following inequalities are valid for the VNFPRP.

$$x_u^{fk} \leq 1 - \sum_{v \in N_{d_k}} y_v^{fk}, \qquad k \in C, \quad u \in N_{s_k}, \quad f \in F^k \tag{4.47}$$

Proof. For a given commodity $k \in C$, if a VNF f is installed at nodes in N_{d_k} and in addition we have no arc going from N_{d_k} to N_{s_k}, then f is not installed at or before nodes in N_{s_k}. ■

Proposition 4.13 *Inequalities* (4.46) *can be generalized by inequalities* (4.48) *which are valid for the VNFPRP. Let* $\mathcal{S}_k$ *be the node separator disconnecting* N_{s_k} *from* N_{d_k},

$$x_u^{fk} \leq 1 - \sum_{v \in N_{d_k}} y_v^{fk} + \sum_{v \in \mathcal{S}_k} x_v^{fk}, \qquad k \in C, \quad u \in N_{s_k}, \quad f \in F^k \tag{4.48}$$

Proof. For a given commodity $k \in C$, if the VNF f is installed at nodes in N_{d_k} and it is not installed at or before nodes in the separator $\mathcal{S}_k$, then, the chosen path does not go from N_{d_k} to N_{s_k}; This means that the function f cannot be installed at or before nodes of N_{s_k}. Note that arcs from N_{d_k} to N_{s_k} may exist. ■

4.3.2 Strengthening inequalities for both models

Besides inequalities (4.38), (4.43) and (4.44) which are also valid for the model DW, in the following we propose additional inequalities that involve only z and w variables, and are therefore valid for both formulations studied in this chapter. With valid inequalities given in Proposition 4.11 we address the setting in which the capacity of a function is not sufficient to handle the full demand of a given commodity (i.e., multiple copies of the same function need to be installed). Proposition 4.14 provides further generalizations of this setting.

Proposition 4.14 *Node capacity constraints (4.3) do not define facets of the polyhedron of the VNFPRP if there exists a node u, such that $c_u > \sum_{k\in C}\sum_{f\in F^k}\lceil\frac{b_k}{m_f}\rceil$.*

1) Therefore, inequalities (4.49) are valid for the VNFPRP and dominate inequalities (4.3).

$$\sum_{f\in F} z_u^f \leq \sum_{k\in C}\sum_{f\in F^k}\lceil\frac{b_k}{m_f}\rceil w_u, \quad u \in N. \tag{4.49}$$

2) Moreover, if $|C^u| < |C|$, then the linear inequalities (4.50) dominate (4.49):

$$\sum_{f\in F} z_u^f \leq \sum_{k\in C^u}\sum_{f\in F^k}\lceil\frac{b_k}{m_f}\rceil w_u, \quad u \in N. \tag{4.50}$$

3) In addition if there exists a conflict between functions in F^k for a given commodity $k \in C$, with $m_{f_1} \leq m_{f_2} \leq \cdots \leq m_{f_{|Q|}}$, $Q \in \mathcal{D}_k$ and $c_u \geq \sum_{k\in C}\sum_{f\in F^k}\lceil\frac{b_k}{m_f}\rceil$, then inequalities (4.50) are dominated by the following inequalities.

$$\sum_{f\in F} z_u^f \leq \sum_{k\in C}\sum_{Q\in\mathcal{D}_k}[(\sum_{f\in F^k}\lceil\frac{b_k}{m_f}\rceil) - \sum_{\substack{i=2\\ f_i\in Q}}^{|Q|}\lceil\frac{b_k}{m_{f_i}}\rceil]w_u, \quad u \in N. \tag{4.51}$$

Proof.

1) If there exists a node u having enough capacity to install VNFs required for all commodities in C, then the number of functions needed to treat all commodities bandwidth is bounded by $\sum_{k\in C}\sum_{f\in F^k}\lceil\frac{b_k}{m_f}\rceil$, which represents the maximum number of VNFs necessary to handle all demands.

2) Only VNFs associated with commodities having at least one path passing through a node u can be installed at node u.

3) The number of VNFs installed at node u when the conflict constraints are considered is bounded by the maximum number of copies needed to install the VNFs with the smallest capacity for each commodity (i.e., in the worst case we will keep the VNFs with the maximum instantiation installed at node u and install other VNFs at the other nodes). ■

4.3.3 Strengthening the model DW

Valid inequalities proposed for the model PF can be "translated" into valid inequalities for the model DW. In this subsection we illustrate how this can be done for inequalities (4.33), (4.34) and (4.45). The remaining inequalities can be translated accordingly. In order to add (4.33), (4.34) and (4.45) to the master problem of the model DW, we rewrite them using parameters a as (4.52), (4.53) and (4.54), respectively:

$$\sum_{p\in\mathcal{T}_k} a_u^{fpk}\tau_p^k \leq w_u, \qquad k\in C, \quad u\in N, \quad f\in F^k \tag{4.52}$$

$$\sum_{p\in\mathcal{T}_k} (a_u^{fpk}+a_u^{gpk})\tau_p^k \leq w_u, \qquad k\in C, \quad u\in N, \quad (f,g)\in\mathcal{A}^k \tag{4.53}$$

$$z_u^f \geq \lceil\frac{b_k}{m_f}\rceil \sum_{p\in\mathcal{T}_k} a_u^{fpk}\tau_p^k, \quad k\in C, \qquad f\in F^k, \quad u\in N \tag{4.54}$$

Adding these valid inequalities generates new (non-negative) dual variables in the dual of the master program, that we denote by δ, η and φ, respectively. Thus, dual constraints (4.26) need to be replaced by inequalities (4.55)

$$\alpha_k - \sum_{u\in N}\left[\sum_{f\in F^k}(b_k\gamma_u^f + \lceil\frac{b_k}{m_f}\rceil\varphi_u^{fk}+\delta_u^{fk})a_u^{fpk} + \sum_{(f,g)\in\mathcal{A}^k}\eta_u^{fgk}(a_u^{fpk}+a_u^{gpk})\right] \leq 0, \quad k\in C, p\in\mathcal{P}_k \tag{4.55}$$

4.4 Branch-and-Price algorithms

In this section, we present two B&P algorithms implemented for the models PF and DW, respectively.

4.4.1 Generic column generation framework

Initialization The restricted master problem of both models is initialized by a subset of columns building a heuristic solution which is obtained in the initialization phase of the algorithm (see Section 4.4.4). If no solution has been found during a time-limit, the CG framework is initialized with an artificial column whose cost is set to a very large number.

Bounding At each iteration of the column generation procedure, the restricted master problem is solved, and a dual solution is provided. Accordingly, the objective function of the pricing problem for each commodity k is updated, and the pricing problem is solved. Depending on the pricing strategy (see Section 4.4.3) multiple columns per commodity having negative reduced costs (or at most one) are added to the restricted master problem. During this process, to reduce the number of CG iterations, we keep track of the Lagrangian bound and compare it to the objective value of the current restricted master problem. If the difference between the two values is smaller than ε, the column generation procedure is stopped and we resort to branching.

4.4.2 Branching

At the end of the column generation phase, the integrality of the solution of the relaxed master problem is verified. If the current solution is not integer, we branch on the most fractional variable, applying the BFS (Breadth-First Search) based branching strategy. Specifically, we explore all the nodes of the same level in the branching tree before moving to the next level. In our implementation the algorithm explores all nodes admitting a fractional feasible solution of the same level by applying the respective branching scheme described below. For each branching node with a fractional solution, two children nodes are created and saved in a queue. The nodes in the queue are explored using the FIFO (First In First Out) method. A global lower bound is calculated at each level. In our preliminary experiments, we also tried the diving strategy as an alternative to the BFS-based branching. Whereas diving is very useful when searching for feasible solutions (see e.g., [57,60,79]), in our case this strategy did not prove useful, because a high-quality feasible solution is used to initialize the CG procedure (see above).

Branching scheme for the model PF In Corollary 3.3 we showed that the binary constraints on λ variables in the PF model can be relaxed to $\lambda \geq 0$. Hence, in our BP implementation of the PF model, we branch only on the (x, y, z, w) variables.

Different branching schemes have been tested for the model PF; The one outperforming all others is to branch first on the most fractional w variables (by imposing either $w \geq 1$ or $w \leq 0$), secondly on z by setting either $z \geq \lceil z^* \rceil$ or $z \leq \lfloor z^* \rfloor$, thirdly on y which are forced to be $y \geq 1$ or $y \leq 0$ and finally on x variables using $x \geq 1$ or $x \leq 0$.

Branching scheme for the model DW In our branching scheme for the model DW we start branching on z variables by setting $z \geq \lceil z \rceil$, or $z \leq \lfloor z \rfloor$ and then on w variables by setting $w \leq 0$, or $w \geq 1$. When z and w variables are integers, we continue by branching on the τ variables. Given that the path-installation variables are generated as and when they are needed, we follow the specific branching scheme for τ variables proposed in Proposition 4.2. Specifically, if we find a pair of two distinct nodes $u, v \in N$ such that $\sum_{p \in \mathcal{T}_k : u \in p, v \in p} \tau_p^k$ is fractional we create two branches by imposing constraints that limit the respective sum to 0, or 1, respectively. If none such pair can be found, we search for a node $u \in N$ and a function $\bar{f} \in F^k$ such that $\sum_{p \in \mathcal{T}_k : a_u^{fpk} = 1} \tau_p^k$ is fractional, and create two branches correspondingly.

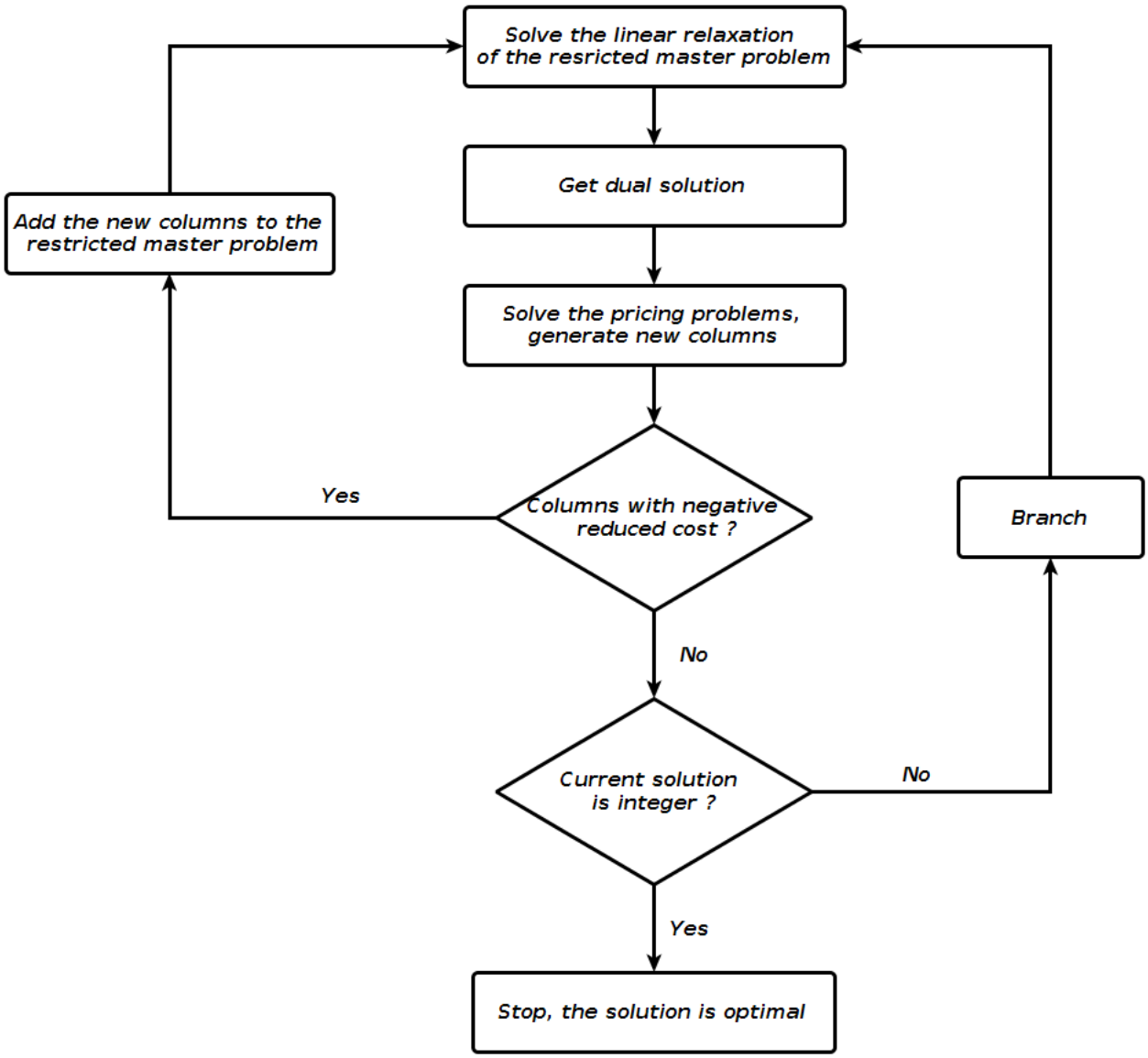


Figure 4.4: Branch-and-Price algorithm.

4.4.3 Pricing strategy

In the sequel, we first propose three different pricing strategies for the model PF. These strategies are later computationally evaluated in Section 4.6. For the model DW, the underlying pricing problem exhibits a rich combinatorial structure. We therefore employ an off-the-shelf MIP solver to price in the columns.

4.4.3.1 Pricing the columns for the model PF

We consider three methods for solving the pricing problem for the formulation PF: 1) We utilize dynamic programming; 2) Based on Yen's algorithm [138], we derive a reduced cost method, and use it in two different ways; and 3) We model the pricing problem as an ILP that we solve using an off-the-shelf solver.

Dynamic programming Let $k \in C$ be a given commodity and let, for each arc $(u, v) \in A$, $\tilde{c}_{uv}$ given by (4.17) be the corresponding arc cost. We denote by $F_v(S, l'_v)$ the minimal cost of the partial path going from source node s_k to node v visiting all nodes in S exactly once and ready to leave node v with latency value equal to l'_v. The following equations (see also [115]) illustrate the recursive function used for the pricing problem associated with commodity k:

$$DP: \begin{cases} F_{s_k}(\emptyset, 0) &= 0 \\ F_v(S, l'_v) &= \min\limits_{v \in \Gamma^+(u)} \{F_u(S - \{v\}, l'_u) + \tilde{c}_{uv} \mid \sum\limits_{(u',v') \in A_S} l_{u'v'} \leq l_k\} \end{cases}$$

where S is the set of nodes in the path, A_S is the set of arcs connecting nodes in S, l'_u (resp. l'_v) represents the sum of the latency of arcs in A_S from source node s_k until node u (resp. v). We utilize the dynamic programming algorithm proposed in [25] (cf.

Sections 2.2 and 2.3).

Algorithm 6: Labeling algorithm based on dynamic programming

- Inspired from Dijkstra algorithm proposed for the SPP.;
- The algorithm starts from the source node.;

for *each iteration, i.e., going from node i to node j using arc (i, j)* **do**

- A label l_j is created for node j based on l_i, in order to store information on the resources and the predecessor arc (i, j).;
- Only feasible labels are created, i.e., satisfying all resource constraints.;

for *each node* **do**

- Only the non-dominated labels (i.e., pareto-optimal) are kept according to the following dominance rule: ;

if *Both are at the same node* **then**

if **for** *each feasible extension of l_2, there is also* **do**

- *A feasible extension of l_1 where $res_r^{l_1} \leq res_r^{l_2}$ with $r \in R$, for exactly the same resources.*

end

then

- l_1 dominates l_2.;

end

end

end

end

- The algorithm stops when there are no more unprocessed labels for all nodes.;
- It then checks whether the destination node could be reached.;
- If yes, return all shortest paths from s_k to d_k satisfying the resource constraints.

Remark: We must have at least one non-decreasing resources (without negative cycles) to use this algorithm.

Reduced cost method For each commodity $k \in C$, we run Yen's algorithm [138] in order to generate all latency-constrained elementary paths between s_k and d_k. We keep all these columns in a *column pool*, denoted by $\mathbb{P}_k$. First, the restricted master problem is initialized by a subset of columns from $\mathbb{P}_k$, which are then added to $\mathcal{P}_k$ and removed from $\mathbb{P}_k$. At each iteration of the CG procedure, the restricted master problem is solved, and a dual solution is generated. Based on this solution, the reduced cost $RC(p)$ of each path $p \in \mathbb{P}_k$ is calculated. If the path $p \in \mathbb{P}_k$ has a negative reduced cost, then p can be added to the restricted master (i.e., to the set $\mathcal{P}_k$) and deleted from

the set $\mathbb{P}_k$. However, multiple columns with negative reduced cost can be added for each commodity. We consider two settings: In the first one (that we refer to as `Red Cost 1`), we add at most 10 columns with negative reduced cost per commodity; In the second one (that we refer to as `Red Cost 2`), all columns from $\mathbb{P}_k$ with negative reduced cost are added in each iteration. We tested both approaches, and the results are shown in Section 4.6.

At the end of the column generation phase, and before resorting to branching, we also try to fix some columns based on their reduced cost. Using an upper bound UB (obtained from the incumbent solution), and the value Z of the current restricted master problem, if there exists a commodity $k \in C$ and a path $p \in \mathbb{P}_k$ such that $RC(p) + Z > UB$, this means that the variable λ_p^k can be fixed to zero. Therefore, in

this case the path p is deleted from $\mathbb{P}_k$.

Algorithm 7: Reduced cost method

```
for commodity k ∈ C do
    - Get the set P_k of all latency-constrained elementary paths using Yen's
      algorithm (Columns);
end
- Initialize the restricted master problem with a subset of columns;
- Remove the added columns associated with each commodity k ∈ C from P_k;
for each iteration of the column generation procedure do
    - Solve the restricted master problem;
    - Get the dual solution;
    for commodity k ∈ C do
        for each path (column) p, p ∈ P_k do
            Calculate the reduced cost of p, RC(p);
            if RC(p) is negative then
                Add the column p to the restricted master problem;
                P_k = P_k ∪ {p};
                P_k = P_k \ {p};
            end
        end
    end
end
- Get LB and UB;
for commodity k ∈ C such that P_k ≠ ∅ do
    for path p ∈ P_k do
        Calculate the reduced cost of p, RC(p);
        if LB + RC(p) > UB then
            P_k = P_k \ {p};
        end
    end
end
```

Solving the pricing problem as an ILP The pricing problem can also be modeled as a commodity flow ILP problem with latency constraints and subtour elimination cuts. We first solve the latency-constrained commodity flow problem, called *Relaxed Pricing Problem* (RPP) presented below. We then generate sub-tour elimination constraints on the fly (if needed) during the column generation procedure. To define the

relaxed pricing problem we introduce binary variables t_{uv} which are set to one if arc $(u, v) \in A$ belongs to the path associated with the current commodity; and to 0, otherwise. The RPP associated with commodity $k \in C$ is equivalent to the following integer linear program, in which constraints (4.56) ensure that the flow goes from the source node s_k to the destination node d_k, whereas (4.57) represents the latency constraint:

$$(RPP) : \min \quad \sum_{(u,v)\in A} \tilde{c}_{uv} t_{uv}$$

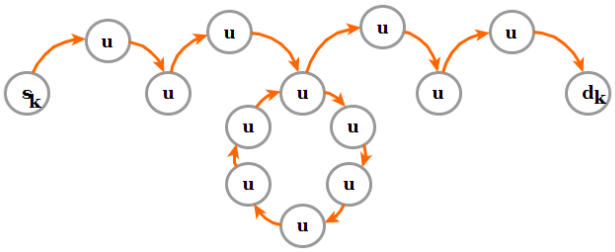
$$\sum_{(u,v)\in A} t_{uv} - \sum_{(v,u)\in A} t_{vu} = \begin{cases} -1 & \text{if } u = d_k, \\ 1 & \text{if } u = s_k, \\ 0 & \text{otherwise.} \end{cases} \qquad u \in N \tag{4.56}$$

$$\sum_{(u,v)\in A} t_{uv} l_{uv} \leq l_k \tag{4.57}$$

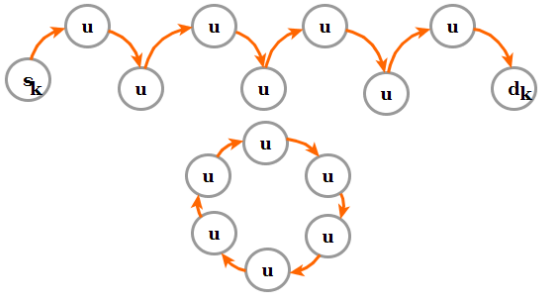
$$t_{uv} \in \{0, 1\} \qquad (u, v) \in A$$

Let $\mathcal{C} = (N_{\mathcal{C}}, A_{\mathcal{C}})$ be the support graph obtained from the binary solution of the RPP model. Since negative arc costs are allowed, $\mathcal{C}$ may contain subtours. Subtour elimination cuts are added at each iteration of the column generation procedure if $\mathcal{C}$ represents one of the two configurations illustrated in Figure 4.5. The connected path shown in Figure 4.5(a) is eliminated by adding the inequality: $\sum_{v\in\Gamma^+(u)} t_{uv} \leq 1$, which aims to force the path to pass only one time through a node u, with degree$(u) \geq 3$. Disconnected paths like the one shown in Figure 4.5(b) are eliminated by the inequality $\sum_{(u,v)\in A_{\mathcal{C}}} t_{uv} \leq |N_{\mathcal{C}}| - 1$.

We point out that the efficiency of this procedure highly depends on the underlying cost function $\tilde{c}$. As we will see in Section 4.6, when there are very few negative cycles in the graph, this procedure may outperform the other alternatives, including dynamic programming.



(a) Connected path



(b) Disconnected path

Figure 4.5: Example of two solutions generated by the relaxed pricing problem.

Algorithm 8: Implementation of pricing problem

- Solve the restricted master problem.
- Get the current dual solution.

for *commodity* $k \in C$ **do**
  { Solve the relaxed pricing problem}
  if *The resulting graph* $\mathcal{C}$ *is not an elementary path* **then**
    if $\mathcal{C}$ *is not connected* **then**
      - Delete the connected component that contains source node.
      for *each connected component* c **do**
        - Add cut $\sum_{(u,v)\in c} t_{uv} \leq |c| - 1$
      end
    end
    else if *There exist a node* u, $\delta(u) \geq 3$ **then**
      - Add cut $\sum_{v\in\Gamma^+(u)} t_{uv} \leq 1$
    end
  end
end

4.4.3.2 Pricing the columns for the DW model

In order to price the columns associated with the DW formulation, the MIP model presented in Section 4.2.3, is solved for each commodity $k \in C$, using an off-the-shelf solver. Thus, at each iteration of the column generation procedure at most one column per commodity with negative reduced cost is generated by the pricing problems. As in the case of the model PF, the column generation phase terminates when no more columns with a negative reduced cost are found, or when the gap between the current value of the restricted master problem and the Lagrangian bound is smaller than ε.

4.4.4 Heuristics

Before entering the B&P phase, we generate a heuristic solution which provides a high-quality upper bound and a promising set of columns that we use to initialize the CG procedure. For the model PF, we employ the MIP-based heuristic presented in Chapter 3. The heuristic solves a compact model derived from the formulation PF in which only a small subset of latency-constrained paths is considered. To obtain this subset, we run Yen's algorithm [138] which provides κ-elementary paths between two nodes, sorted from the shortest to the longest one. The number of generated paths per commodity is capped by κ, and we let $\kappa \in \{10, 15, 20, \ldots, 50\}$. As soon as the underlying MIP-based heuristic finds a feasible solution for a fixed κ value, we stop. However, for some instances, even for $\kappa = 50$ we fail to find a feasible solution.
For the model DW, we start with an artificial column, price-in the columns with negative reduced cost, and then convert the obtained linear program into a MIP.
In both cases, there is a time limit after which this MIP-based heuristic initialization is aborted.

4.5 Comparing the LP-relaxations

In this section, we compare the LP-relaxation values of the path formulation (PF) and Dantzig-Wolfe (DW) formulation. First, a graphic that shows that the LP-relaxation bounds provided by the DW formulation are better than those given by the PF is presented, followed by the LP-relaxation comparison proof.

Proposition 4.15 *We have $Z_{PF} \leq Z_{DW}$, where Z_{PF} (resp. Z_{DW}) is the LP-relaxation*

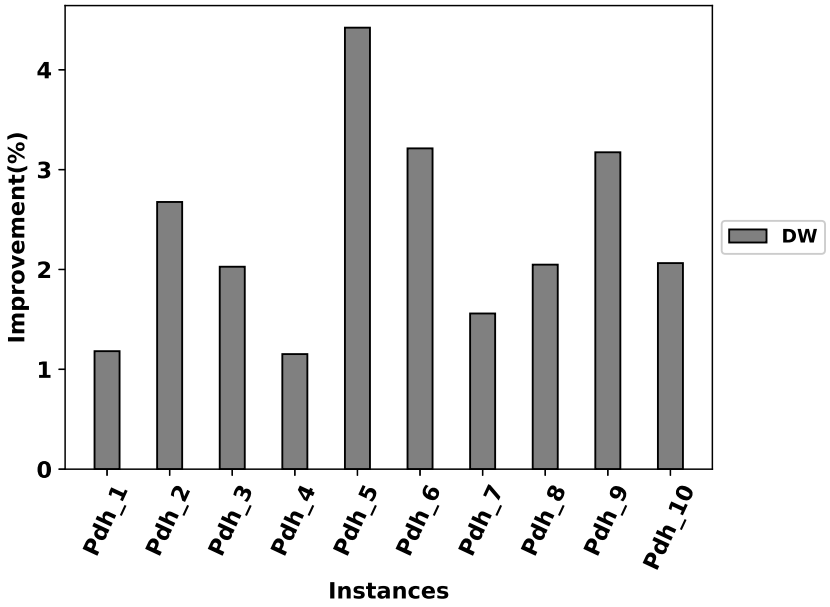


Figure 4.6: Relative improvement of LP-relaxation bounds of the path formulation by using Dantzig-Wolfe formulation on "Pdh" instances.

value of the path formulation (resp. the Dantzig-Wolfe formulation), and there exist instances for which the strict inequality holds.

Proof. We prove that from any feasible solution of DW, we can construct a feasible solution for PF. Let $X_{DW} = (\tau_{DW}, z_{DW}, w_{DW})$ be the feasible solution of the relaxed DW associated with Z_{DW}. From the solution X_{DW}, a feasible solution $X_{PF} = (\lambda_{PF}, x_{PF}, y_{PF}, z_{PF}, w_{PF})$ for PF is built as follows :

- Variables τ_{DW} in Dantzig-Wolfe formulation represent the path-installation chosen for each commodity $k \in C$. From τ_{DW} variables, the information concerning the arcs belonging to each path and the associated VNFs installation and their order are extracted. Therefore, from the values of τ_{DW} in the solution X_{DW}, the value of λ_{PF} variables in path formulation can be obtained. Furthermore, the set of arcs belonging to each path are provided.
- Moreover, in order to get the value of variables x_{PF} and y_{PF} in the path formulation we set $x_{PF} = \sum_{p \in \mathcal{T}_k} \tau_{DW} d_{DW}$ and $y_{PF} = \sum_{p \in \mathcal{T}_k} \tau_{DW} h_{DW}$, with $d_{DW}, h_{DW} \in \{0, 1\}$ are parameters extracted from the variables τ_{DW} (obtained from the pricing problem associated with commodity k in DW formulation).

- Finally, Variables z_{DW} and w_{DW} are exactly the same for both formulations, therefore, their values are obtained by setting $z_{PF} = z_{DW}$ and $w_{PF} = w_{DW}$

Now, we show that the solution X_{PF} built from X_{DW} is feasible for the path formulation, i.e., satisfies all the constraints in the PF. From the DW formulation we can see clearly that path constraints (4.2), node-capacity constraints (4.3) and VNF-capacity constraints (4.4) are satisfied; otherwise d_{DW} is not feasible for DW. Furthermore, each path-installation in DW formulation represents an $s_k - d_k$ elementary path satisfying flow, latency, conflict, installation and precedence constraints. Conflict constraints (4.5) are satisfied by construction, we know from path constraints (4.20) that the sum of τ_{DW} variables is equal to 1, as the parameter $h_{DW} \in \{0, 1\}$, the conflict constraints are satisfied in DW formulation, we conclude that variables y_{PF} satisfy the conflict constraints in the PF. Similarly, as the parameter $d_{DW} \in \{0, 1\}$ and from (4.20), we can see clearly that precedence constraints (4.6), (4.7) and (4.8) are satisfied. Each path-installation passes through the VNFs installations, which means that constraints (4.9) are satisfied. Each path-installation guarantees that all VNFs associated with a given commodity are installed at nodes, accordingly, and by transformation, y_{PF} variables in the path formulation satisfy the installation constraints (4.10). Therefore, from any solution of the Dantzig-Wolfe formulation, a feasible solution for the path formulation can be derived with $Z_{PF} \leq Z_{DW}$. We refer the reader to our computational study, where we report on the instances from which the LP-bound Z_{DW} is strictly stronger than Z_{PF}. ■

4.6 Computational results

In this section we analyze the scalability and the efficiency of the two proposed B&P algorithms and show the benefits of the valid inequalities defined in Section 4.3. The B&P algorithms are compared to two other exact methods using the commercial solver CPLEX: the first one is a compact MIP formulation (denoted by `C`) proposed in Chapter 2 and the second one is the Automatic Benders approach by CPLEX [26] applied to the model `C` in which a family of flow variables is linearly relaxed. Our experiments are designed so as to evaluate the effectiveness and the performance of the proposed extended formulations in terms of CPU time, quality of bounds and final gaps. Eventually we also measure the advantage of the proposed valid inequalities in improving the LP-bounds and reducing the final gaps.

All the experiments described in this section were made using a computer with Intel(R)Xeon(R) CPU E5-2650 v2 processor clocked at 2.60GHz, 32 cores, 2 threads per

core and 252GB RAM, under Linux operating system. All methods are implemented using the Python API for CPLEX, which is run in single-thread mode with a default memory limited to 20GB. All CPLEX parameters were set to their default values. A default time limit of one hour is set for each tested instance. For the initialization heuristic used within the B&P algorithm (cf. Section 4.4.4), the time limit is fixed to 900 seconds.

The following settings represent all tested methods in our computational experiments:

- `DP`: The model PF (introduced in Section 4.1) with the dynamic programming algorithm proposed in Section 4.4.3.1 as pricing method;
- `Red Cost 1`: The model PF with the reduced-cost-based pricing proposed in Section 4.4.3.1, in which a subset of up to 10 paths per commodity with negative reduced cost is added at each iteration of the CG procedure;
- `Red Cost 2`: The model PF with the reduced-cost-based pricing proposed in Section 4.4.3.1, in which all paths with negative reduced cost are added at each iteration of the CG procedure;
- `ILP`: The model PF with the pricing method based on solving the ILP given in Section 4.4.3.1;
- `PF`: The model PF with the best-performing pricing method;
- `PF+VI`: The setting `PF` in which Valid Inequalities (4.33), (4.34), (4.37) − (4.46) and (4.49)−(4.51) presented in Section 4.3 are additionally used to initialize the model;
- `DW`: The model DW presented in Section 4.2;
- `DW+VI`: The setting `DW` in which Valid Inequalities (4.38), (4.43), (4.44), (4.49)−(4.51) and (4.52) − (4.54) from Section 4.3 are added to the model;
- `C`: The compact MIP formulation proposed in Chapter 2;
- `AB`: The Automatic Benders decomposition available in Cplex [26], applied to the setting `C` in which binary flow variables are linearly relaxed.

Benchmark instances

In order to perform our experiments, we have generated a set of instances derived from the SNDlib library [125] of telecommunication networks. The instances creation is detailed in Chapter 3. The set of instance-type used to test our algorithms is defined as follows: {"Abilene", "Atlanta", "Di-yuan", "France", "Geant", "Newyork", "Nobel-eu", "Nobel-germany", "Nobel-us", "Pdh", "Polska", "Ta1"}.

4.6.1 Obtained results

In the sequel we summarize the major results obtained by our computational study. Tables with more detailed information provided per each instance can be found in Section 4.6.2. This subsection is divided into two parts. The first part is devoted to comparing the LP-relaxation bounds generated by applying the column generation algorithm to the models PF and DW, respectively. We start by comparing four different pricing methods proposed for the model PF (namely, `DP`, `ILP`, `Red Cost 1`, `Red Cost 2`) in terms of CPU time, number of added columns and generated iterations. Next, after determining the best configuration (i.e., the best pricing method) for the model PF, we focus on the improvement of the LP-gap, with respect to the LP-relaxation bounds provided by the compact formulation (`C`). We compare the two extended formulations, with and without adding valid inequalities.

In the second part of this subsection, we compare the two proposed B&P algorithms (`PF` and `DW`) with two others alternative MIP approaches, namely the Compact formulation (`C`) and the Automatic Benders of Cplex (`AB`). We report the overall CPU time in seconds, and compare the quality of lower bounds after reaching the time limit. We also report the number of added columns and generated iterations during the B&P algorithms.

4.6.1.1 Comparison between different pricing methods for solving the LP-relaxation of the model PF

Graphical summary of the obtained results comparing the four pricing methods proposed for the model PF is given in Figures 4.7 and 4.8. Cumulative charts representing the CPU times (Figure 4.7), the number of added columns (Figure 4.8(a)) and generated iterations (Figure 4.8(b)) during the column generation procedure are provided. A point with coordinates (x, y) in Figure 4.7 indicates that for y instances, the CPU

time needed to solve the LP-relaxation was x seconds or less. A similar representation of the number of added columns and generated iterations is shown in Figures 4.8(a) and 4.8(b), respectively.

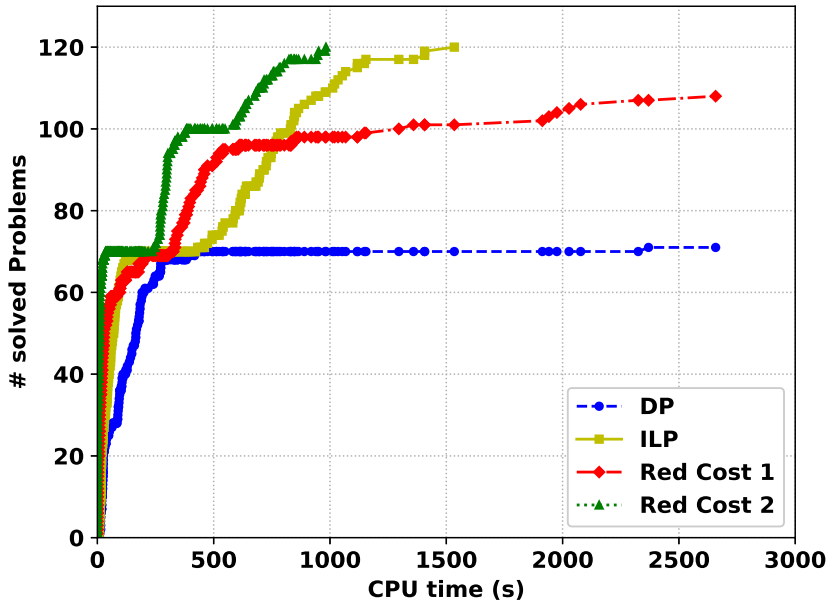
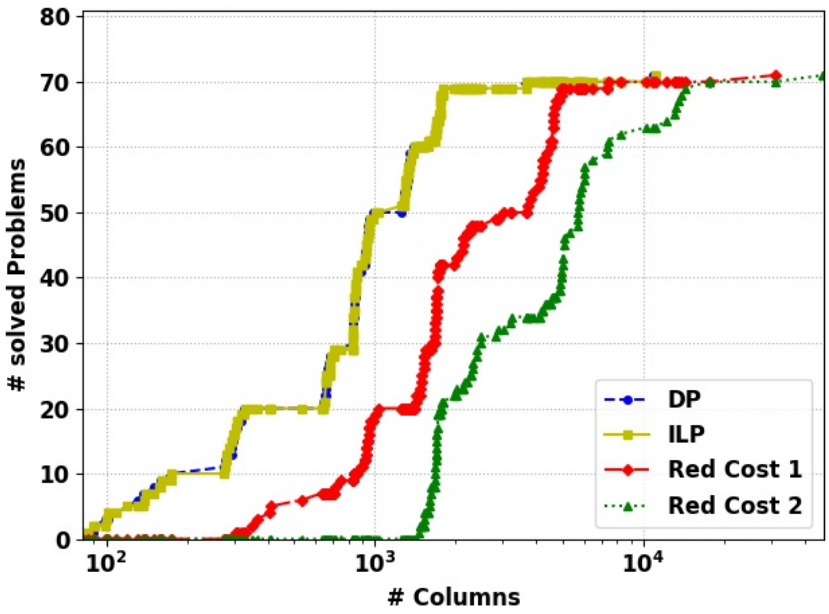


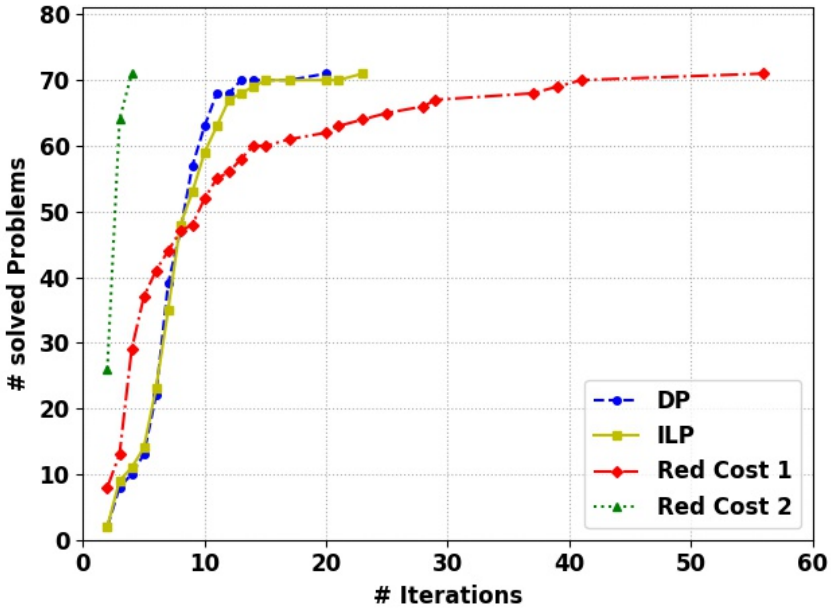
Figure 4.7: CPU time comparison between different pricing methods proposed for the relaxed path formulation.

From Figure 4.7 we observe that all 120 instances are solved by the `Red Cost 2` (resp. `ILP`) setting at the root node within less than 1000s (resp. 1534s). On the contrary, the `Red Cost 1` solves 108 instances with CPU time up to 2700s and the `DP` setting solves only 71 instances without exceeding the time limit. The fact that the `DP` consumes more time to find an elementary resource constrained shortest path, compared to the ILP method, can be explained as follows. In the setting `ILP`, the sub-tour elimination cuts are added to the pricing problem on the fly, only if they are needed. Furthermore, from one CG iteration to the other, we only modify the objective function of the underlying ILP, which allows Cplex to heavily exploit warm-starting techniques.

In order to compare the number of added columns and iterations generated during the CG procedure, we focus on 71 instances for which all four methods were able to solve the problem at the root node without exceeding the time limit of one hour.



(a) # Columns



(b) # Iterations

Figure 4.8: Comparison between four different pricing methods proposed for the relaxed path formulation with respect to the number of added columns and generated iterations.

Figure 4.8(a) shows the number of added columns for `DP`, `ILP`, `Red Cost 1` and

`Red Cost 2` during the column generation procedure. We notice that settings `DP` and `ILP` behave in the same way; this is explained by the fact that at each CG iteration, at most one column (the most violated) is added per commodity. On the other hand, for `Red Cost 1` (resp. `Red Cost 2`) several (resp. all) columns having negative reduced costs are added. This explains the huge number of added columns using `Red Cost 2`.

Figure 4.8(b) allows to compare the number of generated iterations between `DP`, `ILP`, `Red Cost 1` and `Red Cost 2` during the CG procedure. We observe that, as all columns with negative reduced cost are added at each CG iteration, the `Red Cost 2` needs less than 5 iterations to find a solution at the root node. In comparison, the setting `DP` (resp. `ILP`) needs up to 20 (resp. 23) iterations for all instances to find the LP-bound. Finally, we observe that `Red Cost 1` needs up to 25 CG iterations to solve 65 instances, and sometimes even more than 50 iterations to solve the LP-relaxation. This can be accounted to the fact that we are adding a fixed number of columns, which are not necessarily the most violated ones.

Based on the comparison of CPU times between the four pricing methods, in the following, we will consider `Red Cost 2` as our *default pricing method* for the model PF. This setting which will be denoted by `PF` in the remainder of this section.

4.6.1.2 Comparison of LP-relaxation bounds

We now turn our attention to the comparison of the quality of LP-relaxation bounds and the CPU time required for solving LP-relaxation of the formulations PF, DW and `C`. We also show the relative improvement of lower bounds, with respect to the LP-bounds provided by `C`, by both extended formulations with and without adding valid inequalities.

Figure 4.9 provides a cumulative chart in which we report the CPU time for all 120 instances from our test bed. We observe that in less 31 seconds the LP-solution for any of the considered 120 instances can be found by the compact formulation `C`. The CPU time consumed by the model PF is below 1000 seconds. Finally, the model DW can solve only 91 instances at the root node without exceeding the time limit.

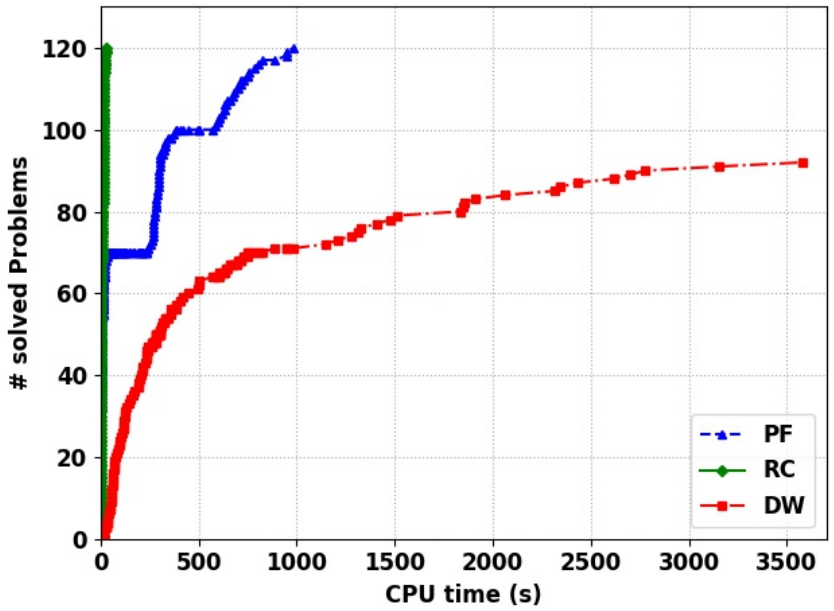


Figure 4.9: CPU time needed to solve the LP-relaxation of the `PF`, `C` and `DW` formulation.

Figures 4.10 and 4.11 depict the CPU time consumed by `PF`, `PF+VI`, `DW` and `DW+VI` settings at the root node of the branching tree. We observe that adding valid inequalities to both formulations increases the CPU time. Valid inequalities slow down the resolution at the root node for the model PF; 75% of instances were solved with CPU time below 300 seconds, whereas after adding valid inequalities, the same percentage of instances needs up to 1025 seconds. Moreover, the overall number of solved instances decreases from 120 to 109. Similarly, for the model DW, after adding valid inequalities the number of instances for which LP-relaxation can be solved drops from 92 to 66 instances.

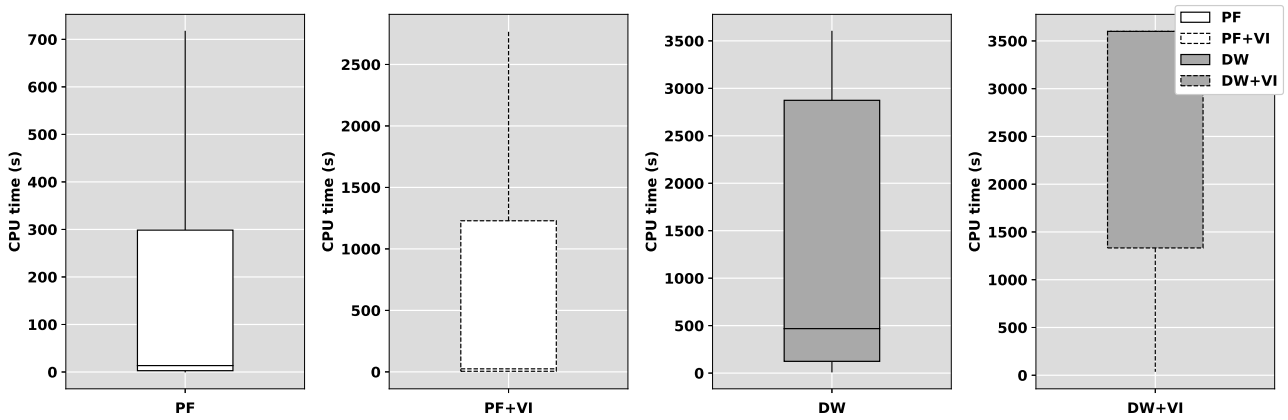


Figure 4.10: CPU time needed to solve LP-relaxation at the root node for PF and DW formulation with and without valid inequalities.

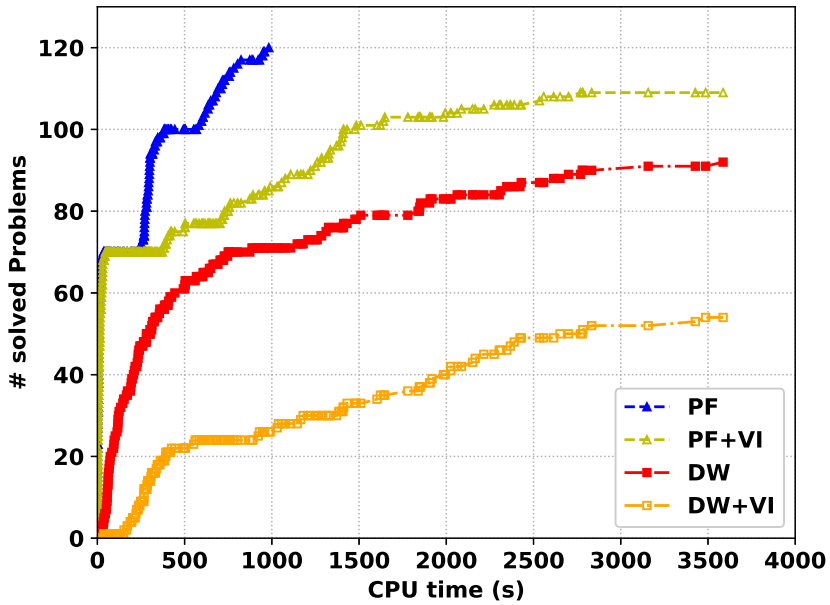


Figure 4.11: CPU time needed to solve LP-relaxation at the root node for PF and DW formulation with and without valid inequalities.

In the following, we focus on 52 instances whose LP-relaxation could be solved at the root node by PF, PF+VI, DW, DW+VI without exceeding the time limit (2 among 54 instances solved by DW+VI are not solved by PF+VI). We compare the relative improvement of lower bounds with respect to the LP-bounds obtained by the compact formulation C. Figure 4.12 illustrates the relative improvement of lower bounds, calculated as $((LB_a - LB_C) / LB_C) * 100$, with $a \in \{$PF, PF+VI, DW, DW+VI$\}$. We observe

that the lower bound at the root node could be improved by between 1% to 14% (resp. by between 3% to 29%) with PF (resp. DW) formulation for all considered instances without adding valid inequalities. Moreover, the potential benefits of adding the valid inequalities to both extended formulations are shown. We notice that adding the valid inequalities further improves the quality of their bounds at the root node. These improvements are significant for all instances. The LP-relaxation bounds of the C formulation are improved from 6% to 90% for 75% of instances solved by the model PF. This improvement ranges between 14% and 100% for 75% of instances solved by the model DW.

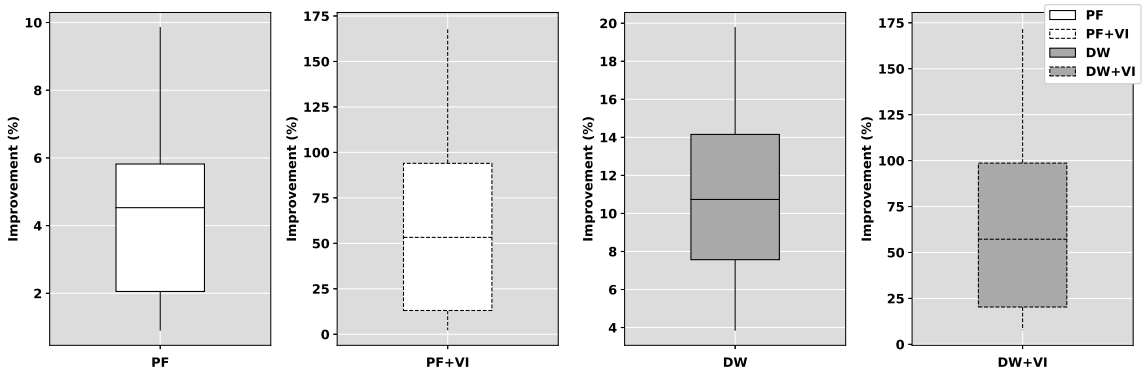


Figure 4.12: Relative improvement of LP-relaxation bounds of C formulation by using PF and DW formulation with and without valid inequalities.

The number of added columns and generated iterations during the column generation algorithms for PF, PF+VI, DW and DW+VI settings are presented in Figures 4.13-4.14 and 4.15-4.16, respectively. From these figures we conclude that for path formulation the number of priced-in columns reduces only moderately for most of the instances (from 10000 to 7500 and the number of iterations from 3 to 2). Similarly, adding these inequalities to Dantzig-Wolfe formulation increases the number of columns and iterations from ≈ 320 to ≈ 2900 and from ≈ 9 to ≈ 41 for 75% of instances, respectively; this is explained by the fact that inequalities (4.52) – (4.54) generate new dual variables and then, the master problem becomes more difficult to solve, thus, needs more columns and iterations to find a lower bound at the root node

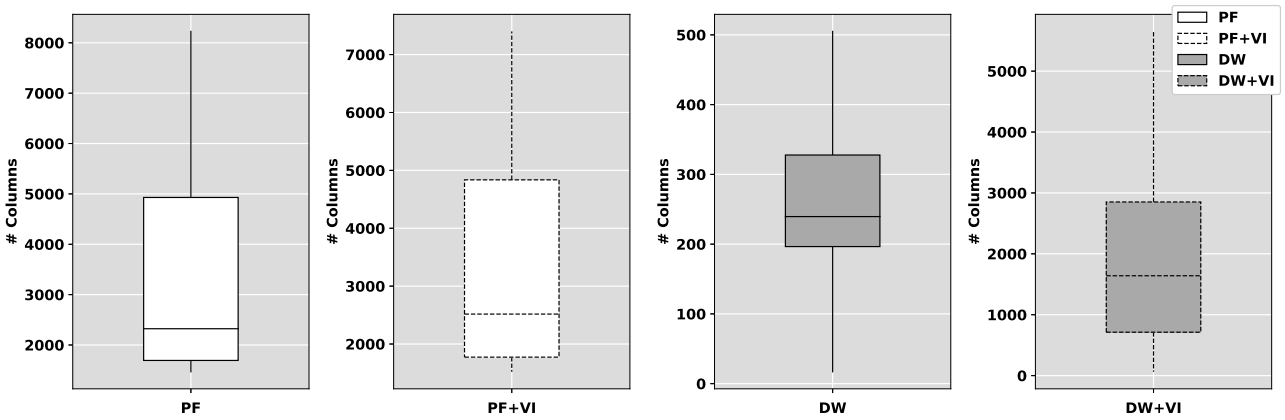


Figure 4.13: Number of added columns by PF and DW formulation with and without valid inequalities.

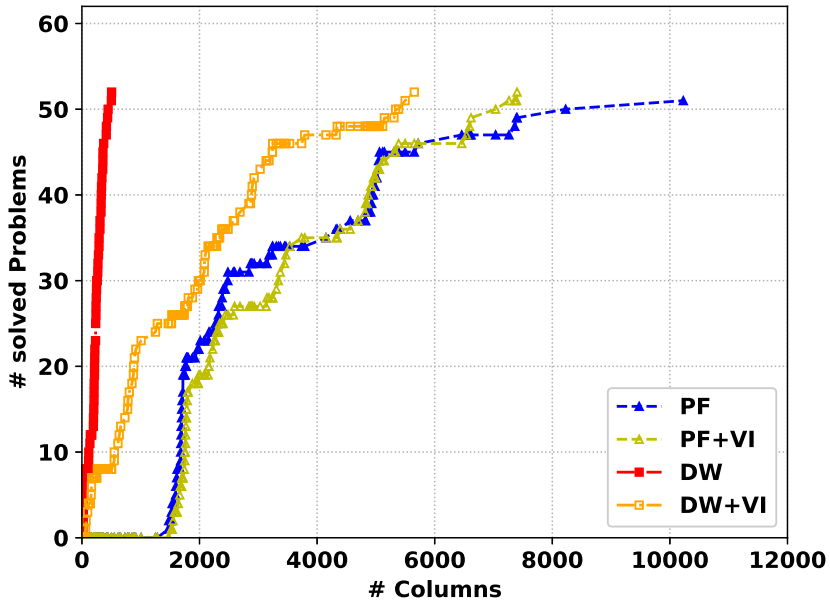


Figure 4.14: Number of added columns by PF and DW formulation with and without valid inequalities.

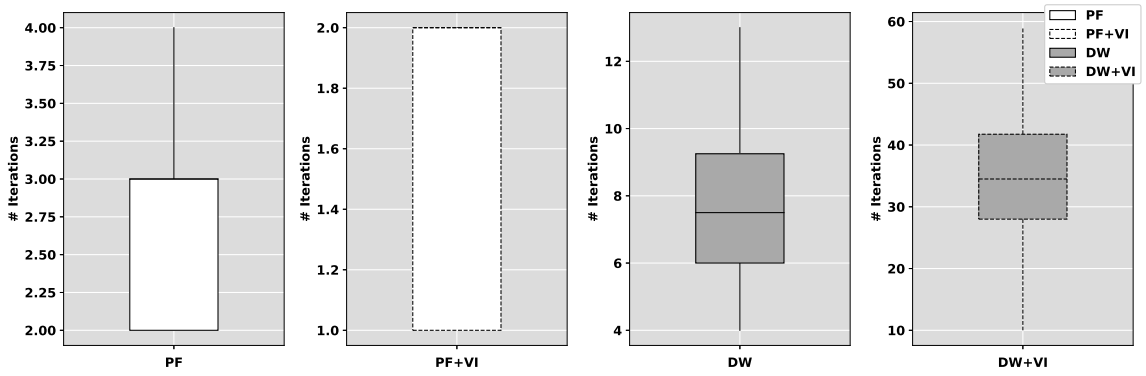


Figure 4.15: Number of generated iterations by PF and DW formulation with and without valid inequalities.

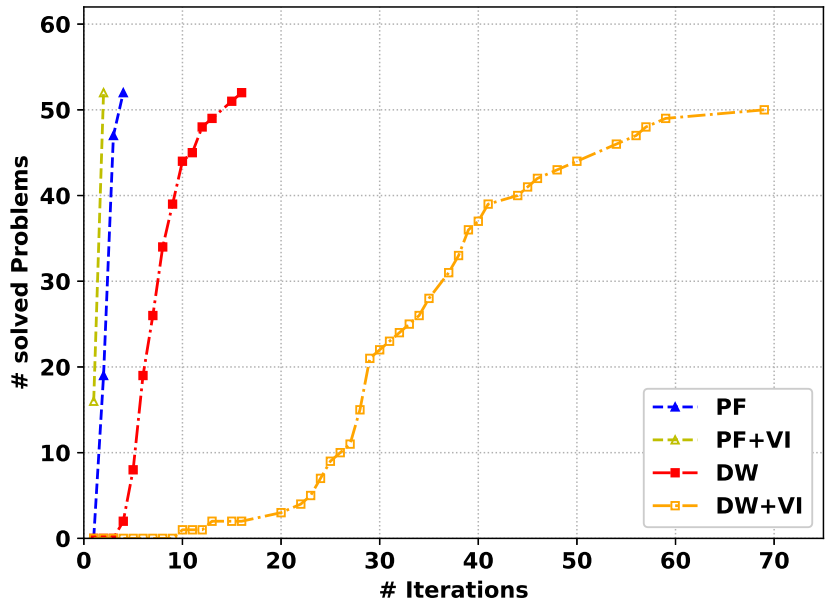


Figure 4.16: Number of generated iterations by PF and DW formulation with and without valid inequalities.

4.6.1.3 Comparison between path formulation, Dantzig-Wolfe formulation, Compact formulation and Automatic Benders

In what follows, we compare the two proposed Branch-and-Price algorithms associated with path formulation and Dantzig-Wolfe formulation with the Compact formulation and the Automatic Benders of Cplex.

In our Branch-and-Price (B&P) algorithms we initialize UBs by using one of the two heuristics defined above. Consequently, we implemented BFS (Breadth-First Search) strategy-based branching in which a global lower bound is updated at each level of the B&P tree.

All proposed methods reach the time limit for the 52 instances for which all methods could solve the LP-relaxation. Thus, in what follows we show only the Gap improvement for all methods, which is calculated as GAP $= (GLB - LB)/LB * 100\%$, where GLB represents the global lower bound found by each setting listed above, (the best known solution provided by Cplex for `C` and `AB` settings, and the best global lower bound provided by the Branch-and-Price algorithm based BFS strategy for `PF+VI` and `DW+VI` settings). Also, we compare the number of added columns and generated nodes and iterations by both Branch-and-Price algorithms.

Figures 4.17 and 4.18 display the relative Gap improvement of lower bounds. We notice that the relative Gap is improved by 5% for 75% of solved instances (i.e., 39 instances) by `DW+VI`, by 0.6% with `PF+VI`, by 2.5% with the Compact formulation and by 2.1% with the Automatic Benders of Cplex. Moreover, for 50 out of 52 instances, the Gap is improved by 10% with Dantzig-Wolfe formulation, whereas for 50 instances the path formulation improves the gap only by 2.5%, the Compact formulation and the automatic Benders by 5%. Therefore, Dantzig-Wolfe formulation provides a better solution quality.

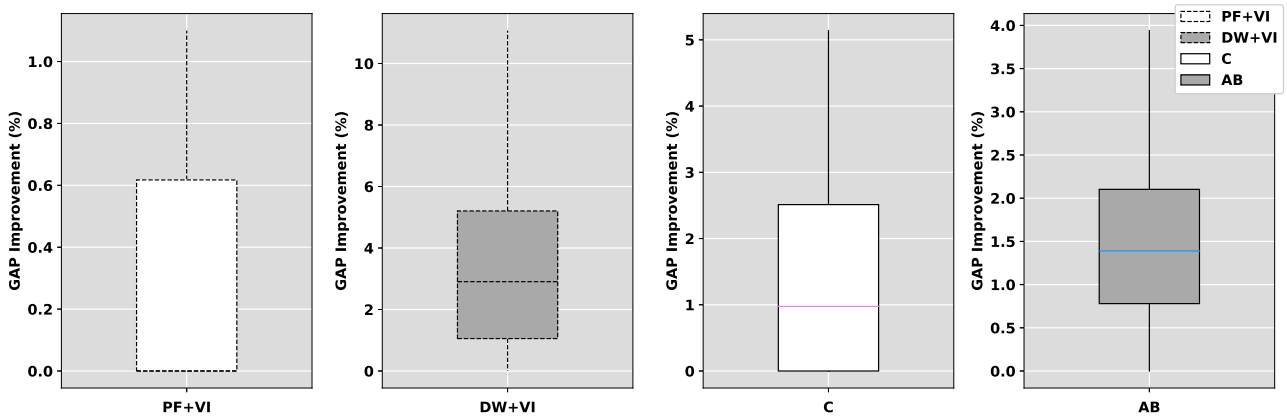


Figure 4.17: Gap improvement comparison between the path formulation, Dantzig-Wolfe formulation, the compact formulation and the Automatic Benders of Cplex.

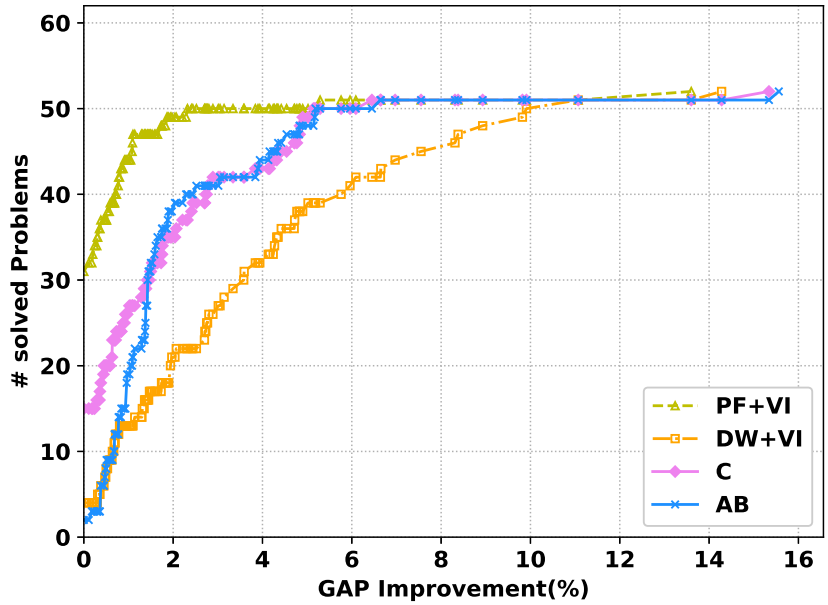


Figure 4.18: Gap improvement comparison between the path formulation, Dantzig-Wolfe formulation, the compact formulation and the Automatic Benders of Cplex.

Box-plots and charts given in Figures 4.19, 4.20 and 4.21 compare the overall number of added columns during the B&P algorithm, the number of branching nodes and the number of generated iterations for `PF+VI` and `DW+VI`, respectively. We observe that the setting `PF+VI` needs more columns and more iterations and also branches more than `DW+VI`. The vast number of added columns for the setting `PF+VI` is explained by the fact that we are adding all columns with negative reduced cost at each iteration of the CG procedure. This is in contrast to `DW+VI` where we are using LP-based pricing method to add only the most violated ones. Furthermore, the number of variables that are required to be integer in the model PF is much larger compared to the model DW, which also explains the larger number of branching nodes for the setting `PF+VI`.

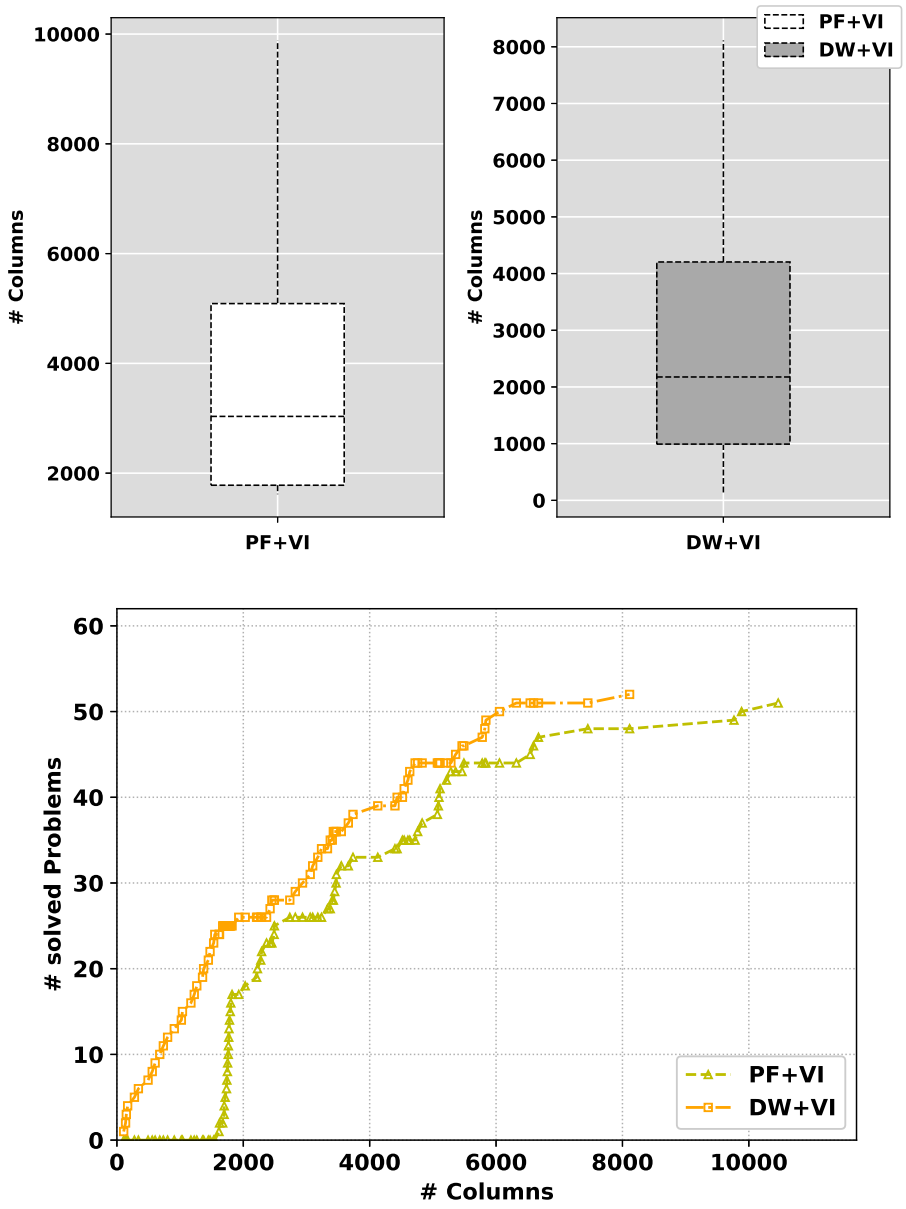


Figure 4.19: Number of added columns comparison between the path formulation and Dantzig-Wolfe formulation.

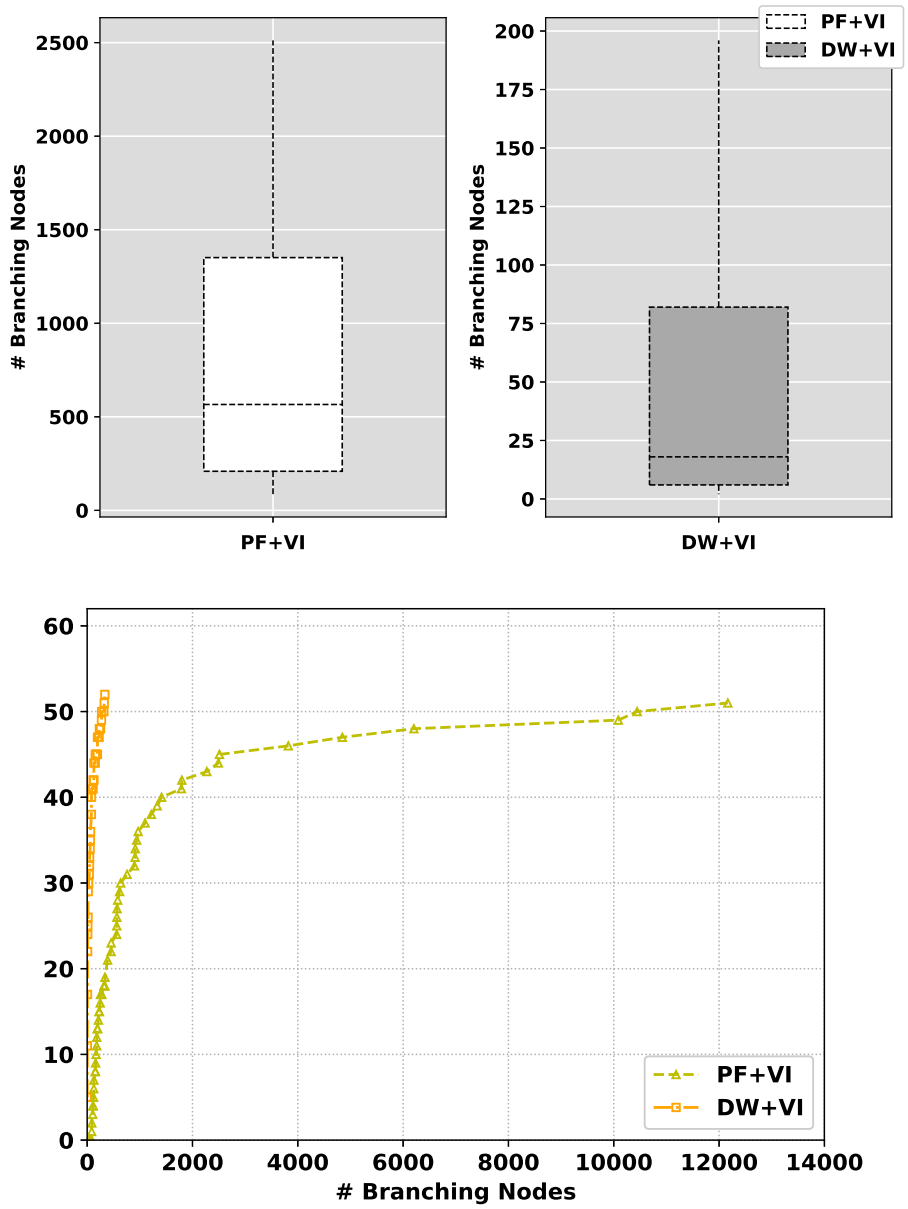


Figure 4.20: Number of branching nodes comparison between the path formulation and Dantzig-Wolfe formulation.

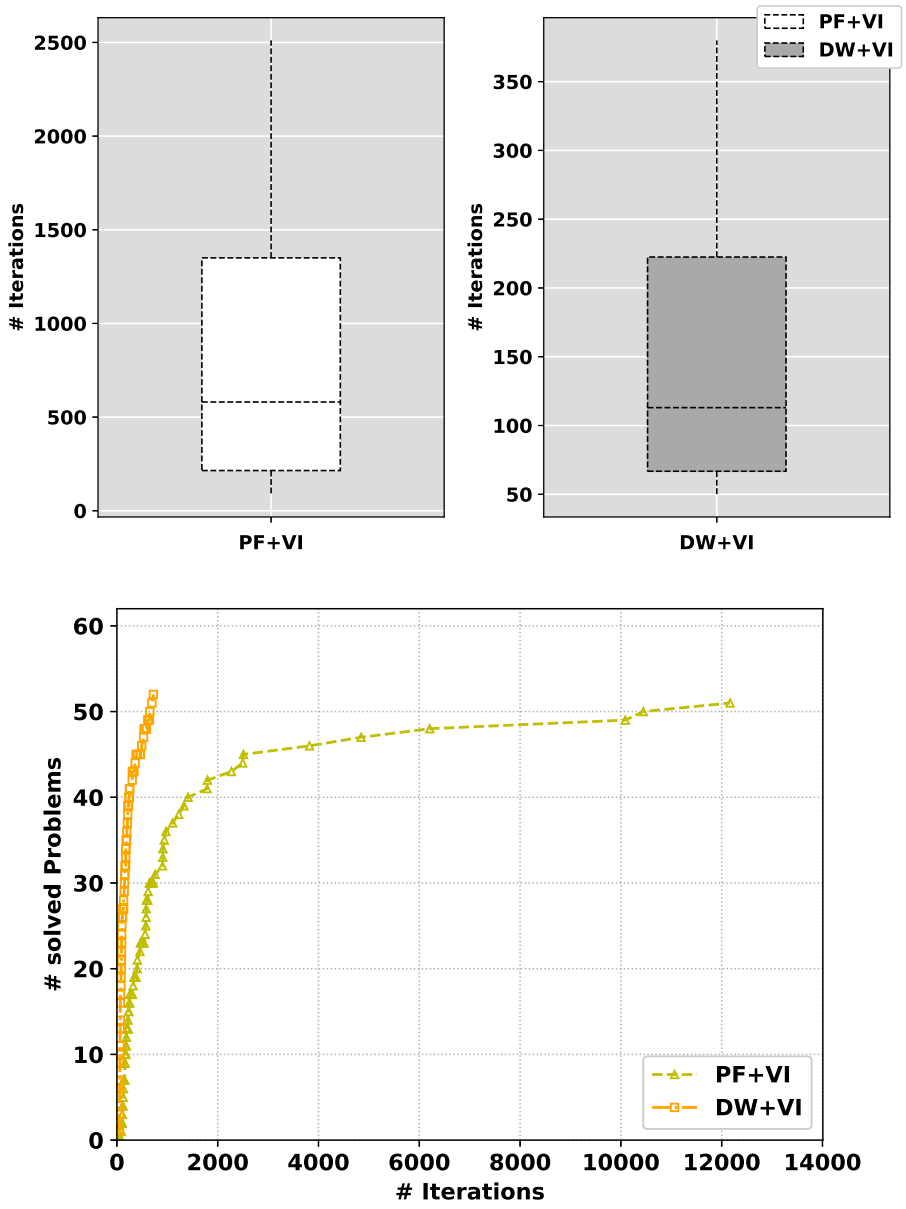


Figure 4.21: Number of generated iterations comparison between the path formulation and Dantzig-Wolfe formulation.

4.6.2 Detailed results

Tables provided in this subsection show more detailed results for each setting introduced in Section 4.6. First, we describe abbreviations shown in each column in the tables:

Abbreviations	Description
Col	Number of added columns during the column generation procedure.
Iter	Number of generated iteration by the column generation procedure.
t[s]	CPU time in seconds.
Gap	Relative gap improvement.
Nodes	Number of generated nodes during the Branch-and-Price algorithm.

Table 4.4: Description of the tables abbreviations

Tables 4.5-4.8 present the results with Path formulation in which different pricing problems are tested. Tables 4.9-4.11 show the detailed results obtained by the relaxed Path formulation, the relaxed Dantzig-Wolfe formulation and Relaxed Compact formulation, for which we compare the CPU time. The results comparing both extended formulations with and without the adding of valid inequalities are provided in Tables 4.12-4.14. Finally, Tables 4.15 and 4.16 display the bounds improvement of both Branch-and-Price algorithms compared to the compact formulation and the automatic Benders of Cplex.

Sign "$-$" in columns "Col", "Iter", "Gap" and "Nodes" in tables indicates that no information can be provided and that no solution is found. Moreover, TL in "t[s]" columns illustrates the fact that the time limit is reached. The relative gap improvement is reported as Gap $= (GLB - LB)/LB * 100\%$, where GLB denotes the global lower bound found by each setting and LB the worst known lower bound (the minimum overall global lower bounds).

Instances					Path formulation											
					DP			ILP			Red Cost 1			Red Cost 2		
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Col	Iter	t[s]	Col	Iter	t[s]	Col	Iter	t[s]	Col	Iter
$Abilene_1$	12	30	132	6	18.99	855	3	10.38	857	4	**0.70**	1718	3	0.78	1718	3
$Abilene_2$	12	30	132	6	15.37	847	3	11.64	849	3	**0.48**	1718	2	0.57	1718	2
$Abilene_3$	12	30	132	6	13.80	839	3	15.09	841	3	**0.45**	1687	2	0.48	1687	2
$Abilene_4$	12	30	132	6	18.62	843	3	16.57	846	3	0.61	1698	2	**0.58**	1698	2
$Abilene_5$	12	30	132	6	13.82	833	3	9.37	834	3	0.57	1683	3	**0.49**	1683	3
$Abilene_6$	12	30	132	6	20.36	832	4	9.10	833	4	**0.43**	1689	2	0.48	1689	2
$Abilene_7$	12	30	132	6	9.11	827	2	12.15	831	2	**0.40**	1680	2	0.42	1680	2
$Abilene_8$	12	30	132	6	9.55	828	2	9.72	834	3	**0.52**	1695	2	0.53	1695	2
$Abilene_9$	12	30	132	6	14.72	826	3	15.50	830	3	**0.40**	1679	2	0.52	1679	2
$Abilene_{10}$	12	30	132	6	19.35	855	4	18.09	855	3	**0.47**	1717	3	0.65	1717	3
$Atlanta_1$	15	44	210	6	122.10	3676	5	65.85	3685	5	**9.39**	7392	2	11.74	7392	2
$Atlanta_2$	15	44	210	6	179.57	1670	10	109.81	1679	11	**8.75**	4233	6	10.49	4563	4
$Atlanta_3$	15	44	210	6	191.68	1762	7	90.54	1759	7	9.01	4616	5	**6.90**	4929	3
$Atlanta_4$	15	44	210	6	165.41	1766	9	76.36	1765	8	8.77	4616	6	**6.54**	4996	3
$Atlanta_5$	15	44	210	6	183.70	1586	9	72.26	1587	7	6.59	3794	5	**5.75**	4151	3
$Atlanta_6$	15	44	210	6	148.62	1687	7	91.19	1692	7	**9.21**	4542	6	10.02	4930	3
$Atlanta_7$	15	44	210	6	165.27	1733	8	78.19	1744	9	**6.74**	4628	5	7.42	4985	3
$Atlanta_8$	15	44	210	6	177.15	1787	8	114.73	1802	8	8.73	4662	6	**8.46**	5046	3
$Atlanta_9$	15	44	210	6	137.63	1693	7	81.66	1705	8	7.69	4530	5	**5.85**	4851	4
$Atlanta_{10}$	15	44	210	6	163.04	1757	7	87.75	1761	7	**6.39**	4762	5	7.16	5046	3
$Di-yuan_1$	11	84	22	6	172.98	304	13	50.78	286	9	104.18	1724	25	**13.51**	7361	2
$Di-yuan_2$	11	84	22	6	168.10	273	11	43.93	279	13	101.03	1479	23	**13.42**	8227	2
$Di-yuan_3$	11	84	22	6	276.56	324	10	61.87	331	14	128.25	2212	20	**27.97**	14277	2
$Di-yuan_4$	11	84	22	6	418.39	324	20	75.17	319	23	177.33	2819	29	**22.35**	13831	2
$Di-yuan_5$	11	84	22	6	272.03	308	10	59.20	307	11	344.66	3011	56	**22.14**	13549	2
$Di-yuan_6$	11	84	22	6	188.97	297	11	48.39	297	12	125.05	2129	28	**16.21**	10228	3
$Di-yuan_7$	11	84	22	6	148.08	278	9	32.05	282	9	218.06	2283	37	**19.95**	13053	2
$Di-yuan_8$	11	84	22	6	269.00	286	11	36.69	278	10	200.32	2123	41	**16.37**	12235	2
$Di-yuan_9$	11	84	22	6	272.64	295	10	56.72	294	12	178.66	2005	21	**34.84**	17608	2
$Di-yuan_{10}$	11	84	22	6	383.29	301	10	44.57	304	15	96.90	2141	17	**20.52**	13190	2

Table 4.5: Results for SNDlib instances with Path formulation in which different pricing problems are tested.

Instances					Path formulation											
					DP			ILP			Red Cost 1			Red Cost 2		
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Col	Iter	t[s]	Col	Iter	t[s]	Col	Iter	t[s]	Col	Iter
$France_1$	25	90	300	6	TL	–	–	889.80	4982	12	1147.86	26093	39	**373.18**	55964	3
$France_2$	25	90	300	6	TL	–	–	847.38	7148	12	540.25	24985	20	**272.30**	47514	5
$France_3$	25	90	300	6	TL	–	–	829.99	3788	11	855.73	18620	29	**333.01**	54355	3
$France_4$	25	90	300	6	TL	–	–	536.18	10935	11	332.81	32844	11	**241.07**	50397	3
$France_5$	25	90	300	6	TL	–	–	640.83	7966	11	511.99	26597	19	**251.89**	47509	3
$France_6$	25	90	300	6	TL	–	–	712.33	5898	9	833.99	23612	29	**326.58**	55720	4
$France_7$	25	90	300	6	TL	–	–	725.85	10988	12	335.39	35667	9	**301.56**	55379	3
$France_8$	25	90	300	6	2369.32	10879	7	1034.70	11063	12	387.12	31016	7	**345.98**	46745	4
$France_9$	25	90	300	6	TL	–	–	727.19	11024	11	375.06	31561	9	**299.81**	50328	3
$France_{10}$	25	90	300	6	TL	–	–	548.89	3636	11	1359.19	18820	48	**296.71**	50403	3
$Geant_1$	22	72	462	6	TL	–	–	1052.82	6193	17	459.45	27045	19	**382.87**	50877	3
$Geant_2$	22	72	462	6	TL	–	–	431.23	5549	9	333.70	21532	12	**298.15**	48652	3
$Geant_3$	22	72	462	6	TL	–	–	493.94	5854	11	422.04	22223	17	**290.98**	47741	3
$Geant_4$	22	72	462	6	TL	–	–	1406.73	6019	12	441.42	22861	20	**269.43**	45164	4
$Geant_5$	22	72	462	6	TL	–	–	612.38	5843	12	443.91	24515	19	**303.14**	49197	3
$Geant_6$	22	72	462	6	TL	–	–	768.70	6121	11	417.78	23754	17	**271.81**	45707	4
$Geant_7$	22	72	462	6	TL	–	–	733.68	6130	13	363.20	24899	14	**283.34**	49418	3
$Geant_8$	22	72	462	6	TL	–	–	635.99	5831	10	377.73	23250	14	**293.47**	51436	3
$Geant_9$	22	72	462	6	TL	–	–	844.38	5768	11	473.46	23537	23	**270.50**	46695	4
$Geant_{10}$	22	72	462	6	TL	–	–	622.04	5931	12	450.92	23985	18	**286.78**	50382	3
$Newyork_1$	16	98	240	6	TL	–	–	1019.83	3634	19	TL	–	–	**823.65**	114459	3
$Newyork_2$	16	98	240	6	TL	–	–	**594.45**	3177	16	TL	–	–	616.65	107859	3
$Newyork_3$	16	98	240	6	TL	–	–	864.99	3611	14	TL	–	–	**649.80**	108805	3
$Newyork_4$	16	98	240	6	TL	–	–	**592.07**	3216	11	TL	–	–	635.65	109429	3
$Newyork_5$	16	98	240	6	TL	–	–	939.13	3567	18	TL	–	–	**756.04**	115911	3
$Newyork_6$	16	98	240	6	TL	–	–	**493.44**	3308	13	2077.54	18748	34	634.24	106288	3
$Newyork_7$	16	98	240	6	TL	–	–	**460.70**	3224	13	2027.64	22764	35	691.47	113000	3
$Newyork_8$	16	98	240	6	TL	–	–	1117.41	3342	14	1975.46	21250	31	**675.41**	114708	3
$Newyork_9$	16	98	240	6	TL	–	–	**583.97**	3128	14	TL	–	–	716.51	121537	3
$Newyork_{10}$	16	98	240	6	TL	–	–	918.98	3442	19	2657.83	26290	48	**604.96**	112781	3

Table 4.6: Results for SNDlib instances with Path formulation in which different pricing problems are tested.

Instances					Path formulation											
					DP			ILP			Red Cost 1			Red Cost 2		
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Col	Iter	t[s]	Col	Iter	t[s]	Col	Iter	t[s]	Col	Iter
$Nobel-eu_1$	28	82	378	6	TL	–	–	**787.72**	4360	12	TL	–	–	982.64	104788	3
$Nobel-eu_2$	28	82	378	6	TL	–	–	**832.90**	4560	12	TL	–	–	947.06	109253	4
$Nobel-eu_3$	28	82	378	6	TL	–	–	**613.89**	5427	11	1912.86	28125	29	803.63	100804	3
$Nobel-eu_4$	28	82	378	6	TL	–	–	753.63	11865	14	1295.97	46523	14	**746.16**	79113	4
$Nobel-eu_5$	28	82	378	6	TL	–	–	**618.72**	3896	10	TL	–	–	950.99	98675	3
$Nobel-eu_6$	28	82	378	6	TL	–	–	**696.20**	4298	12	TL	–	–	716.78	92789	3
$Nobel-eu_7$	28	82	378	6	TL	–	–	**539.18**	4260	11	2325.18	25881	34	781.82	102426	3
$Nobel-eu_8$	28	82	378	6	TL	–	–	759.50	4180	14	TL	–	–	**597.47**	84123	3
$Nobel-eu_9$	28	82	378	6	TL	–	–	682.20	3936	10	TL	–	–	**582.09**	88657	3
$Nobel-eu_{10}$	28	82	378	6	TL	–	–	747.60	6423	14	1941.05	34771	34	**681.54**	92449	3
$Nobel-ger_1$	17	52	121	6	243.14	1320	11	97.41	1302	10	14.79	4098	8	**10.18**	5900	3
$Nobel-ger_2$	17	52	121	6	248.75	1329	9	104.88	1324	8	18.80	4129	11	**12.50**	5705	3
$Nobel-ger_3$	17	52	121	6	181.51	1281	7	70.12	1284	8	18.03	4369	13	**8.82**	5783	3
$Nobel-ger_4$	17	52	121	6	142.38	1263	7	71.24	1260	7	15.54	3734	9	**8.58**	5028	3
$Nobel-ger_5$	17	52	121	6	236.01	1340	9	121.14	1374	11	19.95	4243	11	**10.36**	5724	3
$Nobel-ger_6$	17	52	121	6	192.33	1400	9	159.23	1408	10	20.40	4889	13	**13.77**	5809	4
$Nobel-ger_7$	17	52	121	6	154.46	1328	8	98.02	1338	8	23.55	4634	14	**11.51**	6032	3
$Nobel-ger_8$	17	52	121	6	183.60	1289	8	74.09	1299	8	12.09	3897	7	**10.19**	5300	3
$Nobel-ger_9$	17	52	121	6	129.05	1346	8	177.77	1358	8	21.26	4976	14	**8.42**	6026	3
$Nobel-ger_{10}$	17	52	121	6	204.65	1321	9	102.34	1308	10	15.45	4212	8	**10.99**	6051	3
$Nobel-us_1$	14	42	91	6	88.14	921	7	29.19	918	6	**1.70**	1958	3	1.76	2013	3
$Nobel-us_2$	14	42	91	6	106.22	932	7	57.42	930	7	3.14	2247	4	**2.73**	2385	3
$Nobel-us_3$	14	42	91	6	50.70	924	5	24.75	929	5	**1.81**	2175	3	1.87	2322	3
$Nobel-us_4$	14	42	91	6	96.13	846	8	28.27	837	7	**2.25**	1841	4	2.27	1967	4
$Nobel-us_5$	14	42	91	6	90.21	942	7	57.38	948	9	2.27	2123	5	**1.75**	2271	3
$Nobel-us_6$	14	42	91	6	102.52	982	8	51.08	1005	8	**4.66**	2293	5	5.01	2482	3
$Nobel-us_7$	14	42	91	6	95.96	941	8	42.98	949	8	2.19	2257	4	**2.06**	2402	3
$Nobel-us_8$	14	42	91	6	86.47	958	7	42.39	962	8	2.72	2320	4	**2.59**	2483	3
$Nobel-us_9$	14	42	91	6	63.50	895	6	37.02	894	6	2.08	2040	4	**1.96**	2160	3
$Nobel-us_{10}$	14	42	91	6	89.50	946	7	46.87	941	7	1.88	2220	4	**1.66**	2324	3

Table 4.7: Results for SNDlib instances with Path formulation in which different pricing problems are tested.

Instances					Path formulation											
					DP			ILP			Red Cost 1			Red Cost 2		
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Col	Iter	t[s]	Col	Iter	t[s]	Col	Iter	t[s]	Col	Iter
Pdh_1	11	68	24	6	25.89	82	9	5.73	85	11	6.49	349	11	**1.69**	1776	2
Pdh_2	11	68	24	6	93.98	131	9	25.34	138	9	26.47	745	10	**8.28**	6460	2
Pdh_3	11	68	24	6	38.86	109	6	35.40	102	2	12.03	404	8	**4.58**	4323	2
Pdh_4	11	68	24	6	92.85	137	11	17.03	140	10	26.51	636	12	**5.63**	5061	2
Pdh_5	11	68	24	6	106.26	151	13	48.28	158	12	84.31	1032	39	**6.62**	5751	3
Pdh_6	11	68	24	6	110.15	174	10	83.32	175	10	30.69	717	10	**12.18**	7378	2
Pdh_7	11	68	24	6	55.12	159	9	62.81	161	7	20.22	538	10	**6.20**	5719	2
Pdh_8	11	68	24	6	29.98	91	7	8.82	90	6	16.72	365	10	**5.57**	2867	2
Pdh_9	11	68	24	6	21.06	120	7	5.07	120	6	10.56	409	7	**4.27**	3182	2
Pdh_{10}	11	68	24	6	14.01	99	5	25.20	101	7	6.52	304	5	**2.77**	3241	2
$Polska_1$	12	36	66	6	33.53	650	6	18.48	658	6	1.74	1543	4	**1.55**	1619	3
$Polska_2$	12	36	66	6	25.52	663	6	14.45	662	5	1.56	1492	4	**1.40**	1523	3
$Polska_3$	12	36	66	6	42.04	647	7	22.49	649	7	**1.99**	1430	4	2.01	1468	3
$Polska_4$	12	36	66	6	22.38	660	6	13.82	658	6	**1.39**	1428	4	1.60	1477	4
$Polska_5$	12	36	66	6	26.42	659	7	16.18	658	7	1.52	1521	4	**1.37**	1593	3
$Polska_6$	12	36	66	6	23.77	705	6	18.15	707	6	1.65	1629	4	**1.58**	1704	3
$Polska_7$	12	36	66	6	26.08	850	6	19.76	841	6	1.01	1754	4	**0.97**	1755	3
$Polska_8$	12	36	66	6	24.96	678	6	16.60	679	6	1.75	1523	4	**1.73**	1596	4
$Polska_9$	12	36	66	6	26.25	677	7	19.42	687	8	1.40	1487	4	**1.38**	1535	3
$Polska_{10}$	12	36	66	6	23.23	670	6	17.54	681	8	2.16	1522	4	**2.09**	1612	3
$Ta1_1$	24	102	396	6	TL	–	–	1154.51	4794	11	512.34	20132	17	**302.03**	42939	3
$Ta1_2$	24	102	396	6	TL	–	–	1117.87	4854	9	392.57	19963	12	**298.02**	45994	3
$Ta1_3$	24	102	396	6	TL	–	–	774.66	4918	9	365.99	20583	12	**273.87**	46165	3
$Ta1_4$	24	102	396	6	TL	–	–	847.80	4750	9	314.43	16992	10	**260.81**	42083	3
$Ta1_5$	24	102	396	6	TL	–	–	697.67	5020	9	343.77	20050	11	**280.87**	47909	3
$Ta1_6$	24	102	396	6	TL	–	–	1011.52	5127	12	397.10	20873	14	**325.57**	47528	3
$Ta1_7$	24	102	396	6	TL	–	–	979.64	4969	11	610.35	21820	26	**266.30**	44334	3
$Ta1_8$	24	102	396	6	TL	–	–	1406.75	5173	16	401.71	19924	14	**281.88**	43662	4
$Ta1_9$	24	102	396	6	TL	–	–	1067.35	5155	11	523.65	20325	20	**293.95**	46275	4
$Ta1_{10}$	24	102	396	6	TL	–	–	1534.39	5260	14	460.19	22210	19	**269.40**	44598	3

Table 4.8: Results for SNDlib instances with Path formulation in which different pricing problems are tested.

Instances					PF			DW			C
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Col	Iter	t[s]	Col	Iter	t[s]
$Abilene_1$	12	30	132	6	0.78	1718	3	133.49	330	8	2.40
$Abilene_2$	12	30	132	6	0.57	1718	2	55.75	282	6	2.34
$Abilene_3$	12	30	132	6	0.48	1687	2	51.87	218	5	2.16
$Abilene_4$	12	30	132	6	0.58	1698	2	61.98	334	7	2.82
$Abilene_5$	12	30	132	6	0.49	1683	3	72.64	214	5	2.33
$Abilene_6$	12	30	132	6	0.48	1689	2	52.01	232	6	1.24
$Abilene_7$	12	30	132	6	0.42	1680	2	57.97	220	7	1.14
$Abilene_8$	12	30	132	6	0.53	1695	2	61.45	437	7	1.24
$Abilene_9$	12	30	132	6	0.52	1679	2	57.34	308	5	1.20
$Abilene_{10}$	12	30	132	6	0.65	1717	3	43.41	240	6	1.94
$Atlanta_1$	15	44	210	6	11.74	7392	2	503.66	501	8	6.95
$Atlanta_2$	15	44	210	6	10.49	4563	4	172.56	505	5	4.48
$Atlanta_3$	15	44	210	6	6.90	4929	3	302.41	206	6	2.91
$Atlanta_4$	15	44	210	6	6.54	4996	3	281.46	207	6	3.40
$Atlanta_5$	15	44	210	6	5.75	4151	3	330.98	417	8	2.90
$Atlanta_6$	15	44	210	6	10.02	4930	3	194.32	358	7	3.23
$Atlanta_7$	15	44	210	6	7.42	4985	3	197.63	208	6	2.68
$Atlanta_8$	15	44	210	6	8.46	5046	3	502.80	341	8	3.22
$Atlanta_9$	15	44	210	6	5.85	4851	4	316.95	446	10	3.15
$Atlanta_{10}$	15	44	210	6	7.16	5046	3	358.66	370	12	2.63
$Di-yuan_1$	11	84	22	6	13.51	7361	2	228.98	117	10	0.85
$Di-yuan_2$	11	84	22	6	13.42	8227	2	281.78	136	16	0.45
$Di-yuan_3$	11	84	22	6	27.97	14277	2	384.60	135	11	0.81
$Di-yuan_4$	11	84	22	6	22.35	13831	2	408.55	99	10	0.38
$Di-yuan_5$	11	84	22	6	22.14	13549	2	216.27	103	9	0.46
$Di-yuan_6$	11	84	22	6	16.21	10228	3	355.63	143	15	0.50
$Di-yuan_7$	11	84	22	6	19.95	13053	2	237.66	114	10	0.42
$Di-yuan_8$	11	84	22	6	16.37	12235	2	167.02	86	9	0.70
$Di-yuan_9$	11	84	22	6	34.84	17608	2	567.53	110	10	0.84
$Di-yuan_{10}$	11	84	22	6	20.52	13190	2	151.02	101	8	0.42
$France_1$	25	90	300	6	373.18	55964	3	TL	–	–	19.65
$France_2$	25	90	300	6	272.30	47514	5	TL	–	–	30.85
$France_3$	25	90	300	6	333.01	54355	3	3586.68	500	6	11.13
$France_4$	25	90	300	6	241.07	50397	3	TL	–	–	24.07
$France_5$	25	90	300	6	251.89	47509	3	TL	–	–	19.36
$France_6$	25	90	300	6	326.58	55720	4	3157.17	451	7	12.77
$France_7$	25	90	300	6	301.56	55379	3	TL	–	–	18.51
$France_8$	25	90	300	6	345.98	46745	4	TL	–	–	23.15
$France_9$	25	90	300	6	299.81	50328	3	TL	–	–	25.30
$France_{10}$	25	90	300	6	296.71	50403	3	2700.80	409	5	10.55

Table 4.9: Results for SNDlib instances with relaxed Path formulation, relaxed Dantzig-Wolfe formulation and Relaxed Compact formulation.

Instances					PF			DW			C
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Col	Iter	t[s]	Col	Iter	t[s]
$Geant_1$	22	72	462	6	382.87	50877	3	1510.34	414	4	12.95
$Geant_2$	22	72	462	6	298.15	48652	3	TL	–	–	11.55
$Geant_3$	22	72	462	6	290.98	47741	3	1310.94	430	3	11.69
$Geant_4$	22	72	462	6	269.43	45164	4	1848.17	1306	7	16.15
$Geant_5$	22	72	462	6	303.14	49197	3	2429.54	864	5	11.35
$Geant_6$	22	72	462	6	271.81	45707	4	2346.37	784	5	13.15
$Geant_7$	22	72	462	6	283.34	49418	3	TL	–	–	12.06
$Geant_8$	22	72	462	6	293.47	51436	3	2779.17	436	5	11.75
$Geant_9$	22	72	462	6	270.50	46695	4	1838.10	704	6	10.59
$Geant_{10}$	22	72	462	6	286.78	50382	3	2312.52	423	4	11.66
$Newyork_1$	16	98	240	6	823.65	114459	3	TL	–	–	9.29
$Newyork_2$	16	98	240	6	616.65	107859	3	TL	–	–	8.97
$Newyork_3$	16	98	240	6	649.80	108805	3	TL	–	–	9.89
$Newyork_4$	16	98	240	6	635.65	109429	3	TL	–	–	6.41
$Newyork_5$	16	98	240	6	756.04	115911	3	TL	–	–	7.03
$Newyork_6$	16	98	240	6	634.24	106288	3	TL	–	–	8.94
$Newyork_7$	16	98	240	6	691.47	113000	3	TL	–	–	6.84
$Newyork_8$	16	98	240	6	675.41	114708	3	TL	–	–	11.29
$Newyork_9$	16	98	240	6	716.51	121537	3	TL	–	–	7.01
$Newyork_{10}$	16	98	240	6	604.96	112781	3	TL	–	–	6.96
$Nobel-eu_1$	28	82	378	6	982.64	104788	3	TL	–	–	14.33
$Nobel-eu_2$	28	82	378	6	947.06	109253	4	TL	–	–	17.76
$Nobel-eu_3$	28	82	378	6	803.63	100804	3	TL	–	–	12.60
$Nobel-eu_4$	28	82	378	6	746.16	79113	4	TL	–	–	15.93
$Nobel-eu_5$	28	82	378	6	950.99	98675	3	TL	–	–	14.18
$Nobel-eu_6$	28	82	378	6	716.78	92789	3	TL	–	–	11.92
$Nobel-eu_7$	28	82	378	6	781.82	102426	3	TL	–	–	11.06
$Nobel-eu_8$	28	82	378	6	597.47	84123	3	TL	–	–	11.46
$Nobel-eu_9$	28	82	378	6	582.09	88657	3	TL	–	–	10.18
$Nobel-eu_{10}$	28	82	378	6	681.54	92449	3	TL	–	–	16.89
$Nobel-ger_1$	17	52	121	6	10.18	5900	3	443.90	602	8	2.73
$Nobel-ger_2$	17	52	121	6	12.50	5705	3	749.74	650	11	2.07
$Nobel-ger_3$	17	52	121	6	8.82	5783	3	312.16	904	8	3.23
$Nobel-ger_4$	17	52	121	6	8.58	5028	3	642.54	775	8	1.92
$Nobel-ger_5$	17	52	121	6	10.36	5724	3	492.78	517	9	2.13
$Nobel-ger_6$	17	52	121	6	13.77	5809	4	605.83	1039	10	2.26
$Nobel-ger_7$	17	52	121	6	11.51	6032	3	416.67	428	7	2.39
$Nobel-ger_8$	17	52	121	6	10.19	5300	3	662.80	635	7	1.81
$Nobel-ger_9$	17	52	121	6	8.42	6026	3	727.86	667	11	1.64
$Nobel-ger_{10}$	17	52	121	6	10.99	6051	3	703.07	226	11	3.60

Table 4.10: Results for SNDlib instances with relaxed Path formulation, relaxed Dantzig-Wolfe formulation and Relaxed Compact formulation.

Instances					PF			DW			C
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Col	Iter	t[s]	Col	Iter	t[s]
$Nobel-us_1$	14	42	91	6	1.76	2013	3	264.27	325	12	1.27
$Nobel-us_2$	14	42	91	6	2.73	2385	3	243.46	293	11	2.14
$Nobel-us_3$	14	42	91	6	1.87	2322	3	82.76	291	7	1.31
$Nobel-us_4$	14	42	91	6	2.27	1967	4	112.86	307	7	1.56
$Nobel-us_5$	14	42	91	6	1.75	2271	3	212.29	270	8	1.33
$Nobel-us_6$	14	42	91	6	5.01	2482	3	207.01	354	9	1.66
$Nobel-us_7$	14	42	91	6	2.06	2402	3	121.12	195	6	1.40
$Nobel-us_8$	14	42	91	6	2.59	2483	3	233.45	239	10	1.10
$Nobel-us_9$	14	42	91	6	1.96	2160	3	92.59	273	6	1.27
$Nobel-us_{10}$	14	42	91	6	1.66	2324	3	192.17	234	9	1.99
Pdh_1	11	68	24	6	1.69	1776	2	14.57	17	4	0.56
Pdh_2	11	68	24	6	8.28	6460	2	117.73	31	8	0.69
Pdh_3	11	68	24	6	4.58	4323	2	38.35	32	6	0.33
Pdh_4	11	68	24	6	5.63	5061	2	13.65	33	4	0.63
Pdh_5	11	68	24	6	6.62	5751	3	55.70	30	3	0.45
Pdh_6	11	68	24	6	12.18	7378	2	117.01	50	9	0.33
Pdh_7	11	68	24	6	6.20	5719	2	124.73	46	10	0.58
Pdh_8	11	68	24	6	5.57	2867	2	15.36	27	5	0.59
Pdh_9	11	68	24	6	4.27	3182	2	34.53	25	6	0.50
Pdh_{10}	11	68	24	6	2.77	3241	2	61.58	23	5	0.58
$Polska_1$	12	36	66	6	1.55	1619	3	145.62	327	12	1.17
$Polska_2$	12	36	66	6	1.40	1523	3	68.35	197	6	1.02
$Polska_3$	12	36	66	6	2.01	1468	3	67.62	215	8	1.68
$Polska_4$	12	36	66	6	1.60	1477	4	95.25	252	9	1.58
$Polska_5$	12	36	66	6	1.37	1593	3	102.52	202	8	0.82
$Polska_6$	12	36	66	6	1.58	1704	3	234.06	414	15	0.92
$Polska_7$	12	36	66	6	0.97	1755	3	118.58	249	9	1.05
$Polska_8$	12	36	66	6	1.73	1596	4	96.66	357	13	1.64
$Polska_9$	12	36	66	6	1.38	1535	3	63.90	234	9	0.74
$Polska_{10}$	12	36	66	6	2.09	1612	3	74.21	244	7	0.96
$Ta1_1$	24	102	396	6	302.03	42939	3	885.94	710	6	16.29
$Ta1_2$	24	102	396	6	298.02	45994	3	1907.67	714	6	11.84
$Ta1_3$	24	102	396	6	273.87	46165	3	1852.12	973	6	13.21
$Ta1_4$	24	102	396	6	260.81	42083	3	1476.28	712	5	13.71
$Ta1_5$	24	102	396	6	280.87	47909	3	1324.80	723	5	13.69
$Ta1_6$	24	102	396	6	325.57	47528	3	2062.59	730	5	13.55
$Ta1_7$	24	102	396	6	266.30	44334	3	2615.23	883	7	12.80
$Ta1_8$	24	102	396	6	281.88	43662	4	1411.45	1014	8	15.01
$Ta1_9$	24	102	396	6	293.95	46275	4	1206.72	708	5	11.98
$Ta1_{10}$	24	102	396	6	269.40	44598	3	1145.76	719	6	12.47

Table 4.11: Results for SNDlib instances with relaxed Path formulation, relaxed Dantzig-Wolfe formulation and Relaxed Compact formulation.

Instances					PF				PF+VI				DW				DW+VI			
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap	t[s]	Col	Iter	Gap	t[s]	Col	Iter	Gap	t[s]	Col	Iter	Gap	t[s]	Col	Iter
$Abilene_1$	12	30	132	6	3.32	0.78	1718	3	47.56	3.19	1794	1	7.83	133.49	330	8	48.50	311.32	2097	35
$Abilene_2$	12	30	132	6	5.77	0.57	1718	2	247.07	2.29	1664	1	17.03	55.75	282	6	248.86	153.00	1982	29
$Abilene_3$	12	30	132	6	4.44	0.48	1687	2	89.31	1.86	1764	1	12.43	51.87	218	5	95.19	248.14	1748	35
$Abilene_4$	12	30	132	6	4.60	0.58	1698	2	168.61	1.92	1774	1	8.58	61.98	334	7	172.38	171.88	1809	30
$Abilene_5$	12	30	132	6	13.78	0.49	1683	3	478.48	0.69	1748	1	17.42	72.64	214	5	478.48	36.06	917	10
$Abilene_6$	12	30	132	6	5.12	0.48	1689	2	570.43	0.81	1752	1	11.30	52.01	232	6	570.43	236.55	2576	40
$Abilene_7$	12	30	132	6	5.50	0.42	1680	2	160.11	1.04	1740	1	8.92	57.97	220	7	161.50	201.55	2080	32
$Abilene_8$	12	30	132	6	1.04	0.53	1695	2	51.77	1.65	1770	1	5.05	61.45	437	7	55.91	225.81	2146	34
$Abilene_9$	12	30	132	6	8.70	0.52	1679	2	220.14	2.15	1762	1	19.77	57.34	308	5	222.68	266.56	3031	41
$Abilene_{10}$	12	30	132	6	3.61	0.65	1717	3	438.28	0.97	1782	1	10.52	43.41	240	6	438.57	188.90	2083	33
$Atlanta_1$	15	44	210	6	5.48	11.74	7392	2	15.93	45.32	7401	2	13.69	503.66	501	8	24.71	1845.42	2296	29
$Atlanta_2$	15	44	210	6	4.49	10.49	4563	4	27.17	24.47	4696	2	11.39	172.56	505	5	31.90	2303.34	3239	37
$Atlanta_3$	15	44	210	6	7.81	6.90	4929	3	112.10	30.09	5018	2	16.68	302.41	206	6	121.42	2422.32	5144	41
$Atlanta_4$	15	44	210	6	7.01	6.54	4996	3	79.83	15.64	5307	2	14.13	281.46	207	6	84.59	1779.73	2929	28
$Atlanta_5$	15	44	210	6	6.85	5.75	4151	3	99.32	4.08	6540	1	11.09	330.98	417	8	101.44	2782.86	5653	54
$Atlanta_6$	15	44	210	6	13.05	10.02	4930	3	93.83	21.75	4825	2	19.76	194.32	358	7	97.71	2833.35	4342	45
$Atlanta_7$	15	44	210	6	14.57	7.42	4985	3	113.69	30.85	4911	2	29.05	197.63	208	6	122.65	1383.84	3133	29
$Atlanta_8$	15	44	210	6	5.62	8.46	5046	3	49.84	26.84	5115	2	14.34	502.80	341	8	58.50	1602.83	2688	26
$Atlanta_9$	15	44	210	6	5.22	5.85	4851	4	70.18	24.97	4869	2	13.54	316.95	446	10	78.37	1409.37	3245	31
$Atlanta_{10}$	15	44	210	6	9.85	7.16	5046	3	105.56	31.01	4958	2	15.48	358.66	370	12	108.92	2025.36	3789	37
$Di-yuan_1$	11	84	22	6	1.37	13.51	7361	2	5.96	7.42	5386	2	4.98	228.98	117	10	9.37	1037.92	550	39
$Di-yuan_2$	11	84	22	6	3.43	13.42	8227	2	10.87	9.40	4817	2	12.85	281.78	136	16	20.01	2025.82	551	38
$Di-yuan_3$	11	84	22	6	0.39	27.97	14277	2	3.96	20.58	10295	2	3.81	384.60	135	11	–	TL	–	–
$Di-yuan_4$	11	84	22	6	1.58	22.35	13831	2	18.71	13.99	7343	2	9.36	408.55	99	10	–	TL	–	–
$Di-yuan_5$	11	84	22	6	0.97	22.14	13549	2	18.53	14.65	7724	2	5.82	216.27	103	9	–	TL	–	–
$Di-yuan_6$	11	84	22	6	1.11	16.21	10228	3	7.58	13.92	7263	2	7.20	355.63	143	15	12.16	2389.28	635	48
$Di-yuan_7$	11	84	22	6	0.91	19.95	13053	2	14.00	13.41	7036	2	3.86	237.66	114	10	17.26	2652.03	613	46
$Di-yuan_8$	11	84	22	6	1.74	16.37	12235	2	21.03	14.00	8060	2	4.75	167.02	86	9	–	TL	–	–
$Di-yuan_9$	11	84	22	6	0.39	34.84	17608	2	6.57	28.17	15888	2	9.88	567.53	110	10	–	TL	–	–
$Di-yuan_{10}$	11	84	22	6	1.94	20.52	13190	2	10.41	18.02	9622	2	6.25	151.02	101	8	–	TL	–	–
$France_1$	25	90	300	6	10.61	373.18	55964	3	44.53	990.71	56899	2	–	TL	–	–	–	TL	–	–
$France_2$	25	90	300	6	5.10	272.30	47514	5	17.22	1640.24	48485	2	–	TL	–	–	–	TL	–	–
$France_3$	25	90	300	6	6.24	333.01	54355	3	48.88	2769.04	57207	1	14.39	3586.68	500	6	–	TL	–	–
$France_4$	25	90	300	6	10.90	241.07	50397	3	29.38	500.92	52651	2	–	TL	–	–	–	TL	–	–
$France_5$	25	90	300	6	9.11	251.89	47509	3	32.25	1285.48	50936	2	–	TL	–	–	–	TL	–	–
$France_6$	25	90	300	6	6.61	326.58	55720	4	45.30	2558.16	57172	2	14.57	3157.17	451	7	–	TL	–	–
$France_7$	25	90	300	6	10.77	301.56	55379	3	21.58	958.75	57837	2	–	TL	–	–	4402.30	2165.80	1121	98
$France_8$	25	90	300	6	5.86	345.98	46745	4	20.79	420.02	50014	2	–	TL	–	–	–	TL	–	–
$France_9$	25	90	300	6	5.02	299.81	50328	3	14.56	405.80	49570	2	–	TL	–	–	4403.37	3427.42	2103	239
$France_{10}$	25	90	300	6	13.10	296.71	50403	3	74.57	503.32	51435	2	22.46	2700.80	409	5	–	TL	–	–

Table 4.12: Results for SNDlib instances with relaxed Path formulation, relaxed Dantzig-Wolfe formulation with and without valid inequalities.

Instances					PF				PF+VI				DW				DW+VI			
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap	t[s]	Col	Iter	Gap	t[s]	Col	Iter	Gap	t[s]	Col	Iter	Gap	t[s]	Col	Iter
$Geant_1$	22	72	462	6	2.83	382.87	50877	3	484.94	400.56	41009	2	14.47	1510.34	414	4	–	TL	–	–
$Geant_2$	22	72	462	6	6.65	298.15	48652	3	371.93	707.50	45857	2	–	TL	–	–	–	TL	–	–
$Geant_3$	22	72	462	6	7.30	290.98	47741	3	319.00	1077.01	44656	2	16.21	1310.94	430	3	–	TL	–	–
$Geant_4$	22	72	462	6	2.59	269.43	45164	4	53.49	723.15	45379	2	6.00	1848.17	1306	7	–	TL	–	–
$Geant_5$	22	72	462	6	4.42	303.14	49197	3	431.34	381.11	34961	2	17.45	2429.54	864	5	–	TL	–	–
$Geant_6$	22	72	462	6	3.92	271.81	45707	4	157.94	728.84	34729	2	11.27	2346.37	784	5	–	TL	–	–
$Geant_7$	22	72	462	6	3.77	283.34	49418	3	130.91	2086.82	49933	2	–	TL	–	–	–	TL	–	–
$Geant_8$	22	72	462	6	7.77	293.47	51436	3	182.56	1647.43	50908	2	14.24	2779.17	436	5	–	TL	–	–
$Geant_9$	22	72	462	6	2.58	270.50	46695	4	110.81	2274.09	43065	2	5.29	1838.10	704	6	–	TL	–	–
$Geant_{10}$	22	72	462	6	5.15	286.78	50382	3	457.67	383.16	46688	2	18.57	2312.52	423	4	–	TL	–	–
$Newyork_1$	16	98	240	6	3.55	823.65	114459	3	20.23	873.57	117283	2	–	TL	–	–	–	TL	–	–
$Newyork_2$	16	98	240	6	4.54	616.65	107859	3	36.77	1371.14	113142	2	–	TL	–	–	–	TL	–	–
$Newyork_3$	16	98	240	6	2.11	649.80	108805	3	17.36	1407.36	113158	2	–	TL	–	–	–	TL	–	–
$Newyork_4$	16	98	240	6	3.78	635.65	109429	3	52.18	1260.25	120168	2	–	TL	–	–	–	TL	–	–
$Newyork_5$	16	98	240	6	2.74	756.04	115911	3	33.88	1248.91	119366	2	–	TL	–	–	–	TL	–	–
$Newyork_6$	16	98	240	6	3.46	634.24	106288	3	36.67	2534.42	115101	2	–	TL	–	–	–	TL	–	–
$Newyork_7$	16	98	240	6	2.88	691.47	113000	3	82.48	755.64	115811	2	–	TL	–	–	–	TL	–	–
$Newyork_8$	16	98	240	6	2.36	675.41	114708	3	14.78	1055.77	118820	2	–	TL	–	–	–	TL	–	–
$Newyork_9$	16	98	240	6	3.87	716.51	121537	3	20.95	1403.32	124648	2	–	TL	–	–	–	TL	–	–
$Newyork_{10}$	16	98	240	6	3.49	604.96	112781	3	–	TL	–	–	–	TL	–	–	–	TL	–	–
$Nobel-eu_1$	28	82	378	6	5.44	982.64	104788	3	–	TL	–	–	–	TL	–	–	–	TL	–	–
$Nobel-eu_2$	28	82	378	6	4.92	947.06	109253	4	–	TL	–	–	–	TL	–	–	–	TL	–	–
$Nobel-eu_3$	28	82	378	6	8.76	803.63	100804	3	–	TL	–	–	–	TL	–	–	–	TL	–	–
$Nobel-eu_4$	28	82	378	6	3.30	746.16	79113	4	23.28	1222.18	88498	2	4347.38	1281.69	147	27	–	TL	–	–
$Nobel-eu_5$	28	82	378	6	2.14	950.99	98675	3	–	TL	–	–	–	TL	–	–	–	TL	–	–
$Nobel-eu_6$	28	82	378	6	5.41	716.78	92789	3	–	TL	–	–	–	TL	–	–	–	TL	–	–
$Nobel-eu_7$	28	82	378	6	6.11	781.82	102426	3	–	TL	–	–	–	TL	–	–	–	TL	–	–
$Nobel-eu_8$	28	82	378	6	3.47	597.47	84123	3	80.50	1396.74	86524	2	–	TL	–	–	–	TL	–	–
$Nobel-eu_9$	28	82	378	6	9.10	582.09	88657	3	107.64	1326.34	89365	2	–	TL	–	–	–	TL	–	–
$Nobel-eu_{10}$	28	82	378	6	2.57	681.54	92449	3	–	TL	–	–	–	TL	–	–	–	TL	–	–
$Nobel-ger_1$	17	52	121	6	3.26	10.18	5900	3	79.45	5.06	10180	2	10.21	443.90	434	8	–	TL	–	–
$Nobel-ger_2$	17	52	121	6	7.71	12.50	5705	3	129.04	9.63	7903	2	19.61	749.74	459	11	–	TL	–	–
$Nobel-ger_3$	17	52	121	6	4.44	8.82	5783	3	160.83	9.38	9050	1	10.95	312.16	359	8	–	TL	–	–
$Nobel-ger_4$	17	52	121	6	4.05	8.58	5028	3	45.33	7.95	8582	2	16.80	642.54	463	8	–	TL	–	–
$Nobel-ger_5$	17	52	121	6	4.15	10.36	5724	3	57.83	7.49	9388	2	11.12	492.78	572	9	–	TL	–	–
$Nobel-ger_6$	17	52	121	6	2.82	13.77	5809	4	19.48	39.88	5716	2	7.18	605.83	470	10	–	TL	–	–
$Nobel-ger_7$	17	52	121	6	6.38	11.51	6032	3	32.35	14.71	8215	2	12.97	416.67	415	7	–	TL	–	–
$Nobel-ger_8$	17	52	121	6	9.33	10.19	5300	3	105.01	31.42	6854	2	21.36	662.80	318	7	–	TL	–	–
$Nobel-ger_9$	17	52	121	6	4.45	8.42	6026	3	56.46	4.98	9415	2	16.28	727.86	624	11	–	TL	–	–
$Nobel-ger_{10}$	17	52	121	6	4.89	10.99	6051	3	45.50	30.36	6504	2	13.23	703.07	457	11	–	TL	–	–

Table 4.13: Results for SNDlib instances with relaxed Path formulation, relaxed Dantzig-Wolfe formulation with and without valid inequalities.

Instances					PF				PF+VI				DW				DW+VI			
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap	t[s]	Col	Iter	Gap	t[s]	Col	Iter	Gap	t[s]	Col	Iter	Gap	t[s]	Col	Iter
$Nobel-us_1$	14	42	91	6	5.06	1.76	2013	3	59.59	2.31	3468	1	14.10	264.27	325	12	62.70	2150.41	5497	89
$Nobel-us_2$	14	42	91	6	3.13	2.73	2385	3	60.29	1.80	3533	2	7.29	243.46	293	11	36.97	2367.27	1251	69
$Nobel-us_3$	14	42	91	6	4.54	1.87	2322	3	64.64	5.46	3736	1	8.33	82.76	291	7	69.75	3486.39	5336	89
$Nobel-us_4$	14	42	91	6	3.19	2.27	1967	4	17.14	6.22	2178	2	6.68	112.86	307	7	20.40	921.79	1532	39
$Nobel-us_5$	14	42	91	6	4.59	1.75	2271	3	29.67	20.94	2342	2	10.95	212.29	270	8	35.84	2214.80	2884	54
$Nobel-us_6$	14	42	91	6	1.95	5.01	2482	3	11.94	11.95	2596	2	5.29	207.01	354	9	16.14	929.29	1283	29
$Nobel-us_7$	14	42	91	6	6.14	2.06	2402	3	94.42	4.48	2438	2	12.62	121.12	195	6	101.48	1983.75	2897	57
$Nobel-us_8$	14	42	91	6	4.62	2.59	2483	3	102.69	3.24	3342	2	13.38	233.45	239	10	109.80	1918.53	2381	44
$Nobel-us_9$	14	42	91	6	3.40	1.96	2160	3	40.76	8.10	2246	2	16.03	92.59	273	6	50.88	1179.57	1923	39
$Nobel-us_{10}$	14	42	91	6	4.11	1.66	2324	3	53.70	4.08	3446	1	16.55	192.17	234	9	65.35	1904.15	2841	56
Pdh_1	11	68	24	6	4.52	1.69	1776	2	68.17	1.64	1792	2	5.75	14.57	17	4	68.80	284.78	71	27
Pdh_2	11	68	24	6	5.97	8.28	6460	2	37.70	6.95	6593	2	8.80	117.73	31	8	39.07	1165.68	157	38
Pdh_3	11	68	24	6	5.52	4.58	4323	2	77.91	4.51	4391	2	7.66	38.35	32	6	78.66	552.20	162	37
Pdh_4	11	68	24	6	4.70	5.63	5061	2	52.84	4.05	3306	2	5.91	13.65	33	4	53.71	1030.25	262	59
Pdh_5	11	68	24	6	3.83	6.62	5751	3	43.96	6.56	5951	2	8.42	55.70	30	3	–	TL	–	–
Pdh_6	11	68	24	6	4.13	12.18	7378	2	62.26	5.98	5141	2	7.47	117.01	50	9	–	TL	–	–
Pdh_7	11	68	24	6	4.20	6.20	5719	2	67.21	6.18	6618	2	5.83	124.73	46	10	70.56	1622.56	113	28
Pdh_8	11	68	24	6	8.28	5.57	2867	2	66.92	2.58	1880	2	10.50	15.36	27	5	67.72	343.62	167	50
Pdh_9	11	68	24	6	6.99	4.27	3182	2	41.28	3.08	3147	2	10.39	34.53	25	6	42.77	1431.29	97	29
Pdh_{10}	11	68	24	6	7.04	2.77	3241	2	88.07	2.70	3374	2	9.25	61.58	23	5	91.85	560.51	63	13
$Polska_1$	12	36	66	6	1.27	1.55	1619	3	2.96	1.32	2155	2	8.51	145.62	327	12	10.06	392.43	1007	28
$Polska_2$	12	36	66	6	1.42	1.40	1523	3	13.45	4.13	1574	2	15.42	68.35	197	6	26.69	269.23	893	25
$Polska_3$	12	36	66	6	1.45	2.01	1468	3	4.26	4.59	1520	2	5.34	67.62	215	8	8.37	222.08	660	20
$Polska_4$	12	36	66	6	2.09	1.60	1477	4	9.24	1.91	1989	2	11.39	95.25	252	9	19.16	392.78	777	29
$Polska_5$	12	36	66	6	2.71	1.37	1593	3	10.18	1.07	2290	1	14.23	102.52	202	8	22.51	307.51	851	24
$Polska_6$	12	36	66	6	1.55	1.58	1704	3	3.82	9.60	1698	2	14.93	234.06	414	15	18.77	425.52	880	25
$Polska_7$	12	36	66	6	1.50	0.97	1755	3	7.73	4.55	1550	2	9.72	118.58	249	9	16.15	337.75	731	22
$Polska_8$	12	36	66	6	0.97	1.73	1596	4	2.23	9.84	1639	2	5.12	96.66	357	13	7.69	267.68	783	23
$Polska_9$	12	36	66	6	1.37	1.38	1535	3	5.54	1.09	2224	1	7.02	63.90	234	9	12.71	288.80	806	24
$Polska_{10}$	12	36	66	6	1.62	2.09	1612	3	5.89	5.29	1673	2	10.15	74.21	244	7	13.64	360.65	876	28
$Ta1_1$	24	102	396	6	11.97	302.03	42939	3	267.87	762.29	40326	2	17.65	885.94	710	6	–	TL	–	–
$Ta1_2$	24	102	396	6	14.68	298.02	45994	3	255.47	1994.10	47596	2	25.60	1907.67	714	6	–	TL	–	–
$Ta1_3$	24	102	396	6	10.04	273.87	46165	3	298.28	891.75	46629	2	16.82	1852.12	973	6	–	TL	–	–
$Ta1_4$	24	102	396	6	9.38	260.81	42083	3	–	TL	0	0	20.39	1476.28	712	5	–	TL	–	–
$Ta1_5$	24	102	396	6	8.59	280.87	47909	3	374.72	1335.16	46727	2	15.99	1324.80	723	5	–	TL	–	–
$Ta1_6$	24	102	396	6	9.75	325.57	47528	3	–	TL	0	0	17.38	2062.59	730	5	–	TL	–	–
$Ta1_7$	24	102	396	6	8.15	266.30	44334	3	103.32	1483.64	44707	2	11.63	2615.23	883	7	–	TL	–	–
$Ta1_8$	24	102	396	6	4.41	281.88	43662	4	114.95	1107.99	44309	2	7.85	1411.45	1014	8	–	TL	–	–
$Ta1_9$	24	102	396	6	9.82	293.95	46275	4	–	TL	0	0	14.32	1206.72	708	5	–	TL	–	–
$Ta1_{10}$	24	102	396	6	8.31	269.40	44598	3	217.06	1410.97	44756	2	16.16	1145.76	719	6	–	TL	–	–

Table 4.14: Results for SNDlib instances with relaxed Path formulation, relaxed Dantzig-Wolfe formulation with and without valid inequalities.

Instances					PF+VI				DW+VI				C	AB
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap	Col	Nodes	Iter	Gap	Col	Nodes	Iter	Gap	Gap
$Abilene_1$	12	30	132	6	0.83	1794	580	579	1.47	3662	128	380	0.00	1.43
$Abilene_2$	12	30	132	6	0.00	1713	1790	1790	0.38	6054	196	694	1.35	2.52
$Abilene_3$	12	30	132	6	1.09	1764	758	758	3.34	3425	232	491	0.00	1.38
$Abilene_4$	12	30	132	6	0.00	1774	910	909	1.74	4634	272	614	1.83	1.43
$Abilene_5$	12	30	132	6	0.11	1748	13998	13998	0.00	1224	336	362	1.74	1.90
$Abilene_6$	12	30	132	6	0.00	1752	1330	1329	0.47	5839	324	723	2.44	2.04
$Abilene_7$	12	30	132	6	0.00	1740	940	940	0.50	4719	276	654	1.76	1.09
$Abilene_8$	12	30	132	6	0.00	1770	560	559	2.75	5781	156	529	1.29	0.69
$Abilene_9$	12	30	132	6	0.00	1762	1798	1797	0.71	8113	128	539	0.36	1.50
$Abilene_{10}$	12	30	132	6	0.30	1782	2512	2511	0.00	3059	84	160	2.73	1.88
$Atlanta_1$	15	44	210	6	1.73	7452	86	89	8.93	2452	8	53	0.00	0.96
$Atlanta_2$	15	44	210	6	0.91	4830	170	173	4.32	3376	4	51	0.00	0.38
$Atlanta_3$	15	44	210	6	0.53	5097	106	108	5.02	5462	4	58	0.00	0.96
$Atlanta_4$	15	44	210	6	0.38	5491	194	196	3.01	3238	8	50	0.00	0.51
$Atlanta_5$	15	44	210	6	0.24	6540	88	88	1.31	5822	2	66	0.00	1.61
$Atlanta_6$	15	44	210	6	2.32	5114	128	130	4.27	4546	4	55	0.00	1.75
$Atlanta_7$	15	44	210	6	5.29	5087	122	132	9.82	4605	6	68	0.00	4.35
$Atlanta_8$	15	44	210	6	0.78	5285	126	127	6.65	2937	8	51	0.00	0.47
$Atlanta_9$	15	44	210	6	1.08	5071	110	113	5.96	4128	8	69	0.00	0.18
$Atlanta_{10}$	15	44	210	6	0.35	5215	154	159	1.97	4434	6	61	0.00	0.78
$Di-yuan_1$	11	84	22	6	0.00	9885	562	582	0.68	802	6	88	0.64	0.97
$Di-yuan_2$	11	84	22	6	1.86	9765	564	584	4.77	605	2	54	0.00	3.05
$Di-yuan_3$	11	84	22	6	0.00	15128	374	393	–	–	–	–	2.75	2.90
$Di-yuan_4$	11	84	22	6	0.00	10418	458	468	–	–	–	–	0.68	1.30
$Di-yuan_5$	11	84	22	6	0.00	13117	474	486	–	–	–	–	1.71	2.82
$Di-yuan_6$	11	84	22	6	0.00	10465	388	405	2.80	735	4	63	0.44	1.65
$Di-yuan_7$	11	84	22	6	0.59	11694	388	402	0.00	678	2	58	0.64	1.05
$Di-yuan_8$	11	84	22	6	0.00	9035	372	380	–	–	–	–	0.37	1.00
$Di-yuan_9$	11	84	22	6	0.00	17718	264	268	–	–	–	–	1.25	6.39
$Di-yuan_{10}$	11	84	22	6	7.62	13027	346	368	–	–	–	–	0.00	5.67
$Nobel-ger_1$	17	52	121	6	0.00	10394	192	192	–	–	–	–	0.46	0.93
$Nobel-ger_2$	17	52	121	6	0.91	8428	142	142	–	–	–	–	0.00	0.27
$Nobel-ger_3$	17	52	121	6	0.00	9050	62	61	–	–	–	–	0.61	0.84
$Nobel-ger_4$	17	52	121	6	0.55	8811	152	153	–	–	–	–	0.00	0.39
$Nobel-ger_5$	17	52	121	6	0.00	9507	174	175	–	–	–	–	0.72	1.29
$Nobel-ger_6$	17	52	121	6	0.19	6016	80	83	–	–	–	–	0.00	0.50
$Nobel-ger_7$	17	52	121	6	1.95	8938	176	180	–	–	–	–	0.19	0.00
$Nobel-ger_8$	17	52	121	6	1.06	7096	88	88	–	–	–	–	0.32	0.00
$Nobel-ger_9$	17	52	121	6	0.50	9447	122	122	–	–	–	–	1.50	0.00
$Nobel-ger_{10}$	17	52	121	6	0.07	6697	116	117	–	–	–	–	0.00	0.09

Table 4.15: Bounds improvement results for SNDlib instances with Branch-and-Price algorithms for the Path formulation and the Dantzig-Wolfe formulation compared to the compact formulation and the automatic Benders.

Instances					PF+VI				DW+VI				C	AB
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap	Col	Nodes	Iter	Gap	Col	Nodes	Iter	Gap	Gap
$Nobel-us_1$	14	42	91	6	0.00	3468	160	159	1.94	6321	6	133	0.45	0.38
$Nobel-us_2$	14	42	91	6	13.60	3551	318	322	0.00	1674	2	98	15.34	15.56
$Nobel-us_3$	14	42	91	6	1.10	3736	248	247	4.13	5360	2	93	1.53	0.00
$Nobel-us_4$	14	42	91	6	0.00	2276	214	221	2.70	1928	20	120	0.73	0.84
$Nobel-us_5$	14	42	91	6	0.77	2488	226	232	4.72	3179	4	80	0.00	0.78
$Nobel-us_6$	14	42	91	6	0.00	2736	182	188	3.14	1447	16	82	0.38	1.30
$Nobel-us_7$	14	42	91	6	0.00	2492	338	339	3.59	3098	6	86	0.63	0.49
$Nobel-us_8$	14	42	91	6	0.00	3473	454	458	3.58	2823	4	68	1.02	1.37
$Nobel-us_9$	14	42	91	6	0.20	2368	254	256	6.97	2424	12	90	0.00	0.38
$Nobel-us_{10}$	14	42	91	6	0.70	3446	180	179	8.38	3736	6	94	1.42	0.00
Pdh_1	11	68	24	6	0.00	1822	12166	12167	1.14	170	196	306	6.45	6.64
Pdh_2	11	68	24	6	0.00	6593	2272	2273	0.80	277	20	106	4.71	4.16
Pdh_3	11	68	24	6	0.00	4400	4846	4846	0.31	340	20	151	4.90	4.83
Pdh_4	11	68	24	6	0.00	4518	2492	2495	0.64	493	10	145	4.24	3.90
Pdh_5	11	68	24	6	0.00	5997	4266	4266	–	–	–	–	6.27	6.00
Pdh_6	11	68	24	6	0.00	6218	1832	1833	–	–	–	–	4.99	5.19
Pdh_7	11	68	24	6	0.00	6671	10086	10087	1.94	149	8	54	3.84	3.94
Pdh_8	11	68	24	6	0.00	3332	3820	3822	0.57	559	40	304	4.84	4.54
Pdh_9	11	68	24	6	0.00	4759	6204	6206	1.36	137	10	67	4.82	5.19
Pdh_{10}	11	68	24	6	0.00	3408	10444	10445	2.07	107	28	66	5.14	5.17
$Polska_1$	12	36	66	6	0.00	2210	1100	1106	4.42	1549	44	176	0.29	0.67
$Polska_2$	12	36	66	6	0.00	1697	568	574	11.07	1355	80	236	2.89	1.58
$Polska_3$	12	36	66	6	0.00	1615	898	905	2.71	1020	114	250	0.88	1.14
$Polska_4$	12	36	66	6	0.00	2030	968	974	8.31	1037	80	197	2.21	0.96
$Polska_5$	12	36	66	6	0.00	2290	1414	1413	9.91	1531	60	214	1.49	0.69
$Polska_6$	12	36	66	6	0.00	1802	636	645	14.28	1264	46	153	4.33	1.45
$Polska_7$	12	36	66	6	0.00	1624	1220	1228	7.55	908	82	175	2.77	1.43
$Polska_8$	12	36	66	6	0.00	1698	616	620	3.84	1171	64	186	2.41	2.29
$Polska_9$	12	36	66	6	0.00	2224	914	913	6.09	1474	56	208	2.07	1.39
$Polska_{10}$	12	36	66	6	0.00	1734	456	466	5.76	1372	82	218	0.93	1.39

Table 4.16: Bounds improvement results for SNDlib instances with Branch-and-Price algorithms for the Path formulation and the Dantzig-Wolfe formulation compared to the compact formulation and the automatic Benders.

4.7 Conclusions

In this chapter we have proposed two extended formulations for solving the Virtual Network Functions Placement and Routing problem. The variables of the first formulation (denoted by PF) are latency-constrained paths, whereas the variables of the second formulation (denoted by DW) are latency-constrained paths that also embed the information regarding the function installations at their nodes. In order to strengthen the LP-bounds, we have proposed several families of valid inequalities for both formulations. Their benefits have been computationally demonstrated on a set of instances derived from telecommunication networks. We have presented a branching scheme for each formulation and have developed and implemented the associated Branch-and-Price algorithms. The latter ones are computationally compared with the compact MIP formulation presented in Chapter 2 and the automatic Benders decomposition applied to it. The obtained results have shown that there is a trade-off between the quality of global lower bounds and the time needed to solve the LP-relaxations. Whereas the overall best global lower bounds can be obtained by the model DW, its CPU time is sacrificed by the expensive MIP-based pricing procedure. Our results show that the full potential of extended formulations is still to be exploited. This can be done by developing problem-tailored pricing procedures, alternative branching schemes, or more advanced heuristics that can be used to initialize the upper bounds, or for solving the pricing problem.

Chapter 5

Benders reformulation for the node-capacitated VNFPRP

In this chapter, we study the Uncapacitated Virtual Network Functions Placement and Routing problem, for which node-capacity, VNF-capacity, and conflict constraints are relaxed. We propose some valid inequalities and a Benders decomposition scheme, allowing us to solve the problem, within a Branch-and-Benders-Cut algorithm. Finally, we provide computational results to compare some variants of the proposed formulations and show significant improvements over an approach based on automatic Benders cuts (generated with Cplex).

Contents

In this chapter we assume that VNF capacity is sufficiently large, and that the only capacity we have to consider at the nodes is the number of VNFs that we can install. We call this problem variant the node-capacitated VNFPRP . To improve the capabilities of finding exact solutions for larger instances of practical relevance, we exploit some theoretical properties of a compact model and develop a Benders decomposition approach, in which the VNF placement problem is treated at the master level and the routing problem (which becomes decomposable per commodity) is solved at the subproblem level. We also propose several new families of valid inequalities and use the path-based MILP formulation to provide heuristic solutions. These elements are combined into an efficient Branch-and-Benders-Cut framework which is capable of beating an off-the-shelf solver (in terms of the CPU time and the overall solution quality) on a set of realistic and random benchmark instances considered in our computational study.

Outline of the chapter The chapter is organized as follows. In Section 5.1, we propose a compact MILP formulation based on the model proposed in Chapter 2, study its theoretical properties and introduce new valid inequalities. Section 5.2 is devoted to the Benders reformulation, whereas the Uncapacitated MILP-based heuristic, already defined in Chapter 3, is given in Section 5.3. In Section 5.4, detailed computational results are provided, and some concluding remarks are derived in Section 5.5.

In this chapter, we consider two variants for the problem, in the first variant, we relax only the VNF-capacity constraints, and we call the problem the *node-capacitated and conflict-constrained VNFPRP*. In the second variant, we relax VNF-capacity, node-capacity and conflict constraints, and we call it the *Uncapacitated VNFPRP*.

5.1 Adapted compact MILP formulation

In this section, we first adapt the compact (i.e., polynomial in size) MILP formulation already introduced in Chapter 2 to model the node-capacitated and conflict constrained variant and the uncapacitated variant and the of the VNFPRP. We also show some favorable theoretical properties of this model that allow us to decompose it using Benders decomposition. We conclude this section by proposing three additional families of strengthening inequalities.

5.1.1 MILP formulation for the uncapacitated VNFPRP

In the compact MILP formulation associated with the Uncapacitated version of the problem, the binary decision variables (x, y, w, t) are the same as in the compact formulation introduced in Chapter 2. Similarly, we keep the same set of constraints except for the VNF-capacity, the node-capacity, and the conflict constraints, which are relaxed. The objective function aims to minimize the node activation and the VNF installation costs.

The Uncapacitated VNFPR can then be modeled as follows:

$$(P): \quad \min \quad \sum_{k \in C} \sum_{u \in N} \sum_{f \in F^k} \psi_u^f \ y_u^{fk} + \sum_{u \in N} \psi_u \ w_u \tag{5.1}$$

$$2.2 - 2.3, \ 2.7 - 2.12$$

$$y_u^{fk} \leq w_u \qquad k \in C, f \in F^k, u \in N \tag{5.2}$$

$$y_u^{fk}, x_u^{fk} \in \{0,1\} \qquad u \in N, f \in F, k \in C \tag{5.3}$$

$$w_u \in \{0,1\} \qquad u \in N \tag{5.4}$$

$$t_{uv}^k \in \{0,1\} \qquad (u,v) \in A, k \in C \tag{5.5}$$

Inequalities (5.7) link y and w variables, enforcing that a function can be installed at a node, only if the node is activated. Constraints (5.8)-(5.10) are the integrity constraints.

5.1.2 MILP formulation for the node-capacitated and conflict constrained VNFPRP

In the compact MILP formulation associated with the node-capacitated and conflict constrained version of the problem, the binary decision variables (x, y, w, t) are the same as in the compact formulation introduced in Chapter 2. Similarly, we keep the same set of constraints except for the VNF-capacity constraints, which are relaxed. The objective function aims to minimize the VNF installation and the node activation costs.

The node-capacitated and conflict constrained VNFPR can then be modeled as follows:

$$(P): \quad \min \quad \sum_{k\in C}\sum_{u\in N}\sum_{f\in F^k} \psi_u^f \;\; y_u^{fk} + \sum_{u\in N} \psi_u \;\; w_u \tag{5.6}$$

$$2.2-2.4,\; 2.6-2.12$$

$$\begin{aligned} y_u^{fk} &\leq w_u && k\in C, f\in F^k, u\in N && (5.7)\\ y_u^{fk}, x_u^{fk} &\in \{0,1\} && u\in N, f\in F, k\in C && (5.8)\\ w_u &\in \{0,1\} && u\in N && (5.9)\\ t_{uv}^k &\in \{0,1\} && (u,v)\in A, k\in C && (5.10) \end{aligned}$$

5.1.3 Unsplittable routing paths

Integrality constraints $t_{uv}^k \in \{0,1\}$ guarantee that the s_k-d_k flow cannot be split, i.e., that there is a single path used for routing the flow. In the following, we show that this constraint can be relaxed. Let (P') denote the model (P) in which integrality constraints (5.10) are replaced by

$$t_{uv}^k \geq 0, \quad k\in C, (u,v)\in A.$$

The major result of this section, which will also allow us to apply the Benders decomposition scheme is summarized as follows:

Theorem 5.1 *If the compact formulation (P') has an optimal fractional solution then it must necessarily admit an integer solution with the same objective value.*

The proof of this theorem is provided at the end of the section. We start by proving Lemma 5.2, in which we focus on a single commodity, and assume that the capacity upper bounds on arcs corresponding to constraints (2.7) and capacity lower bounds on nodes corresponding to constraints (2.10) are pre-specified.

Lemma 5.2 *Let $\hat{c}_{uv} \in \mathbb{N}\cup\{0\}$ be an arbitrary capacity function on arcs $(u,v)\in A$, $\hat{c}_u \in \{0,1\}$ be an arbitrary lower bound capacity on nodes $u\in N$ and $\hat{l}\in\mathbb{N}$ a flow-latency limit for a flow to be sent from a given source $s\in N$ to a destination $d\in N$, $s\neq d$. The following system of linear inequalities is either infeasible, or it admits a feasible binary solution.*

$$\sum_{(u,v)\in A} t_{uv} - \sum_{(v,u)\in A} t_{vu} = \begin{cases} -1 & \text{if } u = d, \\ 1 & \text{if } u = s, \\ 0 & \text{otherwise.} \end{cases} \qquad \forall u \in N \tag{5.11}$$

$$\sum_{(u,v)\in A} t_{uv} l_{uv} \leq \hat{l} \tag{5.12}$$

$$0 \leq t_{uv} \leq \hat{c}_{uv} \qquad \forall (u,v) \in A \tag{5.13}$$

$$\sum_{(v,u)\in A} t_{vu} \geq \hat{c}_u \qquad \forall u \in N \tag{5.14}$$

Proof. Assume that the system of linear inequalities is feasible, and let $\hat{t}$ be its fractional feasible solution. By the flow decomposition theorem, we can decompose the solution $\hat{t}$ into $\hat{t}_1, \hat{t}_2, \ldots, \hat{t}_Q$ associated with fractional paths $\mathcal{P}' = \{p_1, p_2, \ldots, p_Q\}$, each of them carrying r_q units of flow, such that $\sum_{q=1}^{Q} r_q = 1$. Let $\hat{t}_{q,uv}$ be the arc variables that compose $\hat{t}_q$ and the path p_q and let $\bar{t}_{q,uv} \in \{0,1\}$ be obtained by applying the ceiling function to values of $\hat{t}_{q,uv}$, i.e., $\bar{t}_{q,uv} = \lceil \hat{t}_{q,uv} \rceil$, for all $(u,v) \in p_q$, $1 \leq q \leq Q$.

For each node $u \in N$ such that $\hat{c}_u = 1$, by the feasibility of $\hat{t}$, all paths $p_q \in \mathcal{P}'$ must pass through node u.

The flow preservation constraints (5.11) are clearly satisfied by each $\hat{t}^k_q$, $1 \leq q \leq Q$. It is easy to see that the flow preservation constrains remain satisfied for each individual path defined by $\bar{t}^k_q$, $1 \leq q \leq Q$. Hence, to satisfy constraints (5.11), (5.13) and (5.14), one could take any of the paths $p_1, \ldots, p_Q$ as a feasible binary solution.

What remains to show is that at least one of the paths $\{p_1, \ldots, p_Q\}$ must also satisfy the latency constraint (5.12). We observe that $\hat{t}_{q,uv}$ has the same value for each arc (u,v) composing the path p_q, for each $1 \leq q \leq Q$, more precisely, this value is equal to r_q defined above. As the sum of the flow is integer, at source and destination nodes we must have at least two paths with fractional flows. Suppose that none of the paths from $\mathcal{P}'$ satisfy the latency constraints, i.e.:

$$l(p) > \hat{l} \quad \forall p \in \mathcal{P}',$$

where $l(p) = \sum_{uv\in p} l_{uv}$ denotes the latency of the path $p \in \mathcal{P}'$. Then:

$$\forall p_q \in \mathcal{P}' : \quad l(p_q) > \hat{l} \quad \Rightarrow \quad \sum_{p_q \in \mathcal{P}'} r_q l(p_q) > \sum_{p_q \in \mathcal{P}'} r_q \hat{l}$$

Since from constraints (5.11) we have $\sum_{p_q \in \mathcal{P}'} r_q = 1$, it follows that:

$$\sum_{p_q \in \mathcal{P}'} r_q l(p_q) > \hat{l},$$

which contradicts the fact that the system (5.11)-(5.14) admits a fractional feasible solution. □

Proof. *[Theorem 5.1]* Let $\hat{X} = (\hat{x}, \hat{y}, \hat{w}, \hat{t})$ be an optimal solution of (P') such that there exists at least one commodity $k \in C$ for which the flow $\hat{t}^k$ is fractional. We will show how to construct a purely integer solution $\bar{X} = (\hat{x}, \hat{y}, \hat{w}, \bar{t})$ which is also feasible for (P'). As the components $\hat{y}$ and $\hat{w}$ that appear in the objective function with non-zero coefficients are the same for both $\hat{X}$ and $\bar{X}$, it follows that the two solutions will have the same objective value.

Let $k \in C$ be a commodity such that in the solution $\hat{X}$, the associated vector $\hat{t}^k$ contains fractional values. In that case, by setting $\hat{l} := l_k$, $s := s_k$, $d := d_k$ and

$$\hat{c}_{uv} := \min_{f \in F^k} \{\hat{y}_v^{fk} + 1 - \hat{x}_v^{fk} + \hat{x}_u^{fk}\} \qquad \forall (u,v) \in A \qquad (5.15)$$

$$\hat{c}_u := \max_{f \in F^k} \{\hat{y}_u^{fk}\} \qquad \forall u \in N \qquad (5.16)$$

we obtain the linear inequality system (5.11)-(5.14).

By Lemma 5.2, among the fractional paths composing the flow $\hat{t}^k$, there exists at least one (integer) path which can be used to substitute $\hat{t}^k$. This procedure can be repeated for all commodities $k \in C$ that admit a fractional flow solution $\hat{t}^k$ in $\hat{X}$, without changing the structure of the binary component $(\hat{x}, \hat{y}, \hat{w})$, due to the fact that the flows are routed separately for each commodity.

■

Corollary 5.3 *Without loss of generality, constraints $t_{uv}^k \in \{0,1\}$, for all $(u,v) \in A$, $k \in C$ can be relaxed into $t_{uv}^k \geq 0$.*

5.1.4 Strengthening inequalities

In the following, we derive three families of valid inequalities that can strengthen the quality of the LP-bounds of the formulation (P). The two first families exploit the precedence constraints imposed for a given commodity, the last one exploit the node-capacity constraints imposed for each node.

Node-Precedence inequalities

Theorem 5.4 *The constraints (5.17) are valid for (P):*

$$y_u^{f_l k} \geq y_u^{f_i k} + y_u^{f_j k} - 1, \qquad k \in C,\ u \in N \setminus \{s_k\},$$
$$i = 1, 2, \ldots, |F^k| - 2, \qquad j = i+2, \ldots, |F^k|, \qquad i < l < j \tag{5.17}$$

Proof. For given positions i and j in the ordering of F^k, with $i = 1, 2, \ldots, |F^k| - 2$ and $j = i+2, \ldots, |F^k|$, if both functions f_i and f_j are installed at node u for commodity k, then to satisfy the precedence constraints, each function f_l between f_i and f_j, with $l = i+1, i+2, \ldots, j-1$, must be installed at node u as well, otherwise we allow creation of cycles, and hence the violation of precedence constraints (see Figure 5.1). ■

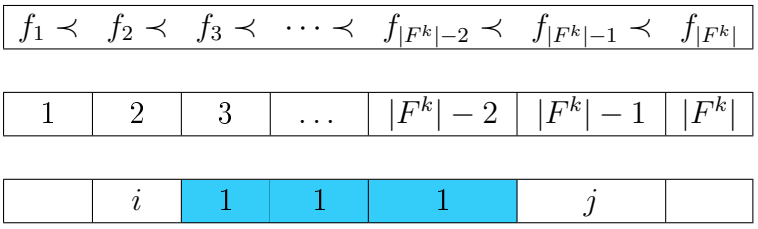


Figure 5.1: Explanation of node-precedence inequalities.

We notice that even though these cuts are polynomial in number (there are $O(|C||F|^3|N|)$ of these inequalities), they may impose a significant burden to the MILP solvers because of their size. We therefore also consider the aggregated version of these constraints.

Aggregated node-precedence inequalities

Theorem 5.5 *The linear inequalities (5.18) are valid for (P):*

$$\sum_{l=i+1}^{j-1} y_u^{f_l k} \geq (j-i-1)(y_u^{f_i k} + y_u^{f_j k} - 1) \qquad k \in C,\ u \in N \setminus \{s_k\},$$
$$i = 1, 2, \ldots, |F^k| - 2, \quad j = i+2, \ldots, |F^k| \tag{5.18}$$

Proof. The constraints are obtained by summing up (5.18) over all $l \in \{i+1, \ldots, j-1\}$, and hence they are valid for (P). ■

Node-capacity inequalities

Theorem 5.6 *The linear inequalities* (5.19) *are valid for* (P)*:*

$$\sum_{l=1}^{i-c_u} y_u^{f_l k} + \sum_{l=i+c_u}^{|F^k|} y_u^{f_l k} \leq (1 - y_u^{f_i k}) c_u \qquad k \in C,\ u \in N \setminus \{s_k\},$$
$$i = c_u + 1, \ldots, |F^k| - c_u. \tag{5.19}$$

Proof. For a given position i in the ordering of F^k, with $i = c_u + 1, \ldots, |F^k| - c_u$, if the function f_i is installed at node u for the commodity k, then the number of functions that must be installed before and after f_i must be less than or equal to the capacity c_u of node u, otherwise the node-capacity constraints are violated. ■

5.2 Problem reformulation using Benders cuts

The size of the compact MILP model presented in Section 5.1 easily becomes intractable, due to the fact that the model contains $O(|N||F||C| + |A||C|)$ variables and constraints. Hence, to deal with instances of realistic size, it is necessary to rely on a problem decomposition in which smaller and easier-to-solve components are solved separately, and then the gained information is combined, and the process is repeated in an iterative way until convergence.

In this chapter we focus on Benders decomposition approach, which can be efficiently exploited thanks to our result given in Section 5.1.3. Corollary 5.3 states that the integrality condition of the flow variables can be relaxed, and hence, using Benders decomposition, we can project t variables out from the model. That way, Benders decomposition allows for a reduction of the total number of variables from $O(|N||F||C| + |A||C|)$ to $O(|N||F||C|)$. In dense networks, this means an order of magnitude reduction of the number of variables. The telecommunication application that motivates our problem may rely on network graphs G that are rather dense, whereas the number of virtual network functions remains typically small. In this section we describe the basic steps for designing an efficient Benders decomposition approach.

The Benders master problem aims at finding an optimal placement of virtual network functions associated to each commodity and deciding in which order these functions should be visited, while "guessing" that a feasible routing path can be found for such a placement. Hence, in the master problem we decide on the value of (x, y, w) variables,

and once these values are fixed, we need to make sure that for each commodity $k \in C$ a routing path can be found such that the nodes are visited according to the order specified by x, and without violating the latency constraint. The master ILP model is initialized as follows:

$$\min \quad \sum_{k \in C} \sum_{u \in N} \sum_{f \in F^k} \psi_u^f \ y_u^{fk} + \sum_{u \in N} \psi_u \ w_u$$

$$\text{s.t. } (x, y, w) \text{ satisfy } (2.8)\text{-}(2.9),\ (2.11)\text{-}(2.12),\ (5.7)\text{-}(5.9)$$

Once a feasible solution $(\hat{x}, \hat{y}, \hat{z})$ of the relaxed master problem is obtained, its feasibility is checked by plugging in its values into the constraints (2.2), (2.3), (2.7) and (2.10). We first observe that this linear system of inequalities is separable, so that the feasibility check can be performed independently for each commodity $k \in C$. For a given binary vector $(\hat{x}, \hat{y})$, the feasibility subproblem for a commodity $k \in C$ can be formulated as the following linear program (we omit the superscript k for the sake of better readability):

$$\min \quad 0$$

$$\sum_{(u,v) \in A} t_{uv} - \sum_{(v,u) \in A} t_{vu} \quad = \quad \begin{cases} 1, & \text{if } u = s_k, \\ -1, & \text{if } u = d_k, \\ 0. & \text{otherwise} \end{cases} \quad \forall u \in N \quad (\alpha_u) \tag{5.20}$$

$$-t_{uv} \geq \hat{x}_v^f - \hat{x}_u^f - \hat{y}_v^f - 1 \quad \forall f \in F^k, \quad \forall (u,v) \in A \quad (\beta_{uv}^f) \tag{5.21}$$

$$\sum_{(v,u) \in A} t_{vu} \geq \hat{y}_u^f \quad \forall f \in F^k, \quad \forall u \in N \quad (\gamma_u^f) \tag{5.22}$$

$$-\sum_{(u,v) \in A} l_{uv} t_{uv} \geq -l_k \quad (\delta) \tag{5.23}$$

$$t_{uv} \geq 0 \quad \forall (u,v) \in A \tag{5.24}$$

Hence, the solution $(\hat{x}, \hat{y}, \hat{w})$ is feasible, iff for each $k \in C$, there are sufficient arc and node capacities to route the s_k-d_k flow and the latency of the routing path is at most l_k. According to LP duality, if the dual of the k-th Benders subproblem is unbounded, its primal is infeasible, and correspondingly, the point $(\hat{x}, \hat{y}, \hat{w})$ has to be cut off.

After associating dual variables α, β, γ and δ to constraints (5.20)-(5.23), respectively, the LP dual of the k-th Benders subproblem reads as follows:

$$\max \quad \alpha_{s_k} - \alpha_{d_k} - \sum_{f \in F^k} \sum_{(u,v) \in A} \beta_{uv}^f (1 - \hat{x}_v^f + \hat{x}_u^f + \hat{y}_v^f) + \sum_{u \in N} \sum_{f \in F^k} \gamma_u^f \hat{y}_u^f - \delta l_k$$

$$\alpha_u - \alpha_v - \sum_{f \in F^k} \beta_{uv}^f + \sum_{f \in F^k} \gamma_v^f - \delta l_{uv} \leq 0 \qquad (u,v) \in A \tag{5.25}$$

$$\alpha \text{ free}, (\beta, \gamma, \delta) \geq 0 \tag{5.26}$$

Hence, any (extreme) ray $(\tilde{\alpha}, \tilde{\beta}, \tilde{\gamma}, \tilde{\delta})$ of the above dual induces a valid Benders feasibility cut:

$$\tilde{\alpha}_{s_k} - \tilde{\alpha}_{d_k} - \sum_{f \in F^k} \sum_{(u,v) \in A} \tilde{\beta}_{uv}^f (1 - x_v^f + x_u^f + y_v^f) + \sum_{u \in N} \sum_{f \in F^k} \tilde{\gamma}_u^f y_u^f - \tilde{\delta} l_k \quad \leq \quad 0 \tag{5.27}$$

These Benders cuts are actively separated at every node of the branch-and-bound tree, and added to the relaxed master problem. The resulting strategy is commonly denoted as *Branch-and-Benders-Cut* (see, e.g., [38]). Effective separation of Benders cuts involves advanced stabilization techniques and proper selection of extreme rays. To this end, we have decided to use the implementation of Benders feasibility cuts, in an *annotated Benders* setting of CPLEX [13], which utilizes the major stabilization and implementation techniques from [51, 53, 54].

Remark: This Benders reformulation can be applied on the node-capacitated and the conflict constrained version of the problem. This can be done by adding constraints (2.4) and (2.6), respectively, to the Benders master problem. Furthermore, this Benders reformulation can be applied on the compact MILP formulation presented in chapter 2 i.e., VNF-capacitated version of the problem. As the t variables do not appear in the VNF capacity constraints.

5.3 MILP-based Heuristic

When models with an exponential number of constraints are implemented within a Branch-and-Cut scheme (as this is the case with our Branch-and-Benders-Cut approach), general purpose solvers in general struggle with finding feasible solutions. This is because they are missing an important information concerning the structure of feasible solutions, as majority of constraints are left out from the model and are separated on the fly during the branching process. Hence, only after enumerating many branching nodes, much deeper in the branching tree, a larger number of cuts becomes available to the solver so that more useful feasible solutions can be found. It is therefore crucial for the performance of Branch-and-Benders-Cut algorithms that

high-quality solutions are provided in the initialization phase. To this end, we use the MILP-based heuristic introduced in Chapter 3, whose solution is handed over to the solver immediately before solving the first LP of the relaxed master problem.

Path-based formulation

The path formulation has been introduced in Chapters 3 and 4. In the Uncapacitated version of the problem the path formulation is a bit different from the one already introduced in previous chapters. The objective function is not the same, and the constraints (5.7) should be added to the model. Also, the VNF-capacity, the node-capacity and the conflict constraints are relaxed. Then, the Uncapacitated VNFPR problem can be modeled as the following MILP:

$$\min \quad \sum_{k \in C} \sum_{u \in N} \sum_{f \in F^k} \psi_u^f \ y_u^{fk} + \sum_{u \in N} \psi_u \ w_u$$

$$(x, y, w) \text{ satisfy } (4.2), (4.6)\text{-}(4.11), (5.7)\text{-}(5.9)$$

$$\lambda_p^k \in \{0, 1\} \qquad\qquad k \in C, p \in \mathcal{P}_k$$

Remark: The MILP-based heuristic is adapted for the node-capacitated and conflict constrained version of the problem by adding inequalities (2.4) and (2.6), respectively.

Heuristic

We generate up to κ elementary shortest $s_k - d_k$-paths for each commodity $k \in C$, using Yen's algorithm [138]. The paths whose length is below l_k are added to the model in the order from the shortest one to the longest one.

Obviously, if Yen's algorithm would generate all possible s_k-d_k-paths with latency less or equal to l_k and there would be less than κ of them, our resulting compact MILP model would be an exact reformulation of the problem. This however does not happen very often in practice (we usually keep $\kappa \leq 50$). Nevertheless, the quality of solutions obtained by this simple MILP heuristic is quite satisfying, as reported in the following section.

5.4 Computational results

The purpose of this section is to test the scalability and the efficiency of the proposed Branch-and-Benders-Cut computational framework and to compare it to other exact methods, using an off-the-shelf solver. To this end, we use the commercial solver CPLEX and compare our approach to two available options: first, solving the problem as a compact formulation (P) introduced in Section 5.1, and second, applying Automatic Benders decomposition [26] to the formulation (P').

Our tests are conducted so as to evaluate: the performance of Benders decomposition in terms of computing time; the size limit of realistic instances to be solved to optimality in a reasonable time limit; the quality of the MILP-based heuristic, and, finally, the possible benefit of valid inequalities proposed in Section 5.1.4 in improving the computing times or reducing the final gaps. We also investigate the performance of our models for problem instances with node-capacity and conflict constraints.

All the tests reported in this section were made using a machine with Intel(R) Xeon(R) CPU E5-2650 v2 processor clocked at 2.60GHz and 252GB RAM, under Linux operating system. All methods are implemented using the Python API for CPLEX, which is run in single-threaded mode and all CPLEX parameters were set to their default values. A default time limit of three hours is set for each instance from the benchmark set described in Section 5.4.1.1, and of one hour for each instance from the benchmark set described in Section 5.2. The default memory limit was set to 20GB.

The following settings were tested in our computational experiments:

- `C`: the compact MILP formulation (P) proposed in Section 5.1;
- `RC`: the relaxed compact MILP formulation (P') in which the flow variables t are relaxed according to Corollary 5.3;
- `AB`: automatic Benders decomposition available in Cplex [26], applied to the model (P');
- `B`: Branch-and-Benders-Cut obtained after decomposing the problem into $|C|$ subproblems and adding Benders feasibility cuts (as explained in Section 5.2);
- `B+VI`: setting `B` in which Valid Inequalities from Section 5.1.4 are additionally used to initialize the model;

- `B+PH`: setting `B` in which initial solutions are obtained using the MILP-Heuristic described in Section 5.3. The number of generated paths by Yen's algorithm is limited to $\kappa = 50$ paths per commodity and the time limit for running the MILP-heuristic is set to 900 seconds;
- `B+VI+PH`: setting `B+PH` in which Valid Inequalities from Section 5.1.4 are additionally used to initialize the model.

5.4.1 Benchmark instances

In order to test the performance of the proposed methods, we generate two set of instances. The first set is generated using Erdős-Rényi graphs, and the second set is obtained from the well-known library of telecommunication network instances, called SNDlib [125].

5.4.1.1 Instances derived from Erdős-Rényi graphs

These instances are generated as Erdős-Rényi graphs using the Python library NetworkX. We vary the number of nodes $|N| \in \{25, 50, 100\}$ and consider the graph density $d \in \{0.5, 0.9\}$. For each arc $(i, j) \in A$, the latency $l_{ij} \in \mathbb{N}$ is chosen uniformly at random from the interval $[1, 25]$, and we set $l_{ij} = l_{ji}$. For each node $u \in N$, its node activation cost $\psi_u \in \mathbb{N}$ is chosen uniformly at random from $[50, 1000]$. The total number of VNFs $|F|$ is set to 10, and for each $u \in N, f \in F$, the installation cost $\psi_u^f \in \mathbb{N}$ is chosen uniformly at random from $[10, 100]$. We vary the total number of commodities, by considering $|C| \in \{10, 15, \ldots, 50\}$. For each commodity, its distinct source and destination nodes are chosen randomly from N. To define the set F^k of VNFs associated to each commodity, we randomly choose between $\{6, 8, 10\}$ functions from the set F and order them randomly. The latency l_k is set to 1.5 times the length of the shortest path (with respect to the values l_{ij}). .

5.4.1.2 Instances derived from the SNDlib

This benchmark set is described on Chapter 3. In this Chapter we test the Benders reformulation on the following instances-type: {"Abilene", "Atlanta", "Dfn-bwin", "Dfn-gwin", "Di-yuan", "Newyork", "Nobel-germany", "Nobel-us", "Pdh", "Polska"}. For each instance-type we have generated ten different instances.

5.4.2 Obtained results

Major results are summarized in this section, whereas more detailed results, given per each instance, can be found in Section 5.4.3. For each of the different settings listed above, we report the overall CPU time in seconds and the percentage gap after reaching the time or memory limit (in case the optimal solution has not been found). To final gap is calculated as GAP $= ((UB - LB)/LB) * 100\%$, where UB denotes the best feasible solution, and LB the global lower bound found in each run.

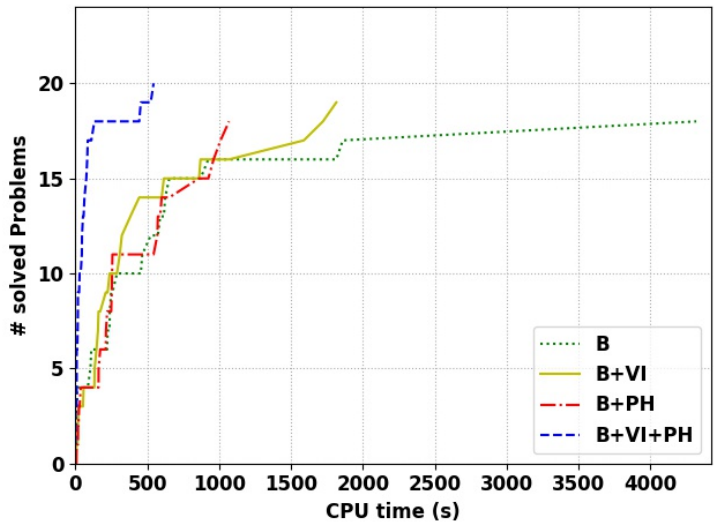
We start by comparing the four Branch-and-Benders-cut configurations (`B`, `B+VI`, `B+PH`, `B+VI+PH`) on the set of Erdős-Rényi graphs. After determining the best configuration, we compare it with three other alternative MILP approaches (namely, `C`, `AB`, and `RC`). In Section 5.4.2.2 we then study the instances derived from SNDlib and compare the best Branch-and-Benders-cut configuration against the available alternatives, and we also study the sensitivity of their performance with respect to the introduction of node-capacity and conflict constraints.

5.4.2.1 Results obtained on Erdős-Rényi graphs

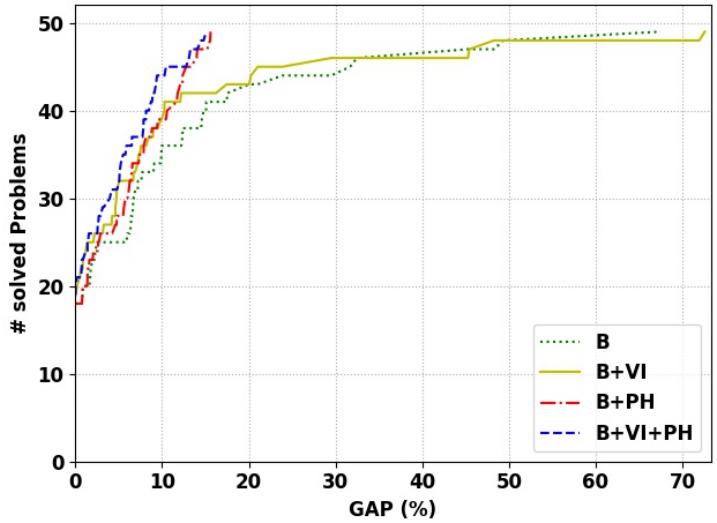
Graphical summary of the obtained results is given in Figures 5.2-5.7, where we provide cumulative charts representing the CPU times and the GAPs for instances with $|N| \in \{25, 50, 100\}$. Tables 5.1-5.6 in Section 5.4.3 provide detailed results obtained for this set of instances.

Comparing four Branch-and-Benders-cut configurations Figures 5.2(a) and 5.3(a) allow to compare the CPU times of the four Branch-and-Benders-Cut settings for the instances with $|N| \in \{25, 50\}$ and $d \in \{0.5, 0.9\}$. A point with coordinates (x, y) in this chart indicates that for y instances, the CPU time needed to prove optimality was below x seconds. A similar representation of the final gaps for the same group of instances is shown in Figures 5.2(b) and 5.3(b). Figures 5.2 and 5.3 show that the `B+VI+PH` setting is outperforming the remaining three Benders settings (`B`, `B+VI`, `B+PH`). For example, the number of instances with $|N| \in \{25, 50\}$, $d = 0.5$ that can be solved to optimality in less than ten minutes is equal to 20 for `B+VI+PH`, whereas it is only 14 (12, 11) for `B+VI` (`B+PH`, `B`, respectively). All 50 instances are solved with a GAP smaller than or equal to 15% by `B+VI+PH` and `B+PH`, whereas, this is true for only 41 and 42 instances when settings `B`, respectively `B+VI`, are employed. A similar behavior can be observed for instances with $|N| \in \{25, 50\}$ and $d = 0.9$. These results clearly exhibit the benefits of adding valid inequalities and path-based heuristic to the basic Benders

decomposition setting. Consequently, in the following we will consider `B+VI+PH` as our default Benders setting and compare it with alternative MILP approaches.

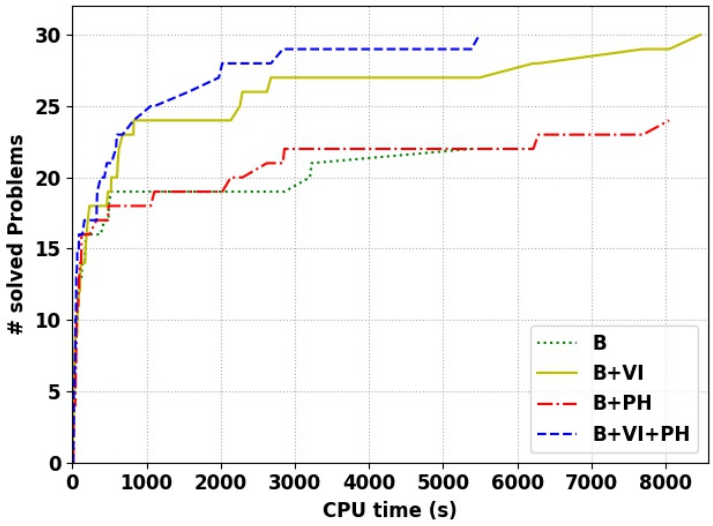


(a) CPU time (s)

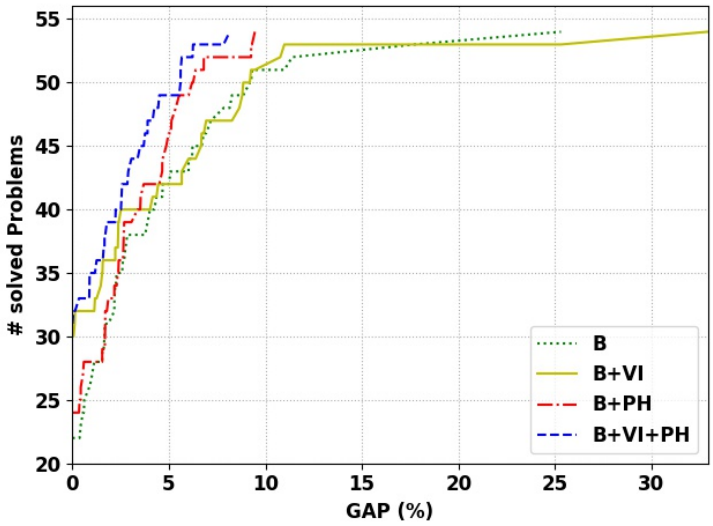


(b) GAP (%)

Figure 5.2: CPU time and GAP comparison between Branch-and-Benders-Cut algorithms with and without valid inequalities and with and without MILP-heuristic ($|N| \in \{25, 50\}$ and $d = 0.5$).



(a) CPU time (s)

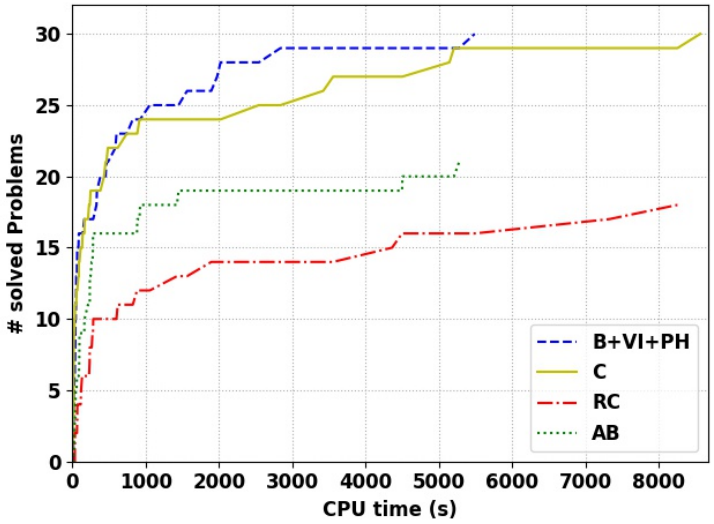


(b) GAP (%)

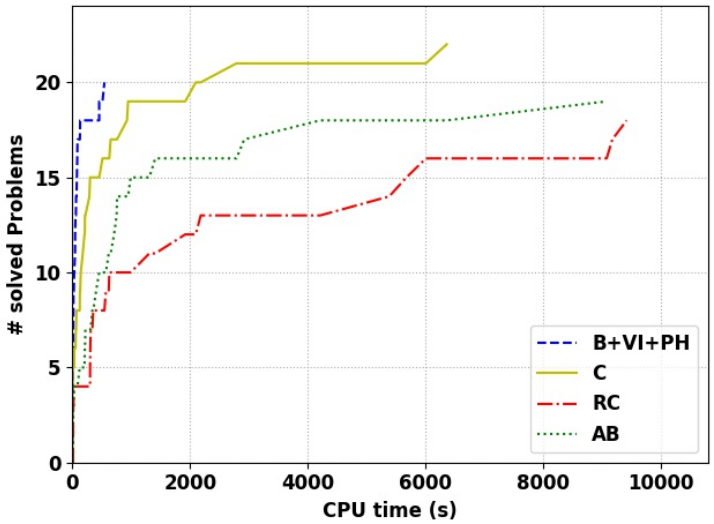
Figure 5.3: CPU time and GAP comparison between Branch-and-Benders-Cut algorithms with and without valid inequalities and with and without MILP-heuristic ($|N| \in \{25, 50\}$ and $d = 0.9$).

Comparison with alternative MILP approaches We now compare our Branch-and-Benders-Cut (`B+VI+PH`) against three alternative MILP methods available by using Cplex as an off-the-shelf MILP solver: `C`, `RC` and `AB`. Cumulative charts for graphs with $|N| \in \{25, 50\}$ and for $d = 0.9$ and $d = 0.5$ are given in Figures 5.4 (CPU times) and 5.5 (GAPs). We notice that all models have more difficulties in finding optimal solutions

for sparser graphs. Nevertheless, the setting `B+VI+PH` remains the best performing one, followed by the compact formulation `C`. Figure 5.5 indicates that for sparser graphs, for 90% (respectively 100%) of the instances the final gaps obtained by the Branch-and-Benders-Cut remain below 10% (respectively 15%), whereas the corresponding figures for the final gaps achieved by the compact model are 20% (respectively 40%).

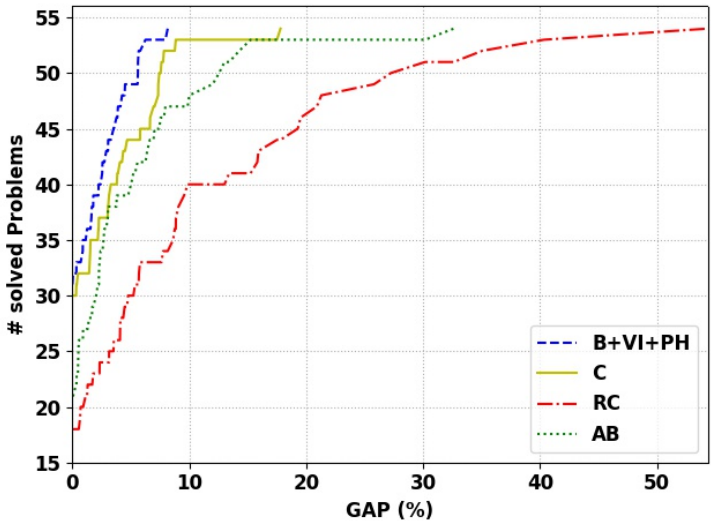


(a) $d = 0.9$

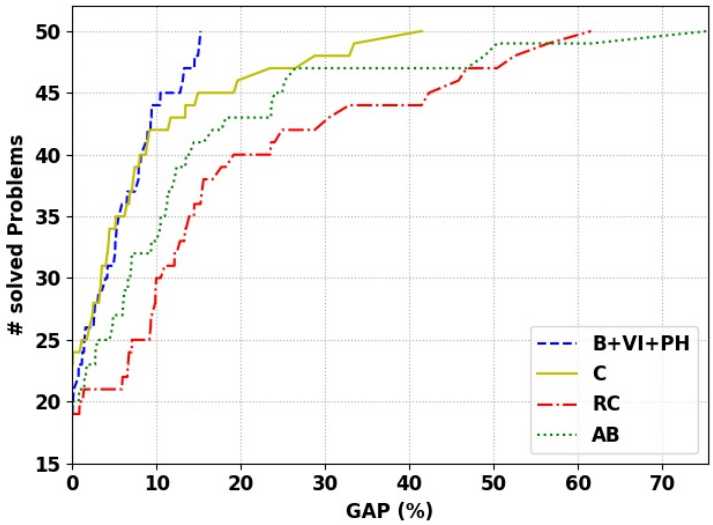


(b) $d = 0.5$

Figure 5.4: CPU time comparison between Branch-and-Benders-Cut algorithm, compact and relaxed compact formulation and Automatic Benders for graphs with 25 and 50 nodes and density 0.9 and 0.5.



(a) $d = 0.9$



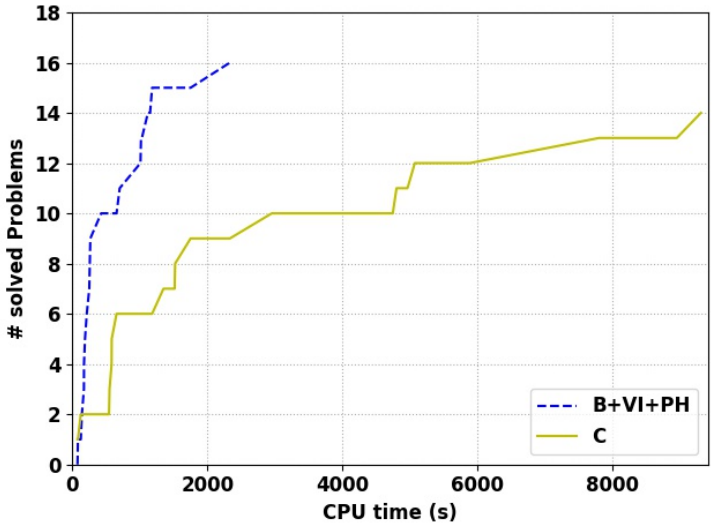
(b) $d = 0.5$

Figure 5.5: GAP comparison between Branch-and-Benders-Cut algorithm, compact and relaxed compact formulation and Automatic Benders for graphs with 25 and 50 nodes and density 0.9 and 0.5.

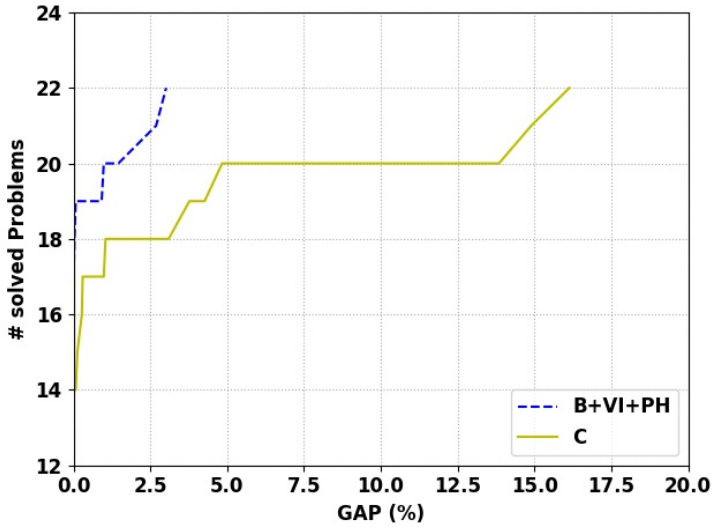
We observe that 20 instances were solved with Branch-and-Benders-Cut in less than 500 seconds whereas the same number of instances was solved by the compact formulation in less than 2000s. On the contrary, the relaxed compact formulation `RC` (resp. the Automatic Benders) can solve only 19 (resp. 18) instances within the given time limit of three hours. In terms of the final gap, Branch-and-Benders-Cut algorithm solves all

instances within the final gap which is below 15%, whereas the same instances were solved with a gap as high as 40% for the compact formulation, 60% for the relaxed compact model, and 75% for the automatic Benders.

Overall, we observe that `RC` and `AB` methods provide solutions of poor quality compared to the Branch-and-Benders-Cut and the Compact model. Therefore, for larger instances with 100 nodes, we compare only the `B+VI+PH` and `C` settings. Figures 5.6 and 5.7 summarize the results obtained by solving instances with 100 nodes and density $d = 0.9$ and $d = 0.5$, respectively. We observe that in the few cases when the number of commodities is rather small, the compact MILP formulation is outperforming (in terms of the CPU time) `B+VI+PH`. However, by increasing the number of commodities, the compact MILP model struggles with the size of the underlying LP-formulation. On the contrary, the Branch-and-Benders-Cut algorithm provides relatively small final gaps. For example, for $d = 0.9$ final gaps are below 3% for `B+VI+PH`, whereas for the compact MILP formulation the final gaps can be as large as 16% (cf. Figure 5.6(b)). Also, from Figure 5.6(a) we notice that 16 out of 22 instances with $d = 0.9$ are solved within less than 2500 seconds by `B+VI+PH`, while this is true for only 9 instances when solved using the Compact formulation.



(a) CPU time (s)

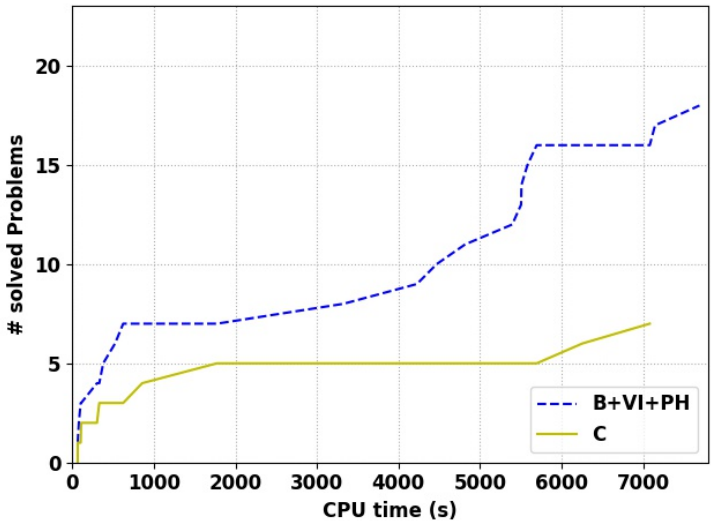


(b) GAP (%)

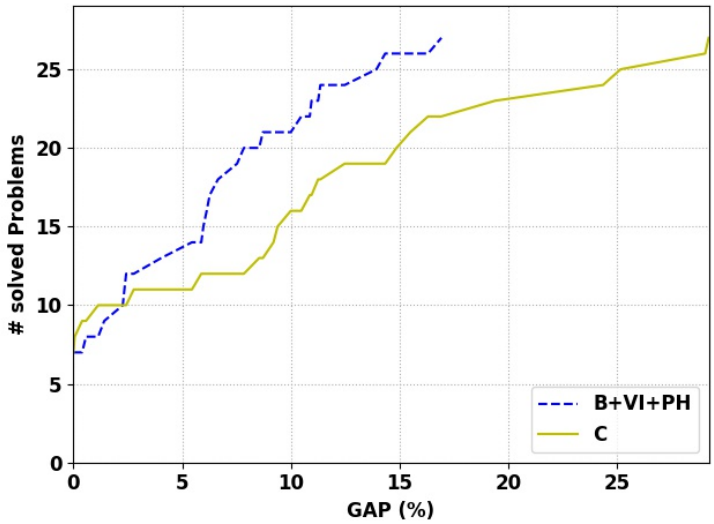
Figure 5.6: CPU time and GAP comparison between Branch-and-Benders-Cut algorithm with valid inequalities and MILP-heuristic and compact formulation ($|N| = 100$ and $d = 0.9$).

Similarly, Figure 5.7 shows that the number of instances with $|N| = 100$ and $d = 0.5$ solved by the Branch-and-Benders-Cut algorithm without exceeding the time limit is 18 (7 are solved to optimality and for 11, the memory limit was exceeded). From Figure 5.7(b) we observe that the largest GAP for `B+VI+PH` was 17%, whereas the same (or smaller) GAP is achieved for only 23 out of 27 instances using the Compact

formulation.



(a) CPU time (s)

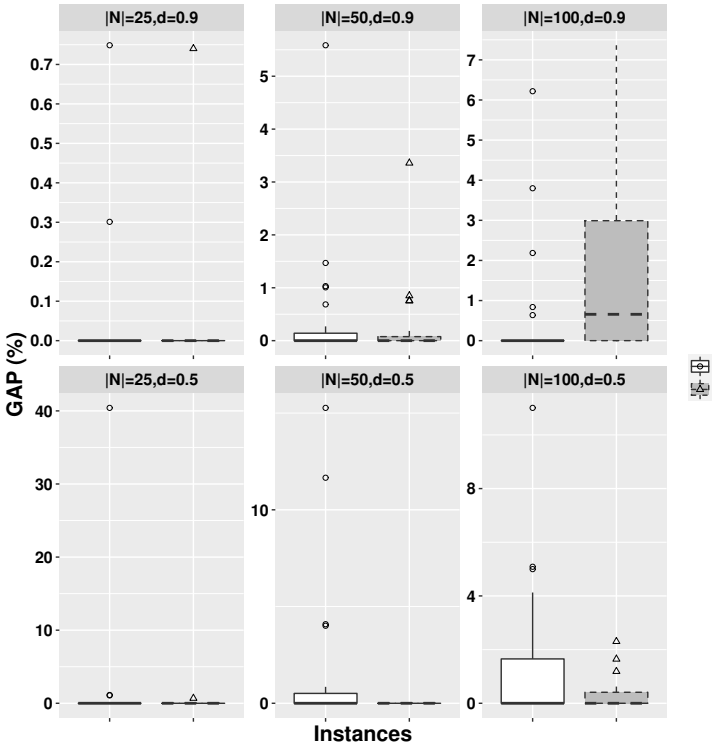


(b) GAP (%)

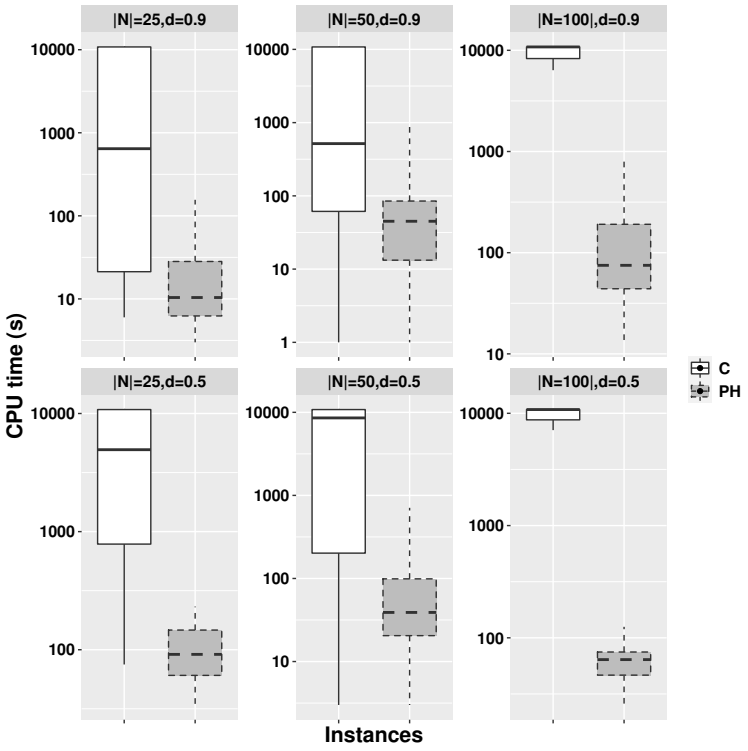
Figure 5.7: CPU time and GAP comparison between Branch-and-Benders-Cut algorithm with valid inequalities and MILP-heuristic and the compact formulation ($|N| = 100$ and $d = 0.5$).

Quality of solutions found by the MILP-based heuristic In the following we analyze the quality of solutions found by the MILP-based heuristic (`PH`) and its overall efficiency. Figure 5.8 provides boxplots in which we are comparing the quality of

the solutions and the CPU times of the MILP-based heuristic (PH) and the Compact formulation (C). The latter one is considered as an off-the-shelf alternative for finding feasible solutions. In the provided boxplots, all instances derived from Erdős-Rényi graphs are taken into account. The relative gaps are calculated with respect to the optimal solution or, alternatively, the best-known upper bound found by any of the four exact methods (contrary to Tables 5.2-5.6 from Section 5.4.3 where CPLEX exit gaps are reported). We observe that in most of the cases the heuristic provides feasible solutions of almost the same (or even better) quality compared to those found by the Compact formulation, but within a significantly less CPU time. By comparing the solution gaps, we notice that the heuristic performs particularly well on sparser graphs. This can be explained by the fact that, by decreasing the graph density, the number of feasible routing paths decreases too, hence the path-based MILP-formulation (which is used within the heuristic) often captures the optimal routing paths. By enlarging the number of nodes or increasing the density, this benefit of the MILP-heuristic might be lost, as it can be observed in Figure 5.8 (a), where e.g., for instances with 100 nodes and $d = 0.9$ the solutions found by the Compact formulation are of better quality than those found by the MILP-heuristic.



(a) GAP (%)

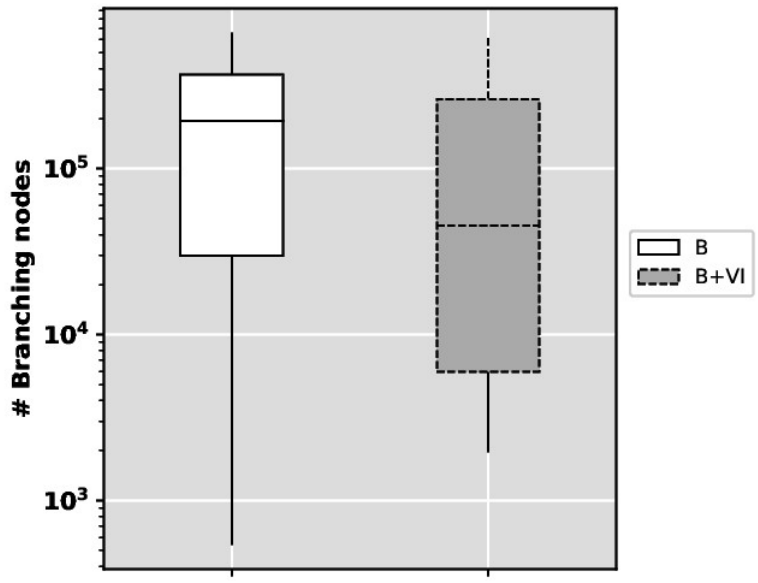


(b) CPU time (s)

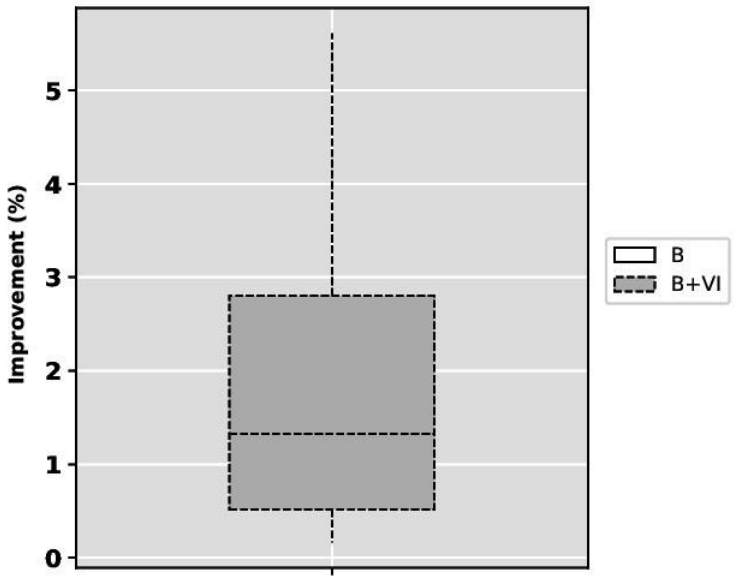
Figure 5.8: GAP and CPU time comparison between compact MILP formulation and MILP-based Heuristic.

Effect of adding valid inequalities Finally, we test the potential benefits of valid inequalities proposed in Section 5.1.4. Figure 5.9(a) compares the size of the branching tree for the basic setting `B` and the setting `B+VI` in which valid inequalities are added to the model. We notice that adding the valid inequalities definitely helps our Benders approach to reduce the number of branching nodes. These reductions can be significant (a few orders of magnitude). For 50% of the solved instances the number of branching nodes was reduced from $\approx 10^5$ to $\approx 10^4$. In Figure 5.9(b) we also show the relative improvement of lower bounds, calculated as (LB_{B+VI} - LB_B) / LB_B * 100.

We observe that the lower bound at the root node could be improved by 1% to 3% for majority of considered instances.



(a) The number of branching nodes for settings `B` and `B+VI` .



(b) Relative improvement of lower bounds at the root node after adding valid inequalities.

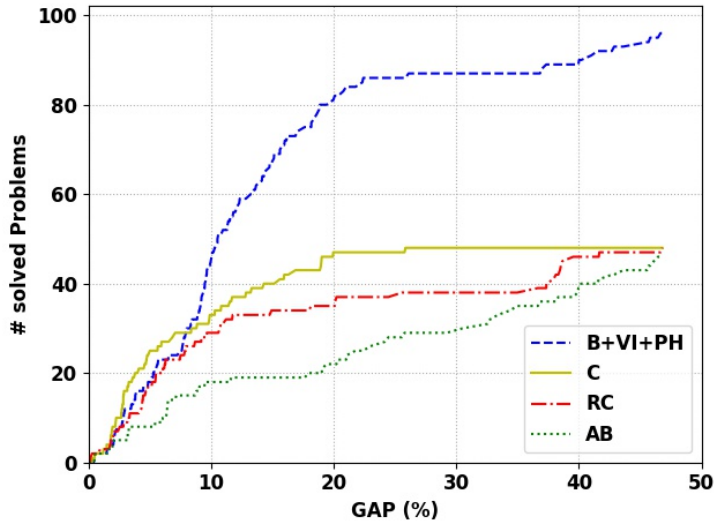
Figure 5.9: Number of branching nodes and bounds improvement of the Branch-and-Benders-Cut approach with and without valid inequalities for graphs with $|N| = 100$ and $d = 0.9$.

5.4.2.2 Results obtained on the set of SNDlib instances

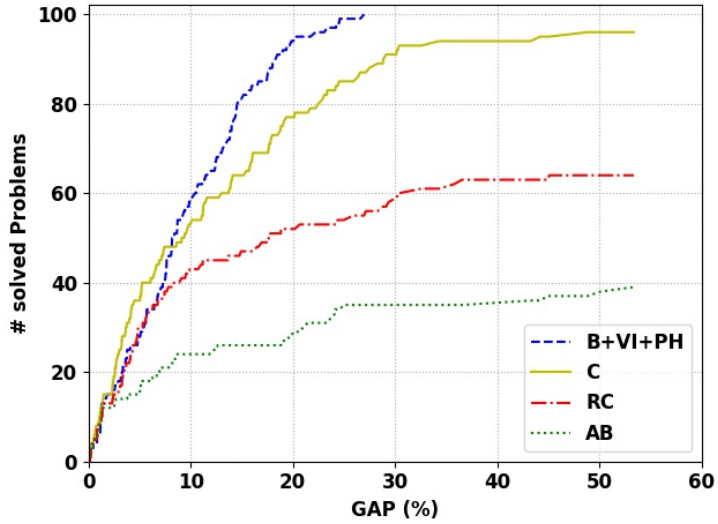
Figure 5.10 compares GAPs of the Branch-and-Benders-Cut approach (`B+VI+PH`), with those obtained using the Compact formulation, the Relaxed Compact formulation and the Automatic Benders of Cplex. Instances with and without node-capacity and conflict constraints are considered. Axis y represents the number of solved instances, and axis x represents the final GAP (given in percentage). The gap is set to 100% if no feasible solution is found.

Figure 5.10(a) shows results for SNDlib instances with node-capacity and conflict constraints. We notice that for 96 out of 100 instances, `B+VI+PH` finds a feasible solution, and the worst obtained gap is below 47%. On the contrary, less than half of the instances could be solved within the same gap for the remaining three approaches (`AB`, `C` and `RC`).

When it comes to SNDlib instances without node-capacity and conflict constraints, Figure 5.10(b) shows that for all instances `B+VI+PH` finds a feasible solution and the worst obtained gap is below 27%. On the contrary, only 86%, 55% and 36% of these instances can be solved within the same gap using the Compact formulation `C`, the Relaxed Compact formulation `RC` and the Automatic Benders `AB`, respectively.



(a) With node-capacity and conflict constraints.



(b) Without node-capacity and conflict constraints.

Figure 5.10: GAP comparison between Branch-and-Benders-Cut algorithm (`B+VI+PH`), Compact (`C`) and Relaxed Compact formulation (`RC`) and Automatic Benders (`AB`) for SNDlib instances with and without node-capacities and conflict constraints.

A more detailed comparison is provided in Figure 5.11 which shows the average final gaps per instance-type for the instances with node-capacity and conflict constraints. Recall that each instance type contains ten associated instances. We observe that no feasible solution is found by the Compact formulation, the Relaxed Compact for-

mulation and the Automatic Benders for instance-types "Abilene", "Atlanta", "Nobel-germany" and "Nobel-us" within the given time limit. On the other hand, the associated final gaps provided by the Branch-and-Benders-cut algorithm are 10.12%, 25.70%, 50.43% and 15.31%, respectively. Similarly, no feasible solutions are obtained by the Compact formulation for instance-type "Newyork", and no solutions are provided by the Relaxed Compact formulation and the Automatic Benders for instance-type "Polska". For these two instance-types, the Branch-and-Benders-cut algorithm provides average final gaps of 30.85% (for "Newyork") and 12.05% (for "Polska"). For instance-types "Dfn-bwin" and "Dfn-gwin" the Compact and the Relaxed Compact formulations slightly outperform the other proposed methods, which can be explained by the small size of these instances (10, resp. 11 nodes and 45, resp. 47 edges). Finally, very small gaps are obtained by all methods but `AB` for instance-type "Pdh", which is comprised by 11 nodes and 34 edges.

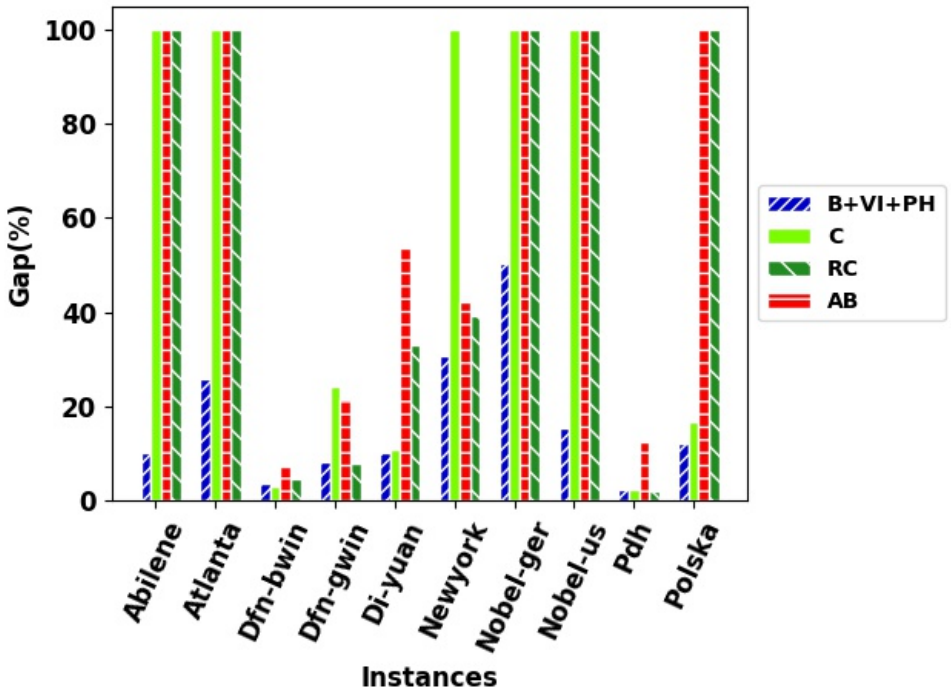


Figure 5.11: Average GAPs per instance-type for SNDlib instances with node-capacities and conflict constraints.

Figure 5.12 shows a similar comparison for the SNDlib instances without node-capacity and conflict constraints. We observe that the Branch-and-Benders-cut algorithm (`B+VI+PH`) provides small final gaps compared to those obtained by the Compact formulation for "Atlanta", "Newyork", "Nobel-germany", "Nobel-us" and "Polska". For

"Abilene" and "Dfn-gwin" the gaps obtained by the Compact formulation are slightly better. Finally, for instance-types "Dfn-bwin", "Di-yuan" and "Pdh", the final gaps provided by both methods are almost the same. On the contrary, `AB` could not find any feasible solution for "Abilene", "Atlanta", "Nobel-germany", "Nobel-us" and "Polska". Similarly, Relaxed compact formulation did not find any feasible solution for "Atlanta" and "Nobel-germany".

From Figures 5.11 and 5.12 we also observe an interesting fact that in very few cases the final gaps of the Compact formulation can be improved by relaxing the integrality condition on the arc variables (cf. "Dfn-gwin", "Newyork" and "Pdh").

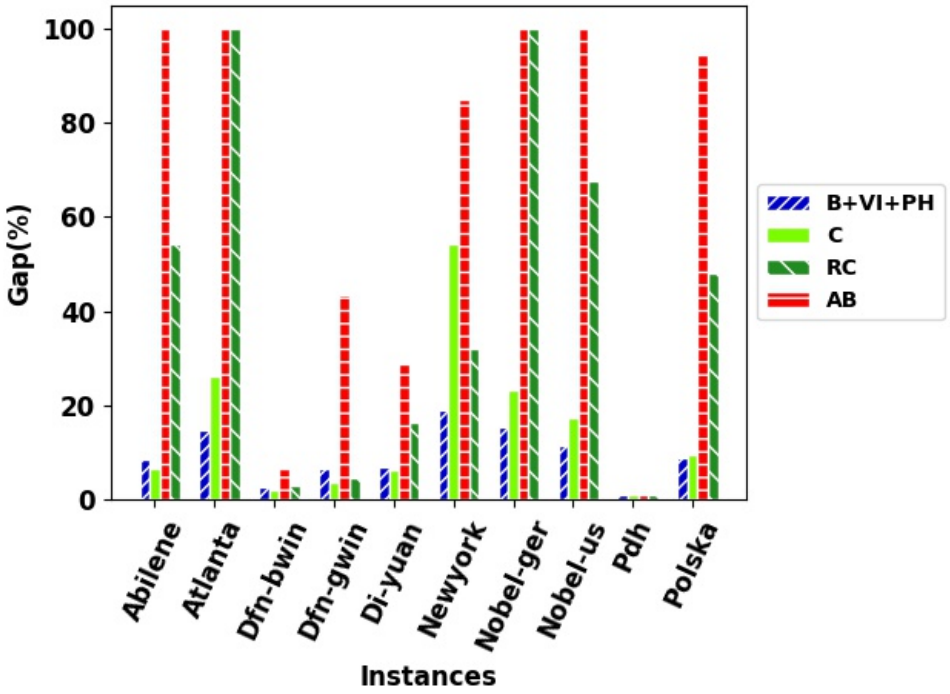


Figure 5.12: Average GAPs per instance-type for SNDlib instances without node-capacities and conflict constraints.

5.4.3 Detailed results

For each of the different settings listed in Section 5.4, we report the overall CPU time in seconds (or TL/"–" if the time/memory limit is reached, respectively), and the final gap in percent after reaching the time or memory limit (in case the optimal solution has not been found). To final gap is reported as Gap $= (UB - LB)/LB * 100\%$, where UB denotes the best feasible solution, and LB the global lower bound found in each run.

In addition, for our MILP-heuristic, in the columns labeled by `PH`, we report the CPU time of the heuristic (TL means that the time limit of 900 seconds was reached without solving the model to optimality), and the gap with respect to the optimal solution (or, the best known upper bound, in case the latter is not available).

Tables 5.1, 5.2 and 5.3 summarize the results obtained for the instances derived from Erdős-Rényi graphs with $d = 0.5$ and with 25, 50 and 100 nodes, respectively. Tables 5.4, 5.5 and 5.6 summarize the results obtained for the instances derived from Erdős-Rényi graphs with $d = 0.9$ and with 25, 50 and 100 nodes, respectively. In the last two columns of Tables 5.2-5.6 we report the CPU time per instance required by the MILP-based heuristic (`PH`) and the relative gap of the obtained solution with respect to the optimal solution (or, alternatively, the best-known upper bound found by any of the four exact methods).

Tables 5.7-5.9 (resp. Tables 5.10-5.12) show detailed results of 100 instances derived from SNDlib with (resp. without) node-capacity and conflict constraints. The CPU time of all solved instances exceeds the time limit (this explains the removal of "t[s]" column in tables). Sign – in "Gap" column in result tables indicates that no feasible solution is found.

Instances				Branch-and-Benders-Cut								C		RC		AB		PH	
				B		B+VI		B+PH		B+VI+PH									
$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap
25	290	10	6	11	0.0	15	0.0	18	0.0	18	0.0	**3**	0.0	22	0.0	24	0.0	9	0.0
25	296	10	8	7	0.0	57	0.0	25	0.0	8	0.0	**1**	0.0	15	0.0	7	0.0	1	0.0
25	326	10	10	13	0.0	21	0.0	37	0.0	89	0.0	**2**	0.0	12	0.0	11	0.0	1	0.0
25	280	15	6	6	0.0	6	0.0	8	0.0	6	0.0	**2**	0.0	13	0.0	5	0.0	2	0.0
25	310	15	8	*TL*	0.9	1722	0.0	*TL*	0.9	7443	0.0	**69**	0.0	9180	0.0	9083	0.0	17	0.0
25	302	15	10	624	0.0	211	0.0	600	0.0	189	0.0	**30**	0.0	311	0.0	708	0.0	63	0.0
25	286	20	6	110	0.0	157	0.0	161	0.0	152	0.0	**59**	0.0	300	0.0	124	0.0	53	0.0
25	292	20	8	239	0.0	309	0.0	255	0.0	247	0.0	**213**	0.0	6010	0.0	401	0.0	46	0.0
25	302	20	10	465	0.0	444	0.0	860	0.0	788	0.0	**7**	0.0	354	0.0	752	0.0	15	0.0
25	308	25	6	589	0.0	615	0.0	1010	0.0	728	0.0	**288**	0.0	1307	0.0	989	0.0	86	0.0
25	316	25	8	653	0.0	382	0.0	573	0.0	628	0.0	**141**	0.0	569	0.0	759	0.0	76	0.0
25	302	25	10	249	0.0	146	0.0	257	0.0	**121**	0.0	927	0.0	2181	0.0	338	0.0	17	0.0
25	312	30	6	925	0.0	870	0.0	1069	0.0	805	0.0	**126**	0.0	9423	0.0	1408	0.0	44	0.0
25	302	30	8	224	0.0	**133**	0.0	212	0.0	238	0.0	205	0.0	627	0.0	455	0.0	9	0.0
25	296	30	10	–	6.6	–	7.3	–	7.4	–	7.8	*TL*	**2.3**	*TL*	12.1	*TL*	4.7	128	8.5
25	280	35	6	*TL*	1.8	*TL*	0.6	*TL*	1.4	*TL*	0.7	*TL*	**0.0**	*TL*	9.8	*TL*	1.7	14	0.7
25	308	35	8	4319	0.0	1589	0.0	961	0.0	**685**	0.0	2789	0.0	5679	0.0	4213	0.0	82	0.7
25	286	35	10	*TL*	72.0	*TL*	73.3	*TL*	15.5	*TL*	13.1	*TL*	**3.4**	*TL*	6.7	*TL*	4.8	29	15.1
25	284	40	6	*TL*	1.7	*TL*	**0.5**	*TL*	1.6	*TL*	0.7	*TL*	33.4	*TL*	–	*TL*	100.0	*TL*	192.0
25	294	40	8	*TL*	9.1	–	9.8	*TL*	7.9	–	8.2	–	**5.1**	*TL*	9.9	–	9.4	93	8.9
25	292	40	10	524	0.0	238	0.0	221	0.0	**218**	0.0	947	0.0	5386	0.0	220	0.0	8	0.0
25	312	45	6	*TL*	2.5	*TL*	2.2	–	2.6	*TL*	1.6	*TL*	**1.1**	*TL*	5.9	*TL*	2.7	13	1.6
25	308	45	8	*TL*	6.9	*TL*	7.6	*TL*	6.5	*TL*	**5.8**	*TL*	6.8	*TL*	15.4	*TL*	7.1	49	6.2
25	328	45	10	*TL*	12.4	*TL*	10.3	*TL*	8.1	–	8.9	*TL*	**4.3**	*TL*	13.9	*TL*	12.1	155	9.8
25	298	50	6	*TL*	6.4	*TL*	4.7	*TL*	2.9	–	**2.7**	*TL*	3.5	*TL*	15.6	*TL*	6.2	219	2.8
25	270	50	8	*TL*	9.9	*TL*	8.5	–	6.3	*TL*	**5.0**	*TL*	8.0	*TL*	13.4	*TL*	10.4	251	5.4
25	328	50	10	*TL*	7.2	*TL*	4.7	*TL*	6.6	*TL*	4.2	*TL*	**4.1**	*TL*	9.3	*TL*	11.3	309	4.4

Table 5.1: Results for graphs with $|N| = 25$ and $d = 0.5$.

Instances				Branch-and-Benders-Cut								C		RC		AB		PH	
				B		B+VI		B+PH		B+VI+PH									
$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap
50	1222	10	6	**104**	0.0	323	0.0	252	0.0	455	0.0	176	0.0	304	0.0	211	0.0	13	0.0
50	1244	10	8	TL	12.4	TL	5.1	TL	12.4	TL	5.1	**299**	0.0	TL	10.0	TL	12.4	81	5.3
50	1266	10	10	TL	17.5	TL	3.3	TL	14.1	TL	3.1	TL	**0.0**	TL	14.5	TL	14.4	131	3.2
50	1260	15	6	TL	14.8	TL	7.2	TL	9.5	TL	6.5	TL	**4.4**	TL	17.7	TL	16.6	44	7.0
50	1192	15	8	289	0.0	161	0.0	173	0.0	**131**	0.0	511	0.0	1918	0.0	617	0.0	62	0.0
50	1234	15	10	TL	2.3	TL	1.2	TL	2.1	TL	**1.1**	TL	2.0	TL	6.6	TL	3.0	35	1.2
50	1276	20	6	TL	0.8	TL	0.2	TL	0.8	TL	0.2	**649**	0.0	TL	0.9	TL	0.8	20	0.2
50	1220	20	8	TL	15.1	TL	12.2	TL	11.8	TL	**9.4**	TL	11.7	TL	24.9	TL	13.5	67	10.4
50	1288	20	10	1860	0.0	1815	0.0	570	0.0	**543**	0.0	6363	0.0	TL	7.1	2921	0.0	80	0.0
50	1252	25	6	TL	20.0	–	20.2	TL	12.1	TL	**10.4**	TL	13.4	TL	19.1	TL	25.2	75	11.7
50	1214	25	8	TL	6.4	TL	0.9	–	5.6	TL	**0.0**	TL	3.3	TL	9.3	TL	6.0	72	0.0
50	1170	30	6	TL	6.0	TL	4.6	TL	5.8	TL	5.2	TL	**2.4**	TL	32.9	–	6.6	41	5.5
50	1204	30	8	TL	2.0	TL	1.4	TL	1.4	TL	1.4	**2096**	0.0	TL	1.4	TL	1.4	55	1.4
50	1234	30	10	TL	14.6	TL	10.3	–	10.6	–	7.9	TL	**7.3**	TL	42.3	TL	23.6	472	8.5
50	1164	35	6	TL	6.7	TL	4.2	TL	4.6	TL	**2.5**	TL	6.4	TL	12.8	TL	7.0	236	2.6
50	1204	35	8	TL	9.9	TL	6.7	TL	6.2	TL	**5.5**	TL	7.2	TL	30.5	TL	10.5	175	5.8
50	1226	35	10	TL	23.9	TL	21.0	TL	11.7	TL	**9.2**	TL	28.8	TL	45.8	TL	26.5	219	10.2
50	1198	40	6	TL	49.3	TL	29.5	TL	15.6	TL	**15.2**	TL	41.4	TL	56.5	TL	48.9	878	17.9
50	1198	40	8	TL	67.5	TL	72.6	TL	16.2	TL	**15.0**	TL	19.6	TL	46.8	TL	75.4	TL	17.7
50	1208	40	10	TL	7.7	–	9.0	TL	5.0	TL	**3.9**	TL	9.1	TL	11.0	TL	11.2	88	4.1
50	1244	45	6	–	32.5	–	17.5	–	8.8	–	**8.7**	TL	8.9	TL	23.6	TL	18.2	25	9.5
50	1242	45	8	TL	31.7	TL	45.4	TL	**12.7**	TL	13.3	TL	14.9	TL	61.6	TL	50.4	44	15.3
50	1330	45	10	–	45.3	–	48.3	–	**13.9**	–	14.5	TL	23.4	TL	52.4	–	23.9	208	17.0

Table 5.2: Results for graphs with $|N| = 50$ and $d = 0.5$.

Instances				B+VI+H		C		PH	
$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Gap	t[s]	Gap	t[s]	Gap
100	4996	10	6	**101.22**	0.00	329.27	0.00	88.37	0.00
100	5058	10	8	**64.18**	0.00	65.08	0.00	35.65	0.00
100	5026	10	10	**78.33**	0.00	106.68	0.00	43.32	0.00
100	4974	15	6	**377.13**	0.00	7081.53	0.00	61.22	0.00
100	5000	15	8	**622.40**	0.00	855.60	0.00	32.49	0.40
100	4966	15	10	TL	2.42	TL	**1.14**	31.95	0.35
100	5030	20	6	**522.76**	0.00	TL	0.40	60.61	1.65
100	4918	20	8	TL	**2.27**	TL	2.77	48.25	2.30
100	4882	20	10	–	**6.26**	TL	9.38	63.97	0.00
100	4916	25	6	–	**10.47**	TL	24.34	110.61	0.42
100	4976	25	8	TL	1.42	**1772.46**	0.00	25.23	0.00
100	4988	25	10	**301.03**	0.00	6254.61	0.00	41.88	0.00
100	4896	30	6	TL	**4.04**	TL	5.88	57.02	0.00
100	4984	30	8	TL	**8.71**	TL	12.47	73.05	0.00
100	4866	30	10	–	**11.35**	TL	25.15	73.07	0.67
100	4966	35	6	TL	**6.12**	TL	16.30	49.54	0.25
100	4878	35	8	–	**7.83**	TL	9.99	69.30	0.58
100	4820	35	10	TL	**5.45**	TL	9.20	64.12	0.00
100	4986	40	6	–	**7.52**	TL	8.54	125.36	0.35
100	4946	40	8	–	**2.34**	TL	10.86	75.34	0.00
100	5060	40	10	TL	**5.97**	TL	14.85	74.16	0.00
100	4910	45	6	–	**13.93**	TL	29.04	156.51	0.04
100	4954	45	8	TL	0.58	TL	**0.06**	44.86	0.35
100	5050	45	10	–	**6.64**	TL	11.25	57.13	1.19
100	5026	50	6	–	**16.91**	TL	29.20	527.16	0.00
100	4954	50	8	–	**14.33**	TL	19.38	106.38	0.00
100	4974	50	10	–	**10.94**	TL	15.49	67.43	0.48

Table 5.3: Results for graphs with $|N| = 100$ and $d = 0.5$

.

Instances				Branch-and-Benders-Cut								C		RC		AB		PH	
				B		B+VI		B+PH		B+VI+PH									
$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap
25	538	10	6	74	0.0	85	0.0	51	0.0	51	0.0	**7**	0.0	63	0.0	38	0.0	4	0.0
25	538	10	8	38	0.0	37	0.0	56	0.0	62	0.0	**15**	0.0	35	0.0	44	0.0	19	0.0
25	536	10	10	35	0.0	16	0.0	79	0.0	42	0.0	**6**	0.0	236	0.0	281	0.0	21	0.0
25	534	15	6	15	0.0	18	0.0	14	0.0	13	0.0	**12**	0.0	118	0.0	33	0.0	3	0.0
25	542	15	8	53	0.0	104	0.0	121	0.0	**40**	0.0	452	0.0	617	0.0	96	0.0	5	0.0
25	542	15	10	177	0.0	**7**	0.0	16	0.0	17	0.0	15	0.0	284	0.0	93	0.0	9	0.0
25	548	20	6	25	0.0	**14**	0.0	43	0.0	18	0.0	18	0.0	4497	0.0	212	0.0	4	0.0
25	554	20	8	**12**	0.0	14	0.0	15	0.0	16	0.0	17	0.0	132	0.0	22	0.0	5	0.0
25	526	20	10	*TL*	2.8	7694	0.0	*TL*	2.6	1974	0.0	**915**	0.0	*TL*	3.5	*TL*	3.0	26	0.0
25	542	25	6	*TL*	2.2	8467	0.0	*TL*	2.2	2839	0.0	**112**	0.0	*TL*	1.8	*TL*	2.2	7	0.0
25	554	25	8	489	0.0	**191**	0.0	325	0.0	381	0.0	2545	0.0	4361	0.0	923	0.0	37	0.0
25	548	25	10	63	0.0	68	0.0	51	0.0	49	0.0	**35**	0.0	67	0.0	163	0.0	29	0.0
25	530	30	6	*TL*	0.5	6215	0.0	*TL*	0.5	5489	0.0	**87**	0.0	*TL*	1.0	*TL*	0.5	7	0.0
25	556	30	8	3203	0.0	2293	0.0	2859	0.0	**2020**	0.0	3422	0.0	*TL*	4.8	4502	0.0	78	0.0
25	544	30	10	*TL*	2.2	*TL*	1.5	*TL*	1.9	*TL*	1.7	*TL*	**0.3**	*TL*	4.1	*TL*	1.7	4	1.7
25	544	35	6	5380	0.0	612	0.0	6284	0.0	1055	0.0	**430**	0.0	*TL*	3.1	*TL*	0.5	6	0.0
25	560	35	8	*TL*	2.8	*TL*	1.6	*TL*	2.4	*TL*	**1.2**	*TL*	1.5	*TL*	5.4	*TL*	2.3	7	1.3
25	550	35	10	*TL*	0.6	479	0.0	*TL*	0.6	328	0.0	**226**	0.0	*TL*	0.7	*TL*	0.3	34	0.0
25	554	40	6	*TL*	2.2	*TL*	1.5	*TL*	2.4	*TL*	1.6	*TL*	**0.4**	*TL*	4.1	*TL*	2.0	48	1.6
25	540	40	8	*TL*	6.0	*TL*	4.4	*TL*	5.4	*TL*	**4.2**	*TL*	6.6	*TL*	9.9	*TL*	7.0	24	4.4
25	532	40	10	*TL*	1.7	*TL*	2.5	*TL*	1.7	*TL*	**0.8**	*TL*	1.5	*TL*	4.5	*TL*	2.7	12	0.9
25	528	45	6	*TL*	7.2	*TL*	8.8	1*TL*	5.0	*TL*	3.8	*TL*	**3.8**	*TL*	15.8	*TL*	12.5	25	4.4
25	524	45	8	*TL*	8.3	*TL*	6.0	*TL*	4.7	*TL*	**2.8**	*TL*	4.4	*TL*	8.9	*TL*	5.2	68	3.0
25	534	45	10	*TL*	2.6	*TL*	0.1	*TL*	1.7	**6748**	0.0	*TL*	2.2	*TL*	5.7	*TL*	2.3	9	0.0
25	548	50	6	*TL*	5.1	*TL*	5.7	*TL*	4.9	*TL*	4.5	*TL*	**4.0**	*TL*	7.8	*TL*	5.0	9	4.7
25	546	50	8	*TL*	4.7	*TL*	2.2	*TL*	2.7	*TL*	**0.0**	*TL*	1.5	*TL*	8.8	*TL*	2.4	117	0.9
25	542	50	10	–	6.2	–	6.7	*TL*	**4.6**	–	5.6	*TL*	4.7	*TL*	17.5	–	8.0	156	5.9

Table 5.4: Results for graphs with $|N| = 25$ and $d = 0.9$.

Instances				Branch-and-Benders-Cut								C		RC		AB		PH	
				B		B+VI		B+PH		B+VI+PH									
$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap
50	2208	10	6	171	0.0	207	0.0	103	0.0	52	0.0	**20**	0.0	234	0.0	237	0.0	14	0.0
50	2190	10	8	12	0.0	9	0.0	16	0.0	11	0.0	**3**	0.0	33	0.0	23	0.0	3	0.0
50	2216	10	10	34	0.0	28	0.0	37	0.0	**27**	0.0	56	0.0	273	0.0	62	0.0	9	0.0
50	2194	15	6	93	0.0	63	0.0	122	0.0	**37**	0.0	82	0.0	7322	0.0	263	0.0	12	0.0
50	2236	15	8	44	0.0	**24**	0.0	47	0.0	33	0.0	30	0.0	881	0.0	95	0.0	14	0.0
50	2222	15	10	134	0.0	230	0.0	88	0.0	**86**	0.0	165	0.0	1894	0.0	237	0.0	39	0.0
50	2204	20	6	–	0.9	2259	0.0	8048	0.0	**334**	0.0	5205	0.0	*TL*	8.6	2788	3.8	21	0.0
50	2204	20	8	1509	4.4	524	0.0	*TL*	2.7	**463**	0.0	*TL*	7.3	*TL*	13.4	*TL*	2.7	20	3.4
50	2192	20	10	*TL*	2.3	*TL*	2.4	*TL*	1.7	*TL*	**0.3**	*TL*	3.3	*TL*	9.6	*TL*	3.0	25	1.2
50	2220	25	6	*TL*	1.1	826	0.0	*TL*	0.4	586	0.0	**247**	0.0	*TL*	1.1	*TL*	0.5	65	0.1
50	2180	25	8	78	0.0	115	0.0	**63**	0.0	65	0.0	140	0.0	1426	0.0	281	0.0	16	0.0
50	2222	25	10	520	0.0	677	0.0	486	0.0	601	0.0	**483**	0.0	8255	0.0	885	0.0	21	0.0
50	2206	30	6	**431**	0.0	626	0.0	1105	0.0	823	0.0	5144	0.0	*TL*	9.1	**1447**	0.0	22	0.0
50	2190	30	8	3226	0.0	175	0.0	2623	0.0	**162**	0.0	3557	0.0	*TL*	1.3	5275	0.0	63	0.0
50	2212	30	10	–	6.8	–	9.3	–	5.1	–	**3.5**	*TL*	8.8	*TL*	40.3	–	14.3	56	3.6
50	2216	35	6	–	9.2	–	8.9	–	6.8	–	**3.0**	*TL*	7.3	*TL*	20.9	–	7.5	34	3.2
50	2208	35	8	–	1.0	2678	0.0	2136	0.0	1564	0.0	**742**	0.0	*TL*	2.3	–	0.9	25	0.0
50	2192	35	10	–	**1.5**	–	6.9	–	3.7	–	2.6	*TL*	7.6	*TL*	30.2	–	1.4	102	2.6
50	2188	40	6	–	11.4	–	10.8	*TL*	6.2	*TL*	**2.5**	*TL*	7.8	*TL*	35.0	–	10.0	55	2.7
50	2190	40	8	*TL*	4.0	*TL*	2.4	*TL*	3.4	*TL*	**2.2**	*TL*	2.3	*TL*	5.8	*TL*	5.6	128	2.3
50	2204	40	10	*TL*	3.9	*TL*	4.2	*TL*	3.5	*TL*	3.9	*TL*	**3.0**	*TL*	21.3	*TL*	6.4	394	4.1
50	2192	45	6	–	17.7	*TL*	8.7	–	6.4	*TL*	**5.6**	*TL*	6.9	*TL*	19.5	–	13.0	103	6.0
50	2178	45	8	*TL*	0.4	*TL*	0.1	*TL*	0.4	*TL*	0.1	**8566**	0.0	*TL*	19.3	*TL*	0.4	60	0.1
50	2220	45	10	–	7.8	–	11.0	*TL*	**5.5**	*TL*	5.6	*TL*	7.4	*TL*	25.8	*TL*	15.2	709	5.9
50	2202	50	6	–	25.3	*TL*	32.9	–	9.5	–	**8.1**	*TL*	17.8	*TL*	54.3	*TL*	32.6	182	8.9
50	2192	50	8	–	1.7	*TL*	**1.1**	*TL*	1.5	*TL*	1.8	*TL*	3.1	*TL*	27.2	*TL*	6.6	96	1.8
50	2238	50	10	–	9.3	–	6.7	*TL*	9.3	*TL*	6.3	*TL*	**5.7**	*TL*	15.9	*TL*	12.0	348	7.5

Table 5.5: Results for graphs with $|N| = 50$ and $d = 0.9$.

Instances				Branch-and-Benders-Cut								C		RC		AB		PH	
				B		B+VI		B+PH		B+VI+PH									
$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap	t[s]	Gap
100	8946	10	6	126	0.0	158	0.0	195	0.0	140	0.0	**123**	0.0	4967	0.0	542	0.0	48	4.1
100	8894	10	8	102	0.0	96	0.0	118	0.0	82	0.0	**75**	0.0	1029	0.0	442	0.0	40	0.0
100	8876	10	10	546	0.0	**241**	0.0	523	0.0	257	0.0	580	0.0	*TL*	3.1	3921	0.0	34	3.8
100	8946	15	6	*TL*	1.1	435	0.0	*TL*	1.1	**425**	0.0	1752	0.0	*TL*	6.0	*TL*	1.5	92	0.0
100	8914	15	8	502	0.0	296	0.0	551	0.0	**265**	0.0	1522	0.0	*TL*	10,8	3368	0.0	57	0.0
100	8902	15	10	225	0.0	206	0.0	230	0.0	**186**	0.0	1350	0.0	*TL*	5.4	1151	0.0	44	0.0
100	9010	20	6	1124	0.0	194	0.0	616	0.0	**170**	0.0	654	0.0	*TL*	7.4	4749	0.0	73	0.0
100	8960	20	8	534	0.0	297	0.0	518	0.0	**249**	0.0	2956	0.0	*TL*	54.8	3330	0.0	81	0.0
100	8938	20	10	1211	0.0	213	0.0	1012	0.0	**171**	0.0	550	0.0	*TL*	1.1	5889	0.0	86	3.2
100	8936	25	6	*TL*	4.3	2925	0.0	*TL*	3.9	**2334**	0.0	7806	0.0	*TL*	6.2	*TL*	4.3	91	3.0
100	8840	25	8	*TL*	3.5	*TL*	3.0	*TL*	4.5	*TL*	**3.0**	*TL*	3.8	*TL*	78.7	*TL*	13.9	98	4.9
100	8954	25	10	212	0.0	227	0.0	253	0.0	**212**	0.0	582	0.0	*TL*	7.6	1516	0.0	55	2.1
100	8974	30	6	4771	0.0	1689	0.0	1849	0.0	**1016**	0.0	5076	0.0	*TL*	37.2	8959	0.0	77	0.0
100	8860	30	8	*TL*	0.7	5728	0.0	*TL*	0.0	**1180**	0.0	*TL*	1.0	*TL*	21.4	*TL*	5.2	100	0.4
100	8990	30	10	*TL*	1.2	1051	0.0	*TL*	0.7	**700**	0.0	9317	0.0	*TL*	39.3	*TL*	0.9	99	4.1
100	8942	35	6	*TL*	0.9	3448	0.0	*TL*	0.0	**1008**	0.0	*TL*	4.8	*TL*	72.6	*TL*	1.4	153	2.9
100	8936	35	8	*TL*	5.9	*TL*	5.7	*TL*	5.7	*TL*	**2.6**	*TL*	16.1	*TL*	81.5	*TL*	45.7	142	4.1
100	8922	35	10	*TL*	1.7	*TL*	0.7	*TL*	1.9	*TL*	**0.9**	*TL*	14.9	*TL*	75.8	*TL*	26.4	232	1.6
100	8904	40	6	*TL*	0.0	*TL*	0.0	*TL*	0.0	*TL*	**0.0**	*TL*	0.1	*TL*	0.8	*TL*	0.8	184	1.4
100	8924	40	8	*TL*	0.3	*TL*	0.1	*TL*	0.0	*TL*	**0.0**	*TL*	0.2	*TL*	0.8	*TL*	0.7	*TL*	7.8
100	8904	40	10	*TL*	0.1	*TL*	0.0	*TL*	0.0	*TL*	**0.0**	*TL*	0.3	*TL*	0.7	*TL*	0.5	148	3.4
100	8926	45	6	3265	0.0	3115	0.0	1927	0.0	**1118**	0.0	4804	0.0	*TL*	0.8	7859	0.0	191	7.4

Table 5.6: Results for graphs with $|N| = 100$ and $d = 0.9$.

Instances					Branch-and-Benders-Cut				C	RC	AB	PH
					B	B+VI	B+PH	B+VI+PH				
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]
$Abilene_1$	12	30	132	6	–	–	8.35	**8.35**	–	–	–	0.33
$Abilene_2$	12	30	132	6	–	–	**9.45**	9.46	–	–	–	0.09
$Abilene_3$	12	30	132	6	–	–	**9.21**	9.22	–	–	–	0.03
$Abilene_4$	12	30	132	6	–	–	10.03	**10.03**	–	–	–	0.20
$Abilene_5$	12	30	132	6	–	–	10.96	**10.96**	–	–	–	0.04
$Abilene_6$	12	30	132	6	–	–	11.82	**11.82**	–	–	–	0.05
$Abilene_7$	12	30	132	6	–	–	7.45	**7.45**	–	–	–	0.09
$Abilene_8$	12	30	132	6	–	–	9.48	**9.48**	–	–	–	0.11
$Abilene_9$	12	30	132	6	–	–	14.75	**14.76**	–	–	–	0.22
$Abilene_{10}$	12	30	132	6	–	–	9.78	**9.78**	–	–	–	0.03
$Atlanta_1$	15	44	210	6	–	–	21.01	**21.01**	–	–	–	1.27
$Atlanta_2$	15	44	210	6	–	–	14.77	**14.77**	–	–	–	2.05
$Atlanta_3$	15	44	210	6	–	–	17.63	**17.63**	–	–	–	1.46
$Atlanta_4$	15	44	210	6	–	–	45.82	**45.82**	–	–	–	53.61
$Atlanta_5$	15	44	210	6	–	–	26.11	**26.11**	–	–	–	13.38
$Atlanta_6$	15	44	210	6	–	–	11.56	**11.56**	–	–	–	0.81
$Atlanta_7$	15	44	210	6	–	–	42.88	**42.88**	–	–	–	48.00
$Atlanta_8$	15	44	210	6	–	–	20.08	**20.08**	–	–	–	8.38
$Atlanta_9$	15	44	210	6	–	–	41.00	**41.00**	–	–	–	42.61
$Atlanta_{10}$	15	44	210	6	–	–	16.19	**16.19**	–	–	–	0.43
$Dfn-bwin_1$	10	90	90	6	5.06	5.06	3.49	3.49	**2.67**	4.30	9.05	2.18
$Dfn-bwin_2$	10	90	90	6	3.75	3.84	2.17	2.18	**1.95**	4.67	6.39	1.02
$Dfn-bwin_3$	10	90	90	6	12.82	12.82	3.81	3.81	**3.60**	6.27	8.80	1.88
$Dfn-bwin_4$	10	90	90	6	6.55	6.55	2.18	2.35	**1.81**	3.18	6.44	1.68
$Dfn-bwin_5$	10	90	90	6	6.71	6.71	3.71	3.63	**2.84**	4.40	9.47	2.03
$Dfn-bwin_6$	10	90	90	6	4.32	4.32	2.98	2.98	**2.71**	4.36	6.49	1.45
$Dfn-bwin_7$	10	90	90	6	13.13	13.13	4.47	4.47	**3.17**	5.19	7.16	1.91
$Dfn-bwin_8$	10	90	90	6	4.85	4.86	5.14	5.14	**3.79**	4.45	6.41	1.69
$Dfn-bwin_9$	10	90	90	6	3.02	2.99	1.49	1.49	**1.47**	1.78	1.96	0.64
$Dfn-bwin_{10}$	10	90	90	6	14.38	14.38	5.40	5.40	**4.57**	6.05	11.64	1.19
$Dfn-gwin_1$	11	94	110	6	19.89	19.91	9.65	9.65	**5.00**	9.19	19.42	4.75
$Dfn-gwin_2$	11	94	110	6	19.55	19.55	7.91	7.93	–	**5.47**	19.05	3.35
$Dfn-gwin_3$	11	94	110	6	–	–	8.45	8.45	**4.79**	6.21	25.78	4.31
$Dfn-gwin_4$	11	94	110	6	22.03	22.03	6.53	6.53	**4.71**	7.86	20.36	1.89
$Dfn-gwin_5$	11	94	110	6	21.30	21.30	9.60	9.60	**6.83**	10.79	21.07	2.59
$Dfn-gwin_6$	11	94	110	6	–	–	9.02	9.02	–	**7.95**	24.18	1.66
$Dfn-gwin_7$	11	94	110	6	–	–	8.00	8.00	**4.10**	7.49	22.98	4.45
$Dfn-gwin_8$	11	94	110	6	18.90	18.90	5.67	5.67	**3.27**	5.49	17.96	2.35
$Dfn-gwin_9$	11	94	110	6	–	–	7.79	7.79	**5.64**	11.21	21.78	2.35
$Dfn-gwin_{10}$	11	94	110	6	25.57	25.57	9.61	9.61	**8.39**	8.69	24.46	1.31

Table 5.7: Results for SNDlib instances with node capacities and conflict constraints (part 1)

.

Instances					Branch-and-Benders-Cut				C	RC	AB	PH
					B	B+VI	B+PH	B+VI+PH				
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]
$Di-yuan_1$	11	84	22	6	39.49	39.49	10.51	**8.39**	11.52	11.75	30.51	2.88
$Di-yuan_2$	11	84	22	6	37.91	37.71	11.38	**8.39**	15.96	25.57	34.39	2.96
$Di-yuan_3$	11	84	22	6	34.21	34.21	8.88	8.89	**8.39**	–	32.76	5.59
$Di-yuan_4$	11	84	22	6	–	–	**10.09**	10.09	10.30	–	–	3.07
$Di-yuan_5$	11	84	22	6	31.89	31.90	9.35	**9.35**	10.82	10.72	34.97	4.99
$Di-yuan_6$	11	84	22	6	42.18	42.19	**9.12**	9.13	9.85	20.12	38.33	2.16
$Di-yuan_7$	11	84	22	6	–	–	4.85	4.85	**2.25**	9.64	–	4.18
$Di-yuan_8$	11	84	22	6	39.06	–	12.15	11.33	**9.89**	18.17	32.55	6.18
$Di-yuan_9$	11	84	22	6	–	–	15.12	**15.12**	16.88	20.22	–	6.01
$Di-yuan_{10}$	11	84	22	6	35.72	35.72	10.51	**10.51**	13.30	14.92	33.39	0.00
$Newyork_1$	16	98	240	6	–	–	46.66	46.66	–	46.82	**46.46**	32.70
$Newyork_2$	16	98	240	6	–	–	19.92	**19.92**	–	39.54	39.97	1.60
$Newyork_3$	16	98	240	6	–	–	41.53	41.53	–	**37.93**	41.63	43.05
$Newyork_4$	16	98	240	6	–	–	45.56	45.56	–	**41.67**	45.69	31.82
$Newyork_5$	16	98	240	6	–	–	16.01	**16.01**	–	38.54	42.69	8.15
$Newyork_6$	16	98	240	6	–	–	18.63	**18.63**	–	38.64	40.19	10.05
$Newyork_7$	16	98	240	6	–	–	36.97	36.97	–	37.36	**36.83**	32.40
$Newyork_8$	16	98	240	6	–	–	40.02	40.02	–	**38.46**	39.91	43.57
$Newyork_9$	16	98	240	6	–	–	20.78	**20.78**	–	36.65	46.11	6.94
$Newyork_{10}$	16	98	240	6	–	–	22.42	**22.42**	–	38.13	43.63	10.23
$Nobel-ger_1$	17	52	121	6	–	–	–	–	–	–	–	–
$Nobel-ger_2$	17	52	121	6	–	–	18.87	**18.87**	–	–	–	1.45
$Nobel-ger_3$	17	52	121	6	–	–	14.18	**14.18**	–	–	–	6.35
$Nobel-ger_4$	17	52	121	6	–	–	–	–	–	–	–	–
$Nobel-ger_5$	17	52	121	6	–	–	18.17	**18.17**	–	–	–	4.09
$Nobel-ger_6$	17	52	121	6	–	–	–	–	–	–	–	–
$Nobel-ger_7$	17	52	121	6	–	–	12.33	**12.34**	–	–	–	0.47
$Nobel-ger_8$	17	52	121	6	–	–	18.42	**18.42**	–	–	–	1.11
$Nobel-ger_9$	17	52	121	6	–	–	22.33	**22.33**	–	–	–	8.54
$Nobel-ger_{10}$	17	52	121	6	–	–	–	–	–	–	–	–
$Nobel-us_1$	14	42	91	6	–	–	15.15	**15.15**	–	–	–	1.80
$Nobel-us_2$	14	42	91	6	–	–	10.58	**10.58**	–	–	–	1.21
$Nobel-us_3$	14	42	91	6	–	–	5.28	**5.28**	–	–	–	1.57
$Nobel-us_4$	14	42	91	6	–	–	12.29	**12.29**	–	–	–	1.71
$Nobel-us_5$	14	42	91	6	–	–	13.67	**13.67**	–	–	–	1.74
$Nobel-us_6$	14	42	91	6	–	–	10.29	**10.29**	–	–	–	0.87
$Nobel-us_7$	14	42	91	6	–	–	14.14	**14.14**	–	–	–	1.80
$Nobel-us_8$	14	42	91	6	–	–	37.34	**37.34**	–	–	–	37.98
$Nobel-us_9$	14	42	91	6	–	–	15.62	**15.62**	–	–	–	4.64
$Nobel-us_{10}$	14	42	91	6	–	–	18.82	**18.82**	–	–	–	3.07

Table 5.8: Results for SNDlib instances with node capacities and conflict constraints (part 2).

Instances					Branch-and-Benders-Cut				C	RC	AB	PH
					B	B+VI	B+PH	B+VI+PH				
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]
Pdh_1	11	68	24	6	0.45	0.46	0.51	0.51	0.51	**0.15**	0.49	0.20
Pdh_2	11	68	24	6	2.48	2.48	2.03	2.03	1.82	**1.80**	3.19	0.77
Pdh_3	11	68	24	6	1.36	1.36	1.80	1.80	1.22	**0.95**	1.64	0.98
Pdh_4	11	68	24	6	3.90	3.90	2.74	2.74	**1.89**	2.12	6.04	0.83
Pdh_5	11	68	24	6	5.05	5.05	3.91	3.91	**2.89**	3.36	5.44	1.27
Pdh_6	11	68	24	6	12.88	12.88	5.62	5.62	7.06	**4.84**	–	0.83
Pdh_7	11	68	24	6	3.33	3.33	2.81	2.81	**2.77**	2.86	3.19	0.13
Pdh_8	11	68	24	6	0.77	0.77	0.45	0.45	0.31	**0.28**	0.53	0.34
Pdh_9	11	68	24	6	3.70	3.70	2.87	2.87	2.83	**2.55**	3.28	0.31
Pdh_{10}	11	68	24	6	2.07	2.07	2.10	2.11	2.21	**2.03**	2.09	0.00
$Polska_1$	12	36	66	6	–	–	9.19	**9.19**	18.96	–	–	2.25
$Polska_2$	12	36	66	6	–	–	14.36	**14.36**	18.99	–	–	2.36
$Polska_3$	12	36	66	6	–	–	7.65	**7.65**	14.31	–	–	0.67
$Polska_4$	12	36	66	6	–	–	13.50	**13.50**	19.96	–	–	2.60
$Polska_5$	12	36	66	6	–	–	15.85	**15.85**	19.03	–	–	2.02
$Polska_6$	12	36	66	6	–	–	17.01	**17.01**	25.85	–	–	3.30
$Polska_7$	12	36	66	6	–	–	13.22	**13.22**	12.84	–	–	2.42
$Polska_8$	12	36	66	6	–	–	7.73	**7.73**	8.85	–	–	0.87
$Polska_9$	12	36	66	6	–	–	9.97	**9.97**	11.71	–	–	2.16
$Polska_{10}$	12	36	66	6	–	–	12.08	**12.08**	15.61	–	–	1.76

Table 5.9: Results for SNDlib instances with node capacities and conflict constraints (part 3).

Instances					Branch-and-Benders-Cut				C		RC		AB		PH	
					B		B+VI		B+PH		B+VI+PH					
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap[%]		Gap[%]		Gap[%]		Gap[%]		Gap[%]		Gap[%]	
$Abilene_1$	12	30	132	6	–		–		6.42		**6.42**		4.49		8.36		–		0.00	
$Abilene_2$	12	30	132	6	–		–		9.45		9.45		**7.14**		–		–		0.00	
$Abilene_3$	12	30	132	6	–		–		7.12		7.12		**6.41**		–		–		0.04	
$Abilene_4$	12	30	132	6	–		–		9.96		9.97		**6.58**		–		–		0.04	
$Abilene_5$	12	30	132	6	–		–		5.10		5.11		**3.24**		5.66		–		0.00	
$Abilene_6$	12	30	132	6	–		–		8.41		8.41		**6.93**		9.90		–		0.00	
$Abilene_7$	12	30	132	6	–		–		7.62		7.63		**5.24**		–		–		0.00	
$Abilene_8$	12	30	132	6	–		–		8.18		8.18		**7.41**		11.00		–		0.00	
$Abilene_9$	12	30	132	6	–		–		15.98		15.98		**15.46**		–		–		0.06	
$Abilene_{10}$	12	30	132	6	–		–		7.28		7.28		**4.17**		9.10		–		0.00	
$Atlanta_1$	15	44	210	6	–		–		19.02		**19.02**		44.20		–		–		0.53	
$Atlanta_2$	15	44	210	6	–		–		**13.65**		13.66		21.72		–		–		0.27	
$Atlanta_3$	15	44	210	6	–		–		13.41		**13.41**		24.53		–		–		0.60	
$Atlanta_4$	15	44	210	6	–		–		14.35		**14.35**		17.98		–		–		1.19	
$Atlanta_5$	15	44	210	6	–		–		8.69		**8.69**		13.81		–		–		0.02	
$Atlanta_6$	15	44	210	6	–		–		12.52		**12.52**		16.02		–		–		0.35	
$Atlanta_7$	15	44	210	6	–		–		**18.29**		18.30		48.86		–		–		0.00	
$Atlanta_8$	15	44	210	6	–		–		16.63		**16.63**		27.49		–		–		0.45	
$Atlanta_9$	15	44	210	6	–		–		12.73		**12.73**		23.08		–		–		0.17	
$Atlanta_{10}$	15	44	210	6	–		–		17.76		**17.76**		22.50		–		–		0.73	
$Dfn-bwin_1$	10	90	90	6	7.23		7.28		3.11		3.16		**2.69**		4.32		6.27		2.16	
$Dfn-bwin_2$	10	90	90	6	1.60		1.65		1.63		1.65		**1.46**		3.27		5.03		0.94	
$Dfn-bwin_3$	10	90	90	6	5.04		5.04		3.36		3.36		**3.03**		4.86		8.57		1.38	
$Dfn-bwin_4$	10	90	90	6	1.27		1.27		**1.10**		1.20		1.13		1.36		1.14		0.97	
$Dfn-bwin_5$	10	90	90	6	8.83		8.83		2.91		2.91		**2.38**		2.44		21.45		0.60	
$Dfn-bwin_6$	10	90	90	6	2.88		3.26		2.57		2.57		**2.46**		4.07		5.13		1.49	
$Dfn-bwin_7$	10	90	90	6	15.72		15.72		3.31		3.77		**2.33**		3.38		6.72		1.95	
$Dfn-bwin_8$	10	90	90	6	3.07		3.07		1.85		1.85		**1.43**		1.45		2.69		1.36	
$Dfn-bwin_9$	10	90	90	6	1.31		1.31		1.15		1.16		**1.05**		1.37		1.25		0.54	
$Dfn-bwin_{10}$	10	90	90	6	5.39		5.39		4.81		4.81		**3.24**		4.35		8.47		1.66	
$Dfn-gwin_1$	11	94	110	6	20.05		20.05		9.35		9.35		**4.13**		5.97		20.05		5.19	
$Dfn-gwin_2$	11	94	110	6	20.63		20.63		8.13		8.15		**3.75**		4.67		19.67		4.86	
$Dfn-gwin_3$	11	94	110	6	–		–		5.41		5.52		**4.26**		4.78		–		1.42	
$Dfn-gwin_4$	11	94	110	6	1.66		1.66		1.18		1.18		**0.65**		0.86		6.96		1.98	
$Dfn-gwin_5$	11	94	110	6	21.47		21.47		7.54		7.54		**5.16**		7.43		21.00		2.28	
$Dfn-gwin_6$	11	94	110	6	10.34		10.28		6.75		6.75		**3.26**		4.10		–		3.80	
$Dfn-gwin_7$	11	94	110	6	24.14		23.96		7.52		7.52		**2.58**		3.34		23.69		5.47	
$Dfn-gwin_8$	11	94	110	6	20.06		20.07		4.09		4.10		**2.60**		3.09		18.96		1.52	
$Dfn-gwin_9$	11	94	110	6	22.50		22.51		7.85		7.85		**5.19**		6.29		–		3.02	
$Dfn-gwin_{10}$	11	94	110	6	25.65		25.65		8.72		8.72		**6.08**		7.22		24.98		2.65	

Table 5.10: Results for SNDlib instances without node capacities and conflict constraints (part1).

Instances					Branch-and-Benders-Cut				C	RC	AB	PH
					B	B+VI	B+PH	B+VI+PH				
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]
$Di-yuan_1$	11	84	22	6	8.01	8.01	5.70	5.70	**5.05**	5.60	8.07	1.43
$Di-yuan_2$	11	84	22	6	–	–	7.60	7.65	**7.32**	7.80	–	0.11
$Di-yuan_3$	11	84	22	6	14.76	15.03	3.39	3.39	**2.77**	2.94	5.11	0.90
$Di-yuan_4$	11	84	22	6	1.49	1.58	1.16	1.17	1.17	**1.01**	1.28	1.25
$Di-yuan_5$	11	84	22	6	20.87	20.31	12.22	12.43	11.23	**11.19**	12.65	1.51
$Di-yuan_6$	11	84	22	6	–	–	14.46	14.49	13.91	**13.67**	24.11	1.73
$Di-yuan_7$	11	84	22	6	1.06	1.06	0.84	0.83	**0.45**	–	0.45	7.13
$Di-yuan_8$	11	84	22	6	19.04	5.08	5.75	**3.28**	3.65	3.71	24.13	2.51
$Di-yuan_9$	11	84	22	6	20.27	20.27	11.83	11.83	9.83	**9.54**	12.32	2.15
$Di-yuan_{10}$	11	84	22	6	–	–	5.31	**5.30**	6.64	6.78	–	3.26
$Newyork_1$	16	98	240	6	53.38	53.38	26.98	**26.98**	–	45.10	53.38	2.77
$Newyork_2$	16	98	240	6	–	–	17.57	**17.57**	26.59	30.28	–	7.48
$Newyork_3$	16	98	240	6	–	–	13.13	**13.13**	22.92	30.13	–	7.67
$Newyork_4$	16	98	240	6	–	–	15.71	**15.71**	–	29.36	–	7.97
$Newyork_5$	16	98	240	6	–	–	13.85	**13.85**	17.85	27.15	–	5.76
$Newyork_6$	16	98	240	6	–	–	24.58	**24.58**	30.41	36.56	–	13.62
$Newyork_7$	16	98	240	6	–	–	17.48	**17.48**	–	32.57	–	7.12
$Newyork_8$	16	98	240	6	–	–	22.23	**22.23**	24.22	28.82	–	14.25
$Newyork_9$	16	98	240	6	50.11	50.11	19.39	**19.39**	–	35.77	50.15	7.52
$Newyork_{10}$	16	98	240	6	45.11	45.11	20.15	**20.15**	20.18	25.92	45.00	11.36
$Nobel-ger_1$	17	52	121	6	–	–	14.45	**14.45**	15.90	–	–	6.16
$Nobel-ger_2$	17	52	121	6	–	–	18.02	**18.02**	30.19	–	–	1.32
$Nobel-ger_3$	17	52	121	6	–	–	8.14	**8.14**	11.19	–	–	0.43
$Nobel-ger_4$	17	52	121	6	–	–	19.86	**19.86**	28.85	–	–	5.01
$Nobel-ger_5$	17	52	121	6	–	–	15.18	**15.18**	23.36	–	–	2.57
$Nobel-ger_6$	17	52	121	6	–	–	11.17	**11.17**	17.62	–	–	0.74
$Nobel-ger_7$	17	52	121	6	–	–	**14.54**	14.55	28.29	–	–	0.88
$Nobel-ger_8$	17	52	121	6	–	–	**24.22**	24.23	34.30	–	–	1.21
$Nobel-ger_9$	17	52	121	6	–	–	14.01	**14.01**	15.79	–	–	1.77
$Nobel-ger_{10}$	17	52	121	6	–	–	14.77	**14.77**	26.37	–	–	1.19
$Nobel-us_1$	14	42	91	6	–	–	10.42	**10.42**	14.00	–	–	0.00
$Nobel-us_2$	14	42	91	6	–	–	5.68	**5.68**	16.08	16.95	–	0.12
$Nobel-us_3$	14	42	91	6	–	–	6.86	**6.87**	11.58	17.79	–	0.72
$Nobel-us_4$	14	42	91	6	36.77	36.77	10.71	**10.71**	13.02	17.64	–	0.74
$Nobel-us_5$	14	42	91	6	–	–	**10.59**	10.60	19.10	–	–	0.39
$Nobel-us_6$	14	42	91	6	–	–	7.63	**7.63**	14.12	–	–	1.01
$Nobel-us_7$	14	42	91	6	43.39	43.39	12.43	**12.43**	17.66	–	–	1.93
$Nobel-us_8$	14	42	91	6	41.10	41.21	14.03	**14.03**	18.78	–	–	0.47
$Nobel-us_9$	14	42	91	6	–	–	11.46	**11.46**	19.36	24.37	–	0.11
$Nobel-us_{10}$	14	42	91	6	–	–	23.35	**23.35**	29.13	–	–	1.25

Table 5.11: Results for SNDlib instances without node capacities and conflict constraints (part 2).

Instances					Branch-and-Benders-Cut				C	RC	AB	PH
					B	B+VI	B+PH	B+VI+PH				
Name	$\|N\|$	$\|A\|$	$\|C\|$	$\|F\|$	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]	Gap[%]
Pdh_1	11	68	24	6	0.23	0.24	**0.18**	0.19	0.28	0.23	0.22	0.21
Pdh_2	11	68	24	6	1.27	1.27	1.17	1.17	**0.73**	0.82	1.08	0.75
Pdh_3	11	68	24	6	0.88	0.89	0.86	0.86	**0.56**	0.61	0.89	0.90
Pdh_4	11	68	24	6	1.08	1.08	1.19	1.20	1.00	**0.83**	1.04	1.18
Pdh_5	11	68	24	6	**2.21**	2.23	2.30	2.32	2.34	2.33	2.56	1.21
Pdh_6	11	68	24	6	4.19	4.19	3.63	3.63	3.62	**3.51**	3.78	0.38
Pdh_7	11	68	24	6	**1.26**	1.26	1.29	1.30	1.35	1.26	1.27	0.08
Pdh_8	11	68	24	6	0.36	0.36	0.46	0.46	**0.13**	0.25	0.53	0.21
Pdh_9	11	68	24	6	**0.06**	0.06	0.11	0.13	0.21	0.15	0.06	0.09
Pdh_{10}	11	68	24	6	0.22	0.25	**0.20**	0.20	0.30	0.30	0.20	0.00
$Polska_1$	12	36	66	6	42.82	42.87	8.12	**8.12**	10.12	16.50	–	0.00
$Polska_2$	12	36	66	6	–	–	**9.84**	9.85	11.37	18.83	–	2.04
$Polska_3$	12	36	66	6	42.78	42.90	**9.20**	9.21	11.11	–	43.26	0.00
$Polska_4$	12	36	66	6	–	–	6.58	**6.58**	9.67	–	–	0.03
$Polska_5$	12	36	66	6	–	–	**13.86**	13.87	9.11	–	–	0.00
$Polska_6$	12	36	66	6	–	–	**18.48**	18.49	18.82	–	–	0.24
$Polska_7$	12	36	66	6	29.79	29.79	4.99	5.00	**3.99**	5.30	–	0.05
$Polska_8$	12	36	66	6	–	–	3.77	3.78	**2.89**	4.78	–	0.00
$Polska_9$	12	36	66	6	–	–	**5.64**	5.65	9.43	14.92	–	0.00
$Polska_{10}$	12	36	66	6	–	–	8.68	8.68	**8.64**	20.67	–	0.14

Table 5.12: Results for SNDlib instances without node capacities and conflict constraints (part 3).

5.5 Conclusions

In this chapter we studied two variants of the Virtual Network Functions Placement and Routing problem (VNFPRP). We provided theoretical results that allowed us to reformulate the problem using Benders decomposition. We proposed three families of valid inequalities to strengthen the LP-bounds.

All these ingredients have been combined in a Branch-and-Benders-Cut framework and tested on a set of realistic benchmark instances. The obtained results by Branch-and-Benders-Cut algorithm and the compact MILP formulation have been compared with the Automatic Benders decomposition provided by Cplex. Computational results have shown that our Branch-and-Benders-Cut algorithm is more efficient compared to the compact MILP formulation and the Automatic Benders of Cplex, in terms of solution quality and CPU time.

Conclusions

In this dissertation, we have studied the Virtual Network Functions Placement and Routing Problem (VNFPRP), for which the sum of the function installation and node activation costs has to be minimized. The studied problem was considered with routing and latency constraints, node and function capacity constraints, incompatibility and chaining constraints. We have shown that the problem is strongly NP-hard even for its simplest version.

In the first part of the thesis, we have investigated the basic properties of the problem and proposed a compact MILP formulation. This formulation does not seem to be strong enough for finding a solution using an off-the-shelf solver. To tackle the problem from a computational perspective, we have proposed a path-based heuristic that provides optimal solutions for some realistic instances from the literature.

Afterwards, we have proposed two extended formulations to model the problem: path formulation and Dantzig-Wolfe formulation. In order to strengthen the LP-bounds, we have proposed several families of valid inequalities and have demonstrated their benefits. We have shown that the LP-bounds of Dantzig-Wolfe formulation are stronger than the LP-bounds of the path formulation. We have presented a branching scheme for each formulation and developed a respective Branch-and-Price algorithm. We have computationally compared both algorithms with the MILP compact formulation and the automatic Benders of Cplex.

In the last part of the dissertation, we have studied a variant of the problem in which the VNFs-capacity and conflict constraints are relaxed. We provided theoretical results that allowed us to reformulate the problem using Benders decomposition and three families of valid inequalities to strengthen the LP-bounds. All these ingredients have been combined in a Branch-and-Benders-Cut framework and tested on a set of realistic benchmark instances.

As perspectives, there are diverse directions for which our future research associated with the VNFPRP can be conducted.

First, it would be interesting to formulate the problem using bi-level optimization. The leader problem takes care of the VNFs-installation part, and the follower problems routes the traffic taking into account these installations. Also, we can tackle the problem using two-stage optimization either to fix the VNFs-installation on nodes, or to fix the routing paths. This approach would allow a heuristic to obtain a larger instances, high-quality solutions.

Moreover, more efficient separation heuristics and more sophisticated preprocessing methods can be developed in order to improve the resolution of the problem. Furthermore, some meta-heuristics can be used, such as the ant colony optimization algorithm (ACO) or the genetic algorithm. Also, investigating new valid inequalities for the problem would be of interest.

It would be also interesting to further investigate different column generation strategies and heuristics to solve the pricing problems, test different branching schemes and define a gap value for the Lagrangian bound to improve the convergence of our Branch-and-Price algorithms.

Finally, we can use robust optimization to deal with the uncertainty of some parameters, for example, the number of traffic requests in a given period (hour, day, week, ...), in order to minimize the costs generated by the unplanned demands.

Bibliography

[1] https://www.networkworld.com/article/3239677/the-osi-model-explained-how-to-understand-and-remember-the-7-layer-network-m
html.

[2] https://tools.ietf.org/html/rfc3234.

[3] Network Devices-Hub, Switch, Router, etc, 2019. https://networkustad.com/2019/05/27/network-devices-hub-switch-router/.

[4] A Cheat Sheet for Understanding "NFV Architecture", March 17, 2015. https://telcocloudbridge.com/wp-content/uploads/2015/03/NFA-Architecture.png.

[5] Bernardetta Addis, Dallal Belabed, Mathieu Bouet, and Stefano Secci. Virtual network functions placement and routing optimization. In *Cloud Networking (CloudNet), 2015 IEEE 4th International Conference on*, pages 171–177. IEEE, 2015.

[6] Bernardetta Addis, Giuliana Carello, Francesca De Bettin, and Meihui Gao. On a Virtual Network Function Placement and Routing problem: properties and formulations. working paper or preprint, November 2018.

[7] Bernardetta Addis, Giuliana Carello, and Meihui Gao. On a virtual network functions placement and routing problem: Some properties and a comparison of two formulations. *Networks*, 75(2):158–182, 2020.

[8] Ilan Adler and A Ulkücü. On the number of iterations in dantzig-wolfe decomposition. *Decomposition of large scale problems*, pages 181–187, 1973.

[9] Ian F Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30, 2014.

[10] Omar Alhussein, Phu Thinh Do, Junling Li, Qiang Ye, Weisen Shi, Weihua Zhuang, Xuemin Shen, Xu Li, and Jaya Rao. Joint VNF placement and multicast traffic routing in 5G core networks. In *2018 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2018.

[11] Zaid Allybokus, Nancy Perrot, Jérémie Leguay, Lorenzo Maggi, and Eric Gourdin. Virtual function placement for service chaining with partial orders and anti-affinity rules. *Networks*, 71(2):97–106, 2018.

[12] C. Alves and J.M. Valério de Carvalho. A stabilized branch-and-price-and-cut algorithm for the multiple length cutting stock problem. *Computers & Operations Research*, 35(4):1315 – 1328, 2008.

[13] Annotating a model for CPLEX, Sept 10, 2019. https://www.ibm.com/support/knowledgecenter/en/SSSA5P_12.9.0/ilog.odms.cplex.help/CPLEX/UsrMan/topics/discr_optim/benders/annotations.html.

[14] Niven-Jenkins B, D Brungard, M Betts, N Sprecher, and S Ueno. Requirements of an mpls transport profile, 2009.

[15] Md Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba. On orchestrating virtual network functions. In *2015 11th International Conference on Network and Service Management (CNSM)*, pages 50–56. IEEE, 2015.

[16] C. Barnhart, A. M. Cohn, E. L. Johnson, D. Klabjan, G. L. Nemhauser, and P. H. Vance. Airline Crew Scheduling. In *Handbook of Transportation Science*, volume 56 of *International Series in Operations Research & Management Science*, pages 517–560. , 2003.

[17] C. Barnhart, C. A. Hane, and P. H. Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Oper. Res.*, 48(2):318–326, 2000.

[18] Arsany Basta, Wolfgang Kellerer, Marco Hoffmann, Hans Jochen Morper, and Klaus Hoffmann. Applying NFV and SDN to LTE mobile core gateways, the functions placement problem. In *Proceedings of the 4th workshop on All things cellular: operations, applications, & challenges*, pages 33–38, 2014.

[19] JF Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4:238–252, 1962.

[20] JF Benders. *Partitioning in mathematical programming.* PhD thesis, Utrecht university, July 4, 1960.

[21] Pankaj Berde, Matteo Gerola, Jonathan Hart, Yuta Higuchi, Masayoshi Kobayashi, Toshio Koide, Bob Lantz, Brian O'Connor, Pavlin Radoslavov, William Snow, et al. ONOS: towards an open, distributed SDN OS. In *Proceedings of the third workshop on Hot topics in software defined networking*, pages 1–6, 2014.

[22] Deval Bhamare, Raj Jain, Mohammed Samaka, and Aiman Erbad. A survey on service function chaining. *Journal of Network and Computer Applications*, 75:138–155, 2016.

[23] Deval Bhamare, Mohammed Samaka, Aiman Erbad, Raj Jain, Lav Gupta, and H Anthony Chan. Optimal virtual network function placement in multi-cloud service function chaining architecture. *Computer Communications*, 102:1–16, 2017.

[24] Ole Bilde and Jakob Krarup. Sharp lower bounds and efficient algorithms for the simple plant location problem. In *Annals of Discrete Mathematics*, volume 1, pages 79–97. Elsevier, 1977.

[25] Natashia Boland, John Dethridge, and Irina Dumitrescu. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34(1):58–68, 2006.

[26] Pierre Bonami, Domenico Salvagnin, and Andrea Tramontani. Implementing Automatic Benders Decomposition in a Modern MIP Solver. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 78–90. Springer, 2020.

[27] Mathieu Bouet, Jérémie Leguay, Théo Combe, and Vania Conan. Cost-based placement of vDPI functions in NFV infrastructures. *International Journal of Network Management*, 25(6):490–506, 2015.

[28] Zheng Cai, Alan L Cox, and TS Ng. Maestro: A system for scalable openflow control. Technical report, , 2010.

[29] Martin Casado, Teemu Koponen, Rajiv Ramanathan, and Scott Shenker. Virtualizing the network forwarding plane. In *Proceedings of the Workshop on Programmable Routers for Extensible Services of Tomorrow*, pages 1–6. ACM, 2010.

[30] Yulun Cheng and Longxiang Yang. VNF deployment and routing for nfv-enabled multicast: A steiner tree-based approach. In *2017 9th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–4. IEEE, 2017.

[31] Margaret Chiosi, Don Clarke, Peter Cablelabs, Chris Donley, Lane Centurylink, Michael Bugenhagen, James Feger, Waqar Khan, Chunfeng China, Hui Cui, Clark Deng, Lei Baohua, Sun Zhenqiang, and Steven Wright. Network Functions Virtualisation (NFV) Network Operator Perspectives on Industry Progress. Technical report, ETSI, Tech, 2013.

[32] Margaret Chiosi, Don Clarke, Peter Willis, Andy Reid, James Feger, Michael Bugenhagen, Waqar Khan, Michael Fargano, Chunfeng Cui, Hui Deng, et al. Network functions virtualisation: An introduction, benefits, enablers, challenges and call for action. In *SDN and OpenFlow world congress*, volume 48, pages 1–16. Darmstadt-Germany, 2012.

[33] Divya Chitimalla, Koteswararao Kondepu, Luca Valcarenghi, Massimo Tornatore, and Biswanath Mukherjee. 5g fronthaul-latency and jitter studies of cpri over ethernet. *IEEE/OSA Journal of Optical Communications and Networking*, 9(2):172–182, 2017.

[34] Sunil Chopra, Bartosz Filipecki, Kangbok Lee, Minseok Ryu, Sangho Shim, and Mathieu Van Vyve. An extended formulation of the convex recoloring problem on a tree. *Mathematical Programming*, 165(2):529–548, 2017.

[35] NM Mosharaf Kabir Chowdhury and Raouf Boutaba. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[36] Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. Near optimal placement of virtual network functions. In *Computer Communications (INFOCOM), 2015 IEEE Conference on*, pages 1346–1354. IEEE, 2015.

[37] Jean-François Cordeau, Fabio Furini, and Ivana Ljubić. Benders decomposition for very large scale partial set covering and maximal covering location problems. *European Journal of Operational Research*, 275(3):882–896, 2019.

[38] Jean-François Cordeau, Fabio Furini, and Ivana Ljubić. Benders decomposition for very large scale partial set covering and maximal covering location problems. *European Journal of Operational Research*, 275(3):882 – 896, 2019.

[39] Roberto Doriguzzi Corin, Sandra Scott-Hayward, Domenico Siracusa, Marco Savi, and Elio Salvadori. Dynamic and application-aware provisioning of chained virtual security network functions. *CoRR*, abs/1901.01704, 2019.

[40] Gérard Cornuéjols, George Nemhauser, and Laurence Wolsey. The uncapicitated facility location problem. Technical report, Cornell University Operations Research and Industrial Engineering, 1983.

[41] Alysson M Costa. A survey on Benders decomposition applied to fixed-charge network design problems. *Computers & operations research*, 32(6):1429–1450, 2005.

[42] George B Dantzig and Philip Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[43] Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

[44] Edsger W Dijkstra et al. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[45] Moshe Dror. Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, 42(5):977–978, 1994.

[46] Abhishek Dwaraki and Tilman Wolf. Adaptive service-chain routing for virtual network functions in software-defined networks. In *Proceedings of the 2016 workshop on Hot topics in Middleboxes and Network Function Virtualization*, pages 32–37, 2016.

[47] David Erickson. The beacon openflow controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 13–18, 2013.

[48] Nick Feamster, Jennifer Rexford, and Ellen Zegura. The road to SDN: an intellectual history of programmable networks. *ACM SIGCOMM Computer Communication Review*, 44(2):87–98, 2014.

[49] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1. Technical report, RFC 2616, june, 1999.

[50] Matteo Fischetti, Ivana Ljubić, and Markus Sinnl. Benders decomposition without separability: A computational study for capacitated facility location problems. *European Journal of Operational Research*, 253(3):557–569, 2016.

[51] Matteo Fischetti, Ivana Ljubić, and Markus Sinnl. Benders decomposition without separability: A computational study for capacitated facility location problems. *European Journal of Operational Research*, 253(3):557–569, 2016.

[52] Matteo Fischetti, Ivana Ljubić, and Markus Sinnl. Redesigning Benders decomposition for large-scale facility location. *Management Science*, 63(7):2146–2162, 2017.

[53] Matteo Fischetti, Ivana Ljubić, and Markus Sinnl. Redesigning Benders decomposition for large-scale facility location. *Management Science*, 63(7):2146–2162, 2017.

[54] Matteo Fischetti, Domenico Salvagnin, and Arrigo Zanette. A note on the selection of Benders' cuts. *Math. Program.*, 124(1-2):175–182, 2010.

[55] Lester Randolph Ford Jr and Delbert R Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

[56] Open Networking Fundation. Software-defined networking: The new norm for networks. *ONF White Paper*, 2:2–6, 2012.

[57] Fabio Furini, Enrico Malaguti, Rosa Medina Durán, Alfredo Persiani, and Paolo Toth. A column generation heuristic for the two-dimensional two-staged guillotine cutting stock problem with multiple stock size. *European Journal of Operational Research*, 218(1):251–260, 2012.

[58] Jokin Garay, Jon Matias, Juanjo Unzilla, and Eduardo Jacob. Service description in the NFV revolution: Trends, challenges and a way forward. *IEEE Communications Magazine*, 54(3):68–74, 2016.

[59] Arthur M Geoffrion. Generalized benders decomposition. *Journal of optimization theory and applications*, 10(4):237–260, 1972.

[60] Matthieu Gérard, François Clautiaux, and Ruslan Sadykov. Column generation based approaches for a tour scheduling problem with a multi-skill heterogeneous workforce. *European Journal of Operational Research*, 252(3):1019–1030, 2016.

[61] Milad Ghaznavi, Nashid Shahriar, Reaz Ahmed, and Raouf Boutaba. Service function chaining simplified. *arXiv preprint arXiv:1601.00751*, 2016.

[62] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

[63] Paul C Gilmore and Ralph E Gomory. A linear programming approach to the cutting stock problem—part ii. *Operations research*, 11(6):863–888, 1963.

[64] Racha Gouareb, Vasilis Friderikos, and A Hamid Aghvami. Delay sensitive virtual network function placement and routing. In *2018 25th International Conference on Telecommunications (ICT)*, pages 394–398. IEEE, 2018.

[65] Albert Greenberg, James Hamilton, David A Maltz, and Parveen Patel. The cost of a cloud: research problems in data center networks. *ACM SIGCOMM computer communication review*, 39(1):68–73, 2008.

[66] R Guerzoni et al. Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action, introductory white paper. In *SDN and OpenFlow World Congress*, volume 1, pages 5–7, 2012.

[67] Abhishek Gupta, M Farhan Habib, Pulak Chowdhury, Massimo Tornatore, and Biswanath Mukherjee. Joint virtual network function placement and routing of traffic in operator networks. *UC Davis, Davis, CA, USA, Tech. Rep*, 2015.

[68] Bo Han, Vijay Gopalakrishnan, Lusheng Ji, and Seungjoon Lee. Network function virtualization: Challenges and opportunities for innovations. *IEEE Communications Magazine*, 53(2):90–97, 2015.

[69] Hajar Hantouti, Nabil Benamar, and Tarik Taleb. Service Function Chaining in 5G and Beyond Networks: Challenges and Open Research Issues. *IEEE Network*, 2020.

[70] Hajar Hantouti, Nabil Benamar, Tarik Taleb, and Abdelquoddous Laghrissi. Traffic steering for service function chaining. *IEEE Communications Surveys & Tutorials*, 21(1):487–507, 2018.

[71] Juliver Gil Herrera and Juan Felipe Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016.

[72] Ali Hmaity, Marco Savi, Leila Askari, Francesco Musumeci, Massimo Tornatore, and Achille Pattavina. Latency-and capacity-aware placement of chained virtual network functions in fmc metro networks. *Optical Switching and Networking*, 35:100536, 2020.

[73] Huawei Releases SDN/NFV Commercial and Technological Innovations, 2017. https://www.huawei.com/en/press-events/news/2017/10/Huawei-SDN-NFV-Commercial-Technological-Innovations.

[74] Nicolas Huin. *Energy efficient software defined networks*. PhD thesis, 2017.

[75] Nicolas Huin, Brigitte Jaumard, and Frédéric Giroire. Optimal network service chain provisioning. *IEEE/ACM Transactions on Networking*, 26(3):1320–1333, 2018.

[76] Nicolas Huin, Andrea Tomassilli, Frédéric Giroire, and Brigitte Jaumard. Energy-efficient service function chain provisioning. *Journal of Optical Communications and Networking*, 10(3):114–124, 2018.

[77] Sushant Jain, Alok Kumar, Subhasree Mandal, Joon Ong, Leon Poutievski, Arjun Singh, Subbaiah Venkata, Jim Wanderer, Junlan Zhou, Min Zhu, et al. B4: Experience with a globally-deployed software defined wan. *ACM SIGCOMM Computer Communication Review*, 43(4):3–14, 2013.

[78] Donald B Johnson. A note on dijkstra's shortest path algorithm. *Journal of the ACM (JACM)*, 20(3):385–388, 1973.

[79] Cédric Joncour, Sophie Michel, Ruslan Sadykov, Dmitry Sverdlov, and François Vanderbeck. Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics*, 36:695–702, 2010.

[80] Sherry Justine and Ratnasamy Sylvia. A survey of enterprise middlebox deployments. Technical report, Citeseer, 2012.

[81] Narumi Kiji, Takehiro Sato, Ryoichi Shinkuma, and Eiji Oki. Virtual network function placement and routing model for multicast service chaining based on merging multiple service paths. In *2019 IEEE 20th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–6. IEEE, 2019.

[82] Narumi Kiji, Takehiro Sato, Ryoichi Shinkuma, and Eiji Oki. Virtual network function placement and routing for multicast service chaining using merged paths. *Optical Switching and Networking*, 36:100554, 2020.

[83] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, et al. Onix: A distributed control platform for large-scale production networks. In *OSDI*, volume 10, pages 1–6, 2010.

[84] Sofie Lambert, Margot Deruyck, Ward Van Heddeghem, Bart Lannoo, Wout Joseph, Didier Colle, Mario Pickavet, and Piet Demeester. Post-peak ict: Graceful degradation for communication networks in an energy constrained future. *IEEE Communications Magazine*, 53(11):166–174, 2015.

[85] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, 16(1):493–512, 2013.

[86] Line MP Larsen, Aleksandra Checko, and Henrik L Christiansen. A survey of the functional splits proposed for 5G mobile crosshaul networks. *IEEE Communications Surveys & Tutorials*, 21(1):146–172, 2018.

[87] Xin Li and Chen Qian. The virtual network function placement problem. In *2015 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 69–70. IEEE, 2015.

[88] Yang Li, Linh Thi Xuan Phan, and Boon Thau Loo. Network functions virtualization with soft real-time guarantees. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[89] Yong Li and Min Chen. Software-defined network function virtualization: A survey. *IEEE Access*, 3:2542–2553, 2015.

[90] Tachun Lin, Zhili Zhou, Massimo Tornatore, and Biswanath Mukherjee. Optimal network function virtualization realizing end-to-end requests. In *2015 IEEE Global Communications Conference (GLOBECOM)*, pages 1–6. IEEE, 2015.

[91] M. E. Lübbecke and J. Desrosiers. Selected Topics in Column Generation. *Operations Research*, 53(6):1007–1023, 2005.

[92] Marco E Lübbecke and Jacques Desrosiers. Selected topics in column generation. *Operations research*, 53(6):1007–1023, 2005.

[93] Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Salete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. Piecing together the nfv provisioning puzzle: Efficient placement and chaining of virtual network functions. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 98–106. IEEE, 2015.

[94] Youcef Magnouche. *The multi-terminal vertex separator problem : Complexity, Polyhedra and Algorithms*. PhD thesis, Université Paris Dauphine, 2017.

[95] Ali Ridha Mahjoub. Polyhedral approaches. *Concepts of Combinatorial Optimization*, pages 261–324, 2014.

[96] Toqeer Mahmood, Tabassam Nawaz, Rehan Ashraf, and Syed M Adnan Shah. Gossip based routing protocol design for ad hoc networks. *International Journal of Computer Science Issues (IJCSI)*, 9(1):177, 2012.

[97] Atefeh Maleki, Md Hossain, Jean-Philippe Georges, Eric Rondeau, and Thierry Divoux. An SDN Perspective to Mitigate the Energy Consumption of Core Networks–GÉANT2. In *International Seeds Conference, Leeds*, 2017.

[98] Antonio Manzalini, Roberto Saracco, Cagatay Buyukkoc, Prosper Chemouil, Slawomir Kuklinski, Andreas Gladisch, Masaki Fukui, E Dekel, D Soldani, M Ulema, et al. Software-defined networks for future networks and services. In *White Paper based on the IEEE Workshop SDN4FNS*, 2013.

[99] SDN and NFV transforming the network:where do we go from here?, 2017. https://www.orange-business.com/en/blogs/connecting-technology/networks/sdn-and-nfv-transforming-the-network-where-do-we-go-from-here.

[100] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[101] Ahmed M Medhat, Tarik Taleb, Asma Elmangoush, Giuseppe A Carella, Stefan Covaci, and Thomas Magedanz. Service function chaining in next generation networks: State of the art and research challenges. *IEEE Communications Magazine*, 55(2):216–223, 2016.

[102] Jan Medved, Robert Varga, Anton Tkacik, and Ken Gray. Opendaylight: Towards a model-driven SDN controller architecture. In *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014*, pages 1–6. IEEE, 2014.

[103] Sevil Mehraghdam, Matthias Keller, and Holger Karl. Specifying and placing chains of virtual network functions. In *Cloud Networking (CloudNet), 2014 IEEE 3rd International Conference on*, pages 7–13. IEEE, 2014.

[104] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Niels Bouten, Filip De Turck, and Steven Davy. Design and evaluation of algorithms for mapping and scheduling of virtual network functions. In *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*, pages 1–9. IEEE, 2015.

[105] Hendrik Moens and Filip De Turck. VNF-P: A model for efficient placement of virtualized network functions. In *Network and Service Management (CNSM), 2014 10th International Conference on*, pages 418–423. IEEE, 2014.

[106] Tareq Monawar, Shafayat Bin Mahmud, and Avijit Hira. Anti-theft vehicle tracking and regaining system with automatic police notifying using haversine formula. In *2017 4th International conference on Advances in Electrical Engineering (ICAEE)*, pages 775–779. IEEE, 2017.

[107] Gábor Nagy and Saïd Salhi. Location-routing: Issues, models and methods. *European journal of operational research*, 177(2):649–672, 2007.

[108] Network Functions Virtualisation NFV and Use Cases. Etsi gs nfv 001 v1. 1.1 (2013-10). , 2013.

[109] Vikrant Nikam, James Gross, and Ahmad Rostami. Vnf service chaining in optical data center networks. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7. IEEE, 2017.

[110] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[111] Ben Pfaff, Justin Pettit, Keith Amidon, Martin Casado, Teemu Koponen, and Scott Shenker. Extending networking into the virtualization layer. In *Hotnets*, 2009.

[112] Kevin Phemius, Mathieu Bouet, and Jérémie Leguay. Disco: Distributed multi-domain SDN controllers. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–4. IEEE, 2014.

[113] Paul Quinn and Tom Nadeau. Problem statement for service function chaining. In *RFC 7498*. RFC Editor, 2015.

[114] Sridhar KN Rao. SDN and its use-cases-NV and NFV. *Network*, 2:H6, 2014.

[115] Giovanni Righini and Matteo Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, 51(3):155–170, 2008.

[116] Palash Roy, Anika Tahsin, Sujan Sarker, Tamal Adhikary, Md Abdur Razzaque, and Mohammad Mehedi Hassan. User mobility and Quality-of-Experience aware placement of Virtual Network Functions in 5G. *Computer Communications*, 150:367–377, 2020.

[117] Ruslan Sadykov and François Vanderbeck. Column generation for extended formulations. *Electronic Notes in Discrete Mathematics*, 37:357–362, 2011.

[118] Gamal Sallam, Gagan R Gupta, Bin Li, and Bo Ji. Shortest path and maximum flow problems under service function chaining constraints. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2132–2140. IEEE, 2018.

[119] Yu Sang, Bo Ji, Gagan R Gupta, Xiaojiang Du, and Lin Ye. Provably efficient algorithms for joint placement and allocation of virtual network functions. In *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2017.

[120] M. Savelsbergh. A Branch-And-Price Algorithm for the Generalized Assignment Problem. *Operations Research*, 45(6):pp. 831–841, 1997.

[121] Marco Savi, Massimo Tornatore, and Giacomo Verticale. Impact of processing costs on service chain placement in network functions virtualization. In *2015 IEEE Conference on Network Function Virtualization and Software Defined Network (NFV-SDN)*, pages 191–197. IEEE, 2015.

[122] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[123] Sachin Sharma, Dimitri Staessens, Didier Colle, Mario Pickavet, and Piet Demeester. Openflow: Meeting carrier-grade recovery requirements. *Computer Communications*, 36(6):656–665, 2013.

[124] Justine Sherry, Shaddi Hasan, Colin Scott, Arvind Krishnamurthy, Sylvia Ratnasamy, and Vyas Sekar. Making middleboxes someone else's problem: network processing as a cloud service. *ACM SIGCOMM Computer Communication Review*, 42(4):13–24, 2012.

[125] SNDlib, Dec 10, 2019. http://sndlib.zib.de/.

[126] Update: AT&T's Stephens: More Than 40% of Network Functions Are Virtualized, 2017. https://www.sdxcentral.com/articles/news/atts-stephens-47-network-functions-virtualized/2017/07/.

[127] Farzad Tashtarian, Amir Varasteh, Ahmadreza Montazerolghaem, and Wolfgang Kellerer. Distributed VNF scaling in large-scale datacenters: An ADMM-based approach. In *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, pages 471–480. IEEE, 2017.

[128] Nippon Telegraph. Telephone Corporation,"Ryu Network Operating System,", 2012.

[129] Andrea Tomassilli. *Towards Next Generation Networks with SDN and NFV*. PhD thesis, UCL-Université Côte d'Azur, 2019.

[130] Andrea Tomassilli, Nicolas Huin, Frédéric Giroire, and Brigitte Jaumard. Energy-efficient service chains with network function virtualization. 2016.

[131] Andrea Tomassilli, Nicolas Huin, Frederic Giroire, and Brigitte Jaumard. Resource requirements for reliable service function chaining. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2018.

[132] Richard M Van Slyke and Roger Wets. L-shaped linear programs with applications to optimal control and stochastic programming. *SIAM Journal on Applied Mathematics*, 17(4):638–663, 1969.

[133] François Vanderbeck. *Decomposition and column generation for integer programs.* PhD thesis, UCL-Université Catholique de Louvain, 1994.

[134] François Vanderbeck. Branching in Branch-and-Price: a generic scheme. *Mathematical Programming*, 130(2):249–294, 2011.

[135] François Vanderbeck and Laurence A Wolsey. Reformulation and decomposition of integer programs. In *50 Years of Integer Programming 1958-2008*, pages 431–502. Springer, 2010.

[136] Ming Xia, Meral Shirazipour, Ying Zhang, Howard Green, and Attila Takacs. Network function placement for NFV chaining in packet/optical datacenters. *Journal of Lightwave Technology*, 33(8):1565–1570, 2015.

[137] Yicheng Xu, Vincent Chau, Chenchen Wu, Yong Zhang, and Yifei Zou. Online joint placement and allocation of virtual network functions with heterogeneous servers. *arXiv preprint arXiv:2001.02349*, 2020.

[138] Jin Y Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, 1971.

[139] Jiao Zhang, Zenan Wang, Ningning Ma, Tao Huang, and Yunjie Liu. Enabling efficient service function chaining by integrating NFV and SDN: architecture, challenges and opportunities. *IEEE Network*, 32(6):152–159, 2018.

[140] Sai Qian Zhang, Ali Tizghadam, Byungchul Park, Hadi Bannazadeh, and Alberto Leon-Garcia. Joint NFV placement and routing for multicast service on SDN. In *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium*, pages 333–341. IEEE, 2016.

RÉSUMÉ

La virtualisation des fonctions réseau et les réseaux définis par logiciel sont deux nouvelles technologies prometteuses qui émergent dans la nouvelle génération des réseaux de télécommunication. Leur introduction permet la minimisation du temps de traitement des services, et un gain en énergie et en coûts. Dans cette thèse, nous étudions un problème confronté par les fournisseurs des services réseau, intitulé le problème de placement des fonctions virtuelles et de routage des services dans les réseaux définis par logiciel.

Nous commençons par prouver que le problème est NP-Difficile au sens fort, même en considérant une seule commodité, et en relâchant les contraintes de capacité sur les nœuds et sur les fonctions virtuelles, et aussi les contraintes de latence et de précédence. Nous proposons ensuite une formulation PLNE (Programmation linéaire en nombres entiers) compacte pour modéliser le problème. Cette formulation ne semble pas être assez forte pour trouver des solutions réalisables en un temps raisonnable à l'aide d'un solveur standard. Afin de remédier à cela, nous fournissons, une heuristique basée sur une formulation PLNE.

Nous proposons également, deux formulations PLNE étendues pour modéliser le problème et nous définissons l'algorithme de Branch-and-Pricer associé. En plus, nous définissons plusieurs familles d'inégalités valides afin de renforcer la relaxation linéaire. Nous comparons les deux algorithmes proposés et nous discutons leur performance.

Ensuite, nous étudions une variante du problème et nous présentons des résultats théoriques qui nous permettent de la reformuler en appliquant la décomposition de Benders. Nous renforçons cette reformulation par un ensemble d'inégalités valides. Tout cela est combiné dans algorithme de Branch-and-Benders-cut que nous avons testé et comparé avec l'algorithme de Benders automatique inclus dans le solveur commercial Cplex.

Les résultats numériques indiquent que nos approches de décomposition et de reformulation sont plus efficaces par rapport aux deux méthodes (la formulation compacte et l'algorithme de Benders automatique) fournies par un solveur standard sur un ensemble d'instances réalistes, à la fois en terme de temps CPU et en terme de qualité de la solution. Les résultats indiquent également que notre heuristique fournit des solutions de haute qualité.

MOTS CLÉS

Optimisation combinatoire, Génération de colonnes, Branch-and-Price, inégalités valides, décomposition de Benders, Branch-and-Benders-Cut, Heuristique, Fonctions virtualisées de réseau, Réseaux définis par logiciel, Chaînage de fonctions de service.

ABSTRACT

Network Functions Virtualization (NFV) and Software Defined Networking (SDN) are two promising techniques for the next generation of telecommunication networks. Their introduction allows time, energy, and cost minimization. In this dissertation, we study a problem faced by network service providers, named, the Virtual Network Functions Placement and Routing problem (VNFPRP) in Software Defined Networks.

We start proving that the VNFPRP is NP-hard in a strong sense, even for a single commodity and without node-capacity, VNFs-capacity, latency, and precedence constraints. We provide a compact Mixed Integer Linear Programming (MILP) formulation to model it. This formulation does not seem to be strong enough for finding a solution using an off-the-shelf solver. In order to tackle the problem from a computational perspective, we provide MILP-based heuristic. The obtained results indicate that our MILP-heuristic provides high-quality solutions.

Moreover, we propose two extended formulations for the problem and derive a Branch-and-Price algorithm. We also provide several families of valid inequalities to strengthen the LP-bounds. We present computational results and discuss the performance of each algorithm.

Furthermore, we discuss a variant of the problem, and we provide theoretical results that allow us to derive Benders reformulation of the problem, along with several families of valid inequalities. These ingredients are combined in a Branch-and-Benders-Cut framework and computationally tested on a wide range of realistic instances and compared with the Automatic Benders decomposition provided by Cplex.

Computational results indicate that our decomposition and reformulation approaches are more efficient compared to the two methods (the MILP compact formulation and the automatic Benders) provided by the off-the-shelf solver on a set of realistic instances, both in terms of CPU time and overall solution quality.

KEYWORDS

Combinatorial optimization, Column generation, Branch-and-Price, Valid inequalities, Benders decomposition, Branch-and-Benders-cut, Heuristic, Virtual Network Functions, Software-Defined Networking, Service Functions Chaining.