UNIVERSITÀ DEGLI STUDI DI MILANO

Mathematical Programming Algorithms for Network Optimization Problems

Dipartimento di Matematica ”Federico Enriques”

Dottorato in Matematica e Statistica per le Scienze Computazionali - XXVI Ciclo

PhD Thesis of: Fabio Colombo

Tutor: Prof. Marco Trubian

Co-Tutor: Ing. Roberto Cordone

Coordinator: Prof. Giovanni Naldi

Contents

Chapter 1
Introduction

1.1 Network Optimization Problems

The term *network* may denote many different concepts, depending on the considered area of study: for instance, in the telecommunications sector it denotes a set of interconnected electronic devices that communicate with each other (Bertsekas and Gallager, 1987), in the railway industry it indicates how railroad yards are connected by rail tracks (Ahuja et al., 2005), and in social sciences it refers to the way in which different human beings interact (Wellman and Berkowitz, 1988). What all these definitions have in common is that all of them describe collections of simple items interconnected to each other to form a complex structure. Throughout the rest of this thesis, we will adopt the connectivity among simple components generating a more complex system as the characterizing property of a network.

Networks provide a general framework to describe both natural phenomena and human technologies. In the real world, there is an incredible large number of complex systems that can be better understood if we are able to identify one or more networks interconnecting the simple parts of the considered system. Starting by enumerating the smallest parts of a system and, then, analyzing how these parts are related to each other, is a powerful method to dominate its complexity.

The practical applications of networks range from physical networks that provide transportation, water, power and communication, to logical networks that allow us to simply model many decisions processes like, for example, the way in which a pool of different tasks requiring the same resources are scheduled on a machine (Herroelen et al., 1998), or the way in which the different components of an IT system interact to satisfy the user requirements (Harrold et al., 1992). Other applications arise also in biology and natural sciences (see, for example, De Jong 2002; Dunne et al. 2002; Proulx et al. 2005).

In this thesis we consider combinatorial optimization problems (Lawler, 1976) defined by means of networks, these problems arise when we need to take effective decisions to build or manage complex interconnected systems, both satisfying the design constraints and minimizing the costs that we need to pay (or maximizing the profit that we can obtain).

The main mathematical tools developed in the literature to model networks are graphs, their variants and their extensions. As a result, the problems that we analyze in this thesis can be easily and effectively reformulated as combinatorial optimization problems that require to identify special structures in one or more given graphs.

Two of these special structures are particularly recurring throughout all the following chapters: trees and paths. This is not at all surprising. A tree is a minimal set of edges in an undirected graph that guarantees the connectivity of the spanned nodes and, as a consequence, in applications it represents the cheapest way to join each item of the considered network to each other. A path, on the other side, is both a particular type of tree and a basic component of each tree. While a tree allows us to cheaply satisfy a many-to-many connectivity requirement, a path allows us to cheaply satisfy a one-to-one connectivity requirement.

In the remaining part of this thesis we focus our attention on the four following problems:

- The *Multicast Routing and Wavelength Assignment with Delay Constraint in WDM networks with heterogeneous capabilities* (MRWADC) problem: this problem (Chen et al. 2008; Colombo and Trubian 2013) arises in the telecommunications industry and it requires to define an efficient way to make multicast transmissions on a WDM optical network (Mukherjee, 2006). In more formal terms, to solve the MRWADC problem we need to identify, in a given directed graph that models the considered WDM optical network, a set of arborescences that connect the source of the transmission to all its destinations. These arborescences need to satisfy several quality-of-service constraints, taking into account the heterogeneity of the electronic devices which constitute the WDM network.
- The *Homogeneous Area Problem* (HAP): this problem (Colombo et al., 2011, 2012; Ceselli et al., 2013; Colombo et al., 2013) arises from a particular requirement of an intermediate level of the Italian government called *province*. Each province needs to coordinate the common activities of the towns that belong to its territory. To practically perform its coordination role, the province of Milan created a "customer care" layer composed by a certain number of employees that have the task to support the towns of the province in their administrative works. For the sake of efficiency, the employees of this "customer care" layer have been partitioned in small groups and each group is assigned to a particular subset of towns that have in common a large number of activities. The HAP requires to identify the set of towns assigned to each group in order to minimize the redundancies generated by the towns that, despite having some activities in common, have been assigned to different groups. Since, for both historical and practical reasons, the towns in a particular subset need to be adjacent, the HAP can be effectively modeled as a particular Graph Partitioning Problem (GPP, see Fjallström 1998) that requires the connectivity of the obtained subgraphs and the satisfaction of nonlinear knapsack constraints.
- *Knapsack Prize Collecting Steiner Tree Problem* (KPCSTP): we starting considering this problem when we need to develop the two Column Generation methods

(Lübbecke and Desrosiers, 2005) for the MRWADC problem and for the HAP. In both cases the Pricing Problem requires to find an arborescence that minimizes the difference between its cost and the prizes associated with the spanned nodes. The two problems differ in the side constraints that their feasible solutions need to satisfy and in the way in which the cost of an arborescence is defined. The ILP formulations and the resolution methods that we developed to tackle these two Pricing Problems share many characteristics with the ones used to solve other similar problems. To exemplify these similarities, we considered KPCSTP as a prototype for all these problems: it requires to find a tree that minimizes the difference between the cost of the used arcs and the prize of the spanned nodes. However, not all trees are feasible: each node is associated with a nonnegative weight and the sum of the weights of the nodes spanned by a feasible tree cannot exceed a given threshold. In the thesis we propose several Integer Linear Programming (ILP) formulations for the KPCSTP and compare the resulting optimization methods with an other method proposed in the literature (Cordone and Trubian, 2008).

- The *Train Design Optimization* (TDO) problem: this problem was the topic of the second problem solving competition, sponsored in 2011 by the Railway Application Section (RAS) of the Institute for Operations Research and the Management Sciences (INFORMS)[1]. The TDO problem arises in the freight railroad industry: a freight railroad company receives requests from customers to transport a set of railcars from an origin rail yard to a destination rail yard. To satisfy these requests, the company first aggregates the railcars having the same origin and the same destination in larger blocks, and then it defines a *trip plan* to transport the obtained blocks to their correct destinations. The TDO problem requires to identify a trip plan that efficiently uses the limited resources of the considered rail company. More formally, given a railway network, a set of blocks and the segments of the network in which a crew can legally drive a train, the TDO problem requires to define a set of trains and the way in which the given blocks can be transported to their destinations by these trains, both satisfying operational constraints and minimizing the transportation costs. We participated to the contest and we won the second prize. After the competition, we continued to work on the TDO problem and in this thesis we describe the improved method that we have finally obtained.

All the combinatorial optimization problems we consider in this thesis arise from real world applications. As a consequence, they take in account many side constraints and practical aspects that contribute to increase their complexity. To deal with this complexity, we exploited a large part of the tools and techniques developed in the field of Mathematical Programming and Operations Research, starting from exact methods (see Section 2.1.1) that, solving the consider problems to optimality, cannot scale to high dimensions but allow us to deeply investigate the structure of the problem, passing through heuristic methods (see Section 2.1.2) that are able to

[1] For further details, see the official website `http://www.informs.org/Community/RAS/Problem-Solving-Competition/`.

find good feasible solutions in a reasonable amount of time, without giving guarantees on their quality, arriving to hybrid methods (see Section 2.1.3) that either exploit the information gained by partially executing exact methods to effectively drive the heuristics or speed up exact methods by heuristically solving some related subproblems.

Among these tools and techniques, the Dantzig-Wolfe decomposition principle (Dantzig and Wolfe, 1960) and the Column Generation method are the ones we employed in a more extensive way: the Dantzig-Wolfe decomposition allowed us to apply the *divide et impera* principle to the resolution process of complex ILP formulations, and applying the Column Generation method we were able to implicitly solve their Linear Programming (LP) Relaxation. In Chapter 2 we propose a short introduction to these two techniques and we describe which changes are required in the Column Generation method in order to simultaneously generate both columns and rows (we applied this modified method to efficiently solve the TDO problem, see Chapter 6).

1.2 Original Contributions

Our contributions can be divided in three different parts: modeling of new problems, methodological advances and practical results.

From the modeling point of view, our first contributions lie in the definition and analysis of several formulations for two combinatorial optimization problems requiring the identification of a set of constrained arborescences in given directed graphs. In particular, for the MRWADC problem we propose a new extended formulation obtained applying the Dantzig-Wolfe decomposition to a well-known compact formulation. To model the HAP, we first discuss its distinctive aspects which make it different from the standard GPP; then we propose two compact formulations and we discuss how to solve some symmetry related issues that afflict both of them. One way to solve these issues is to define a new extended formulation obtained applying the Dantzig-Wolfe decomposition to one of the previous compact formulations. To actually solve the extended formulations for the MRWADC problem and for the HAP, we need to solve also the two corresponding Pricing Problems which can be modeled as constrained prize collecting arborescence problems. The final modeling contribution regards the TDO problem which we model using an extended formulation based on an exponential number of variables and constraints. To avoid the complete enumeration of all these variables and constraints, we need to define and solve two different Pricing Problems: one to generate new train routes and the other to generate the paths followed by the blocks, denoted in the following by block-paths.

A first methodological contribution lies in the definition of two effective Column Generation methods, one for the MRWADC problem and one for the HAP. The extended formulations we propose for these two problems are both associated with a $\mathscr{N}\mathscr{P}$-hard Pricing Problem. Thus, to terminate the Column Generation method in a reasonable amount of time, our algorithms generate new columns by combining exact and heuristic methods. Indeed, we developed two Tabu Search (Glover and Laguna, 1997) heuristics that take in account the different structures of the two considered Pricing Problems. We use them until they are able to find negative reduced cost columns and, when they fail, we fall back to exact methods based on ILP formulations. One of the two ILP formulations that we propose to tackle the Pricing Problem associated with the MRWADC extended formulation contains two different exponential families of constraints. The first family is based on the well-known connectivity cuts for the *Prize Collecting Steiner Tree Problem* (PCSTP, Ljubić et al. 2006) and, to dynamically generate them, we apply the standard separation algorithm based on the minimum cut problem. Instead, to dynamically generate the constraints in the second family, we propose and implement a new separation algorithm based on the minimum cost network flow problem (Ahuja et al., 1993). Moreover, starting from the ILP formulations developed for the Pricing Problems associated with the HAP and with the MRWADC problem, we derive similar ILP formulations to solve also the KPCSTP. Another methodological contribution consists in the definition of different Column Generation based heuristics for the HAP. In particular, we considered three different strategies to implement them. The first two exploit a partial exploration of a particular branching tree (Joncour et al., 2010; Cacchiani et al., 2012) and differ in the way in which they generate and process new branching nodes. The last method is based on the Large Neighborhood Search (Pisinger and Røpke, 2010), and its repair phase is based on the Column Generation method.

The last methodological contribution resides in the definition of a Simultaneous Row and Column Generation based heuristic for the TDO. This type of algorithms has only recently been considered in the literature (Zak, 2002; Feillet et al., 2010; Muter et al., 2012), and ours is one of the first Simultaneous Row and Column Generation based methods that deal with all the complex practical aspects of a combinatorial optimization problem arising in an industrial setting. As by product of the definition of this algorithm, we developed fast heuristics to solve the two Pricing Problems respectively associated with the generation of new train routes and new block-paths.

From the practical point of view, this thesis describes several effective methods to solve complex optimization problems arising both in industrial settings and in the public sector. In particular, applying the methods that we developed for the HAP, we were able to obtain the optimal solution for the medium-size instance describing the Monza province and we found near optimal solutions for the bigger instance describing the Milan province. Moreover, applying the heuristic method that we developed for the TDO, we obtained good quality solutions for two real world instances provided by a first class US railroad company.

Overall, the work described in this thesis required the original implementation, tuning and testing of two Branch & Cut algorithms, two Column Generation methods, three Column Generation based heuristics, one Simultaneous Row and Column Generation based heuristic and three Tabu Search heuristics.

1.3 Thesis outline

In Chapter 2, we introduce the well-known Dantzig-Wolfe decomposition principle and the Column Generation method that are two techniques widely used in the following chapters. In the end, we describe how to change the Column Generation method to dynamically generate, not only the columns of an extended formulation, but also its rows.

In Chapter 3, we study different methods to solve the KPCSTP. We describe how to adapt the methods already proposed in the literature for the PCSTP to consider the knapsack constraint and we report the results of the computational experiments that we performed in order to investigate the impact of the added constraint on the different proposed methods.

In Chapter 4, we describe the Column Generation method that we developed for the MRWADC problem. We start by introducing a new compact formulation for the problem and the extended formulation that we obtained applying to it the Dantzig-Wolfe decomposition. We describe the Column Generation method that we use to obtain the LP-Relaxation of the extended formulation and the different algorithms that we developed to efficiently solve the associated Pricing Problem. The final section reports a computational comparison between the solutions obtained by solving the compact formulation and the ones obtained by adding the integrality constraint to the last generated Reduced Master Problem.

In Chapter 5, we describe the different methods that we developed to solve the HAP. The chapter begins with a short survey of the techniques proposed in the literature to solve similar GPPs and provides different arguments showing that the HAP cannot be solved by simply modifying them. Then, we introduce and compare two different compact ILP formulations for the HAP, noting that both of them have scalability issues. The chapter continues by presenting a new extended formulation that overcomes these issues and by introducing the Column Generation method that we used to obtain its LP-Relaxation and the algorithms that we developed to solve the associated Pricing Problem. The chapter ends with the description and the computational comparison among two local search heuristics and three different hybrid heuristics that make use of the Column Generation method as a subroutine.

In Chapter 6, we describe the Simultaneous Column and Row Generation based heuristic that we developed to solve the TDO Problem. Since this problem is quite complex, the first part of the chapter is devoted to the description of the several operational constraints and different objective function components that contribute to the definition of the problem. The chapter continues with a short survey of the three other methods that participated to the final round of the second RAS problem solving competition. After this survey, we introduce the extended ILP formulation on which our heuristic is based. This formulation presents both an exponential number of columns and an exponential number of rows. Then we describe the Simultaneous Column Row Generation heuristic that we employed to obtain good feasible solutions and the algorithms that we developed to solve the two associated Pricing Problems. The final section of the chapter provides a computational comparison among our method and the other methods proposed to solve the TDO problem during the competition.

1.4 References

R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Prentice-Hall, Inc., 1993.

R. K. Ahuja, C. B. Cunha, and G. Şahin. Network Models in Railroad Planning and Scheduling. In J. Cole Smith, editor, *Tutorials in Operations Research: Emerging Theory, Methods, and Applications*, chapter 3, pages 54–101. INFORMS, 2005.

D. P. Bertsekas and R. Gallager. *Data networks*. Prentice Hall, 1987.

V. Cacchiani, V. C. Hemmelmayr, and F. Tricoire. A set-covering based heuristic algorithm for the periodic vehicle routing problem. *Discrete Applied Mathematics*, 2012. doi: 10.1016/j.dam.2012.08.032.

A. Ceselli, F. Colombo, R. Cordone, and M. Trubian. Employee workload balancing by graph partitioning. *Discrete Applied Mathematics*, 2013. doi: 10.1016/j.dam.2013.02.014.

M. T. Chen, B. M. T. Lin, and S. S. Tseng. Multicast routing and wavelength assignment with delay constraints in WDM networks with heterogeneous capabilities. *Journal of Network and Computer Applications*, 31(1):47–65, 2008.

F. Colombo and M. Trubian. A column generation approach for multicast routing and wavelength assignment with delay constraints in heterogeneous wdm networks. *Annals of Operations Research*, 2013. doi: 10.1007/s10479-013-1403-7.

F. Colombo, R. Cordone, and M. Trubian. On the partition of an administrative region in homogenous districts. In *Atti del Convegno AIRO 2011*, Brescia, Italy, 2011.

F. Colombo, R. Cordone, and M. Trubian. Upper and lower bounds for the homogenous areas problem. In *Proceedings of the 11th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, München, Germany, 2012.

F. Colombo, R. Cordone, and M. Trubian. Column-generation based bound for the homogeneous areas problem, 2013. Submitted to European Journal of Operational Research, under second review round.

R. Cordone and M. Trubian. A Relax-and-Cut Algorithm for the Knapsack Node Weighted Steiner Tree Problem. *Asia-Pacific Journal of Operational Research*, 25(3):373–391, 2008.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8(1):101–111, 1960.

H. De Jong. Modeling and simulation of genetic regulatory systems: a literature review. *Journal of computational biology*, 9(1):67–103, 2002.

J. A. Dunne, R. J. Williams, and N. D Martinez. Network structure and biodiversity loss in food webs: robustness increases with connectance. *Ecology Letters*, 5(4): 558–567, 2002.

D. Feillet, M. Gendreau, A. L. Medaglia, and J. L. Walteros. A note on branch-and-cut-and-price. *Operations Research Letters*, 38(5):346 – 353, 2010.

P. O. Fjallström. Algorithms for graph partitioning: A survey. *Linköping Electronic Articles in Computer and Information Science*, 10, 1998.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

M. J. Harrold, J. D. McGregor, and K. J. Fitzpatrick. Incremental testing of object-oriented class structures. In *Proceedings of the 14th international conference on Software engineering*, ICSE '92, pages 68–80, New York, NY, USA, 1992. ACM.

W. Herroelen, B. De Reyck, and E. Demeulemeester. Resource-constrained project scheduling: A survey of recent developments. *Computers & Operations Research*, 25(4):279 – 302, 1998.

C. Joncour, S. Michel, R. Sadykov, D. Sverdlov, and F. Vanderbeck. Column generation based primal heuristics. *Electronic Notes in Discrete Mathematics*, 36:695 – 702, 2010.

E. Lawler. *Combinatorial Optimizations: Networks and Matroids*. Holt, Rinehar and Winston, 1976.

I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem. *Mathematical programming*, 105(2-3):427–449, 2006.

M. E. Lübbecke and J. Desrosiers. Selected Topics in Column Generation. *Operations Research*, 53(6):1007–1023, 2005.

B. Mukherjee. *Optical WDM Networks*. Springer, 2006.

İ. Muter, Ş. İ. Birbil, and K. Bülbül. Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows. *Mathematical Programming*, 2012. doi: 10.1007/s10107-012-0561-8.

D. Pisinger and S. Røpke. Large neighborhood search. *Handbook of metaheuristics*, pages 399–419, 2010.

S. R. Proulx, D. E. L. Promislow, and P. C. Phillips. Network thinking in ecology and evolution. *Trends in Ecology & Evolution*, 20(6):345 – 353, 2005.

B. Wellman and S. D. Berkowitz. *Social structures: A network approach*, volume 2. Cambridge University Press, 1988.

E. J. Zak. Row and column generation technique for a multistage cutting stock problem. *Computers & Operations Research*, 29(9):1143 – 1156, 2002.

Chapter 2
The Column Generation method and its extensions

2.1 Exact methods and Heuristics: how to combine them?

It is well known that a large amount of the optimization problems arising both in industry and in academia belongs to the class of the $\mathscr{NP}$-hard problems (Garey and Johnson, 1979). As a consequence, unless $\mathscr{NP} = \mathscr{P}$, we cannot expect to develop fast polynomial time algorithms to solve them. Nonetheless, the practical need to efficiently solve these hard problems has given rise to two main lines of investigation in the Operations Research community: some efforts have been aimed to develop *exact methods* that are able to find optimal solutions in a more efficient way w.r.t. the simple enumerative approaches. On the other side, since exact methods generally require too much computational resources when the dimension of the considered instances increases too much, the focus of other investigations has been the development of fast *heuristics* that are able to find good solutions in a reasonable amount of time. An often underestimated result of the investigation on exact methods is the increased knowledge of the problem structure and of the peculiar characteristics of its optimal solutions: to systemically improve the heuristic methods with the information gained through the execution of exact methods, in the last decades *hybrid methods* that combine both exact and heuristic methods have been developed. In the remaining part of this section we propose a short survey of these three broad classes of optimization methods.

2.1.1 Exact methods

The final aim of each exact method is to identify an optimal solution for the considered problem. Exact methods are what we need when we are facing strategic problems whose instances have modest size: the time required to find an optimal solution is measured in hours or days and we have enough time to wait and enough interest to know the optimal solution. Though the required time is not a tight constraint in a typical use case of exact methods, the fewer computational efforts required by *ad hoc* methods when related to naive enumerative algorithms are crucial to increase the size of instances that can be practically solved. To minimize their computational resources requirements, good exact methods need to take into account the particular structure of the considered problem. Nonetheless, during the last decades, in order to simplify the development of new exact methods some general algorithms design techniques have been proposed in the literature. When the considered optimization problem can be easily defined by means of a Mathematical Programming formulation, Branch & Bound (see Lawler and Wood 1966 for a survey) is probably the most frequently used exact method.

The most important decisions to make, in order to effectively implement a Branch & Bound algorithm are the following ones:

- Choose the order in which the algorithm visits the nodes of the branching tree.
- Choose the way in which the feasible solutions associated with a given node are split among its children nodes.
- Choose the procedure to compute the lower bound associated with a given node.

In the previous literature, several papers has been focused on the last point: a good bounding procedure must be fast and must be able to produce strong lower bounds. If the obtained lower bounds are near to the cost of the optimal solution, the chances to prune great portions of the branching tree increase significantly. Often the bounding procedures consist in methods that effectively compute the LP-Relaxation of an ILP formulation for the considered instance. Generally, tighter lower bounds can be obtained starting from larger ILP formulations. Some of these formulations have a number of variables or constraints that is not bound by a polynomial in the dimension of the considered problem. Thus, we cannot solve their LP-Relaxations applying a standard LP method. To obtain the LP-Relaxations of these formulations without enumerating all their constraints and variables, different methods have been developed in the literature. If the number of constraints is too high we can use Cutting Planes methods (introduced for the first time in Dantzig et al. 1954) and the obtained Branch & Bound algorithm belongs to the Branch & Cut family. Otherwise, if the number of columns is too high we can apply the Column Generation method (see Section 2.2.4) and the obtained Branch & Bound algorithm belongs to the Branch & Price family. Combination of both techniques can be used as well and depending on the type of constraints we want to dynamically add to our LP-Relaxation formulation, we can apply Branch & Cut & Price methods (Fukasawa et al., 2006) or Simultaneous Column and Row Generation approaches (see Section 2.4).

2.1.2 Heuristics

Heuristics are optimization methods that partially explore the solution space in order to find good solutions in a reasonable amount of time. In general, the best solution found by a heuristic is not the optimal one, and the heuristic itself does not provide any information about the quality of the found solution. Nonetheless, in the last decades many efficient heuristic methods have been developed by researchers both in academia and in industry in order to effectively solve optimization problems. Computational experiments have shown that well implemented heuristics are able to find near optimal solutions for several optimization problems without requiring too much computational resources. The analysis of the heuristics proposed and developed in the past years has brought to the identification of many recurring patterns in the structure of similar efficient heuristics.These patterns have been generalized in order to define frameworks, called metaheuristics, that allow to easily developed efficient heuristics. Examples of metaheuristics are Tabu Search (Glover, 1986), Ant Colony Optimization (Colorni et al., 1991), Very Large Scale Neighboorhood Search (Ahuja et al., 2002) and many others (see Gendreau and Potvin 2010 for a survey of the most important ones).

2.1.3 Hybrid Methods

In the last years, in the Operations Research community, we observed a growing interest in the effective methods that can be obtained by hybridizing exact methods with heuristics (Maniezzo et al., 2010). These hybrid methods can be divided in two broad classes:

- *Exact methods enhanced by heuristics*: this class contains exact methods that exploit heuristics to efficiently solve some of the subproblems arising during their execution. Examples of these methods are the primal heuristics embedded in many commercial ILP solvers (see, for example, Danna et al. 2005), the fast heuristics used to iteratively solve the separation problems arising in the Cutting Planes methods (see, for example, Lysgaard et al. 2004) or the pricing heuristics developed to solve the hard Pricing Problems arising in some implementations of the Branch & Price framework (see, for example, Bettinelli et al. 2011).
- *Heuristics enhanced by exact methods*: this class contains heuristics that employ exact methods to efficiently solve subproblems arising in some of their crucial steps like the exploration of large neighborhood (see Ahuja et al., 2002), the recombination of different solutions (see Rothberg, 2007), the repair of a partially destroyed solution (see Parragh and Schmid, 2013).

Recently has been noted (Boschetti and Maniezzo, 2009; Boschetti et al., 2010) that the large set of decomposition techniques developed in the previous literature to exactly solve large scale Mathematical Programming formulations can be exploited in the definition of effective strategies that combine heuristics and exact methods.

The most important representatives of these techniques are Lagrangian Relaxation (Geoffrion, 1974), Benders Decomposition (Benders, 1962) and Dantzig-Wolfe decomposition and their characterizing property is the ability to reduce the resolution of a large and intractable problem to the iterative resolution of smaller and more tractable problems. In general, the subproblems in which these techniques decompose the original problems are linked together by hierarchical relations and the resulting structure can be exploited to design effective hybrid metaheuristics in which only some of the subproblems are solved exactly, leaving the remaining ones to *ad hoc* heuristics. In this thesis, when we need to decompose a large scale problem, we focus our attention on the Dantzig-Wolfe decomposition technique. As a consequence, after a general introduction to the Dantzig-Wolfe decomposition in the following section, in Section 2.3 we survey the methods proposed in the literature to turn exact methods based on this technique into hybrid heuristics.

2.2 Dantzig-Wolfe decomposition and Column Generation method

As discussed in Section 2.1.1, a key element of each Branch & Bound method based on a LP-Relaxation procedure is the identification of a good ILP formulation that correctly describe the considered problem. The two most important criteria to evaluate the quality of an ILP formulation are the tightness of its LP-Relaxation and the amount of time required to compute it.

In their seminal work (Dantzig and Wolfe, 1960), G. B. Dantzig and P. Wolfe described a general procedure, now called Dantzig-Wolfe decomposition, that allows us to transform a compact LP formulation in an extended one. The obtained formulation is not weaker (as a matter of fact, it is often much stronger) than the original one. However, in general, the number of its variables is exponential in the dimension of the problem and we cannot easily solve it with a standard LP method.

Two years before the publication of the work of Dantzig and Wolfe, L. Ford and D. Fulkerson proposed a method to implicitly enumerate the variables of an extended formulation for a multi-commodity flow problem (Ford and Fulkerson, 1958). In 1961 this method has been generalized and extended to solve the Cutting Stock Problem by P. Gilmore and R. Gomory (Gilmore and Gomory, 1961). The method described in the latter paper has been developed and refined in the last decades and it is now well known as the Column Generation method (Lübbecke and Desrosiers, 2005). In the remaining part of this section, we describe both the Dantzig-Wolfe decomposition and the Column Generation method in more details.

2.2.1 The Dantzig-Wolfe decomposition

The Dantzig-Wolfe decomposition has been introduced to efficiently solve LP formulations that have the following block diagonal structure:

$$\begin{aligned}
&\min c_1^\top x_1 + c_2^\top x_2 + \ldots + c_m^\top x_m && \text{(2.1a)}\\
&A_1 x_1 + A_2 x_1 + \ldots + A_m x_m = b && \text{(2.1b)}\\
&D_1 x_1 \qquad\qquad\qquad = u_1 && \text{(2.1c)}\\
&\qquad D_2 x_2 \qquad\qquad = u_2 && \text{(2.1d)}\\
&\qquad\qquad \ddots \qquad = \vdots && \text{(2.1e)}\\
&\qquad\qquad\quad D_m x_m = u_m && \text{(2.1f)}\\
&x^\top = (x_1^\top, x_2^\top, \ldots, x_m^\top) \geq 0 && \text{(2.1g)}
\end{aligned}$$

The feasible region associated with this formulation can be stated as the intersection of n polyhedra defined as $P_k = \left\{ x_k \in \mathbb{R}_+^{L_k} : D_k x_k = u_k \right\}$ for each $k = 1, \ldots, m$, with constraints (2.1b). Note that here we use L_k to denote the dimension of vector x_k and $L = \sum_k L_k$ to denote the total number of variables (i.e. the dimension of vector x). Each P_k acts only on a portion (denoted by x_k) of the decision variables and, since constraints (2.1b) are the only ones that link together all the variables, they are called *linking constraints*.

Since the Minkowski-Weyl theorem (see, for example, Schrijver, 1998) states that each point of a polyhedron can be generated by a convex combination of its extreme points plus a conic combination of its extreme rays, each P_k can be defined as follows:

$$P_k = \left\{ x_k = \sum_{i=1}^{N_k} \lambda_i v_i^k + \sum_{i=1}^{M_k} \delta_i r_i^k \mid \lambda_i, \delta_i \geq 0, \sum_{i=1}^{N_k} \lambda_i = 1 \right\}$$

Where the sets $\left\{ v_1^k, \ldots, v_{N_k}^k \right\}$ and $\left\{ r_1^k, \ldots, r_{M_k}^k \right\}$ contain respectively all the P_k extreme points and all the P_k extreme rays.

Since in the following we consider only bounded polyhedra (i.e. polytopes), we assume that the set of extreme rays is empty and that the polytope P_k can be simply defined as the set of convex combinations of its extreme points:

$$P_k = \left\{ x_k = \sum_{i=1}^{N_k} \lambda_i v_i^k \mid \lambda_i \geq 0, \sum_{i=1}^{N_k} \lambda_i = 1 \right\}$$

The key idea behind the Dantzig-Wolfe decomposition is the introduction of the previous extended definition of P_k into formulation (2.1), in order to implicitly satisfy constraints (2.1c-2.1f) and to make decisions, not directly on x_k, but on the values of the convex combination coefficients:

$$\min c_1^\top \sum_{i=1}^{N_1} \lambda_i^1 v_i^1 + c_2^\top \sum_{i=1}^{N_2} \lambda_i^2 v_i^2 + \ldots + c_m^\top \sum_{i=1}^{N_m} \lambda_i^m v_i^m \tag{2.2a}$$

$$A_1 \sum_{i=1}^{N_1} \lambda_i^1 v_i^1 + A_2 \sum_{i=1}^{N_2} \lambda_i^2 v_i^2 + \ldots + A_m \sum_{i=1}^{N_m} \lambda_i^m v_i^m = b \tag{2.2b}$$

$$\sum_{i=1}^{N_1} \lambda_i^1 = 1 \tag{2.2c}$$

$$\sum_{i=1}^{N_2} \lambda_i^2 = 1 \tag{2.2d}$$

$$\ddots \qquad \vdots \tag{2.2e}$$

$$\sum_{i=1}^{N_m} \lambda_i^m = 1 \tag{2.2f}$$

$$\lambda_i^k \geq 0 \text{ for each } k = 1, \ldots, m \text{ and } i = 1, \ldots, N_k \tag{2.2g}$$

The number of extreme points in each polytope P_k is generally exponential on the number of constraints and variables of its compact description $\left\{ x_k \in \mathbb{R}_+^{L_k} : D_k x_k \geq 0 \right\}$. As a consequence, while the number of constraints in the extended formulation (2.2) is less than the number of constraints in the compact one (2.1), the number of variables in the extended formulation is huge and it is not possible to solve it by applying standard LP methods.

2.2.2 Dantzig-Wolfe decomposition and ILP formulation

Despite the Dantzig-Wolfe decomposition has been initially developed to improve the tractability of large scale LP formulations, in the last two decades it has been adapted and extensively exploited to obtain tighter LP-Relaxations for several ILP formulations (Vanderbeck, 1994; Barnhart et al., 1998). To show how the same decomposition principle can be also applied to ILP formulations, we start by considering the compact ILP formulation obtained by adding integrality constraint to the compact LP formulation (2.1):

$$\min c_1^\top x_1 + c_2^\top x_2 + \ldots + c_m^\top x_m \tag{2.3a}$$
$$A_1x_1 + A_2x_1 + \ldots + A_mx_m = b \tag{2.3b}$$
$$D_1x_1 \qquad\qquad = u_1 \tag{2.3c}$$
$$D_2x_2 \qquad = u_2 \tag{2.3d}$$
$$\ddots \quad = \vdots \tag{2.3e}$$
$$D_mx_m = u_m \tag{2.3f}$$
$$x^\top = (x_1^\top, x_2^\top, \ldots, x_m^\top) \in \mathbb{N}^L \tag{2.3g}$$

The classical approach to solve this formulation requires to remove its integrality constraint, obtaining a LP formulation that, using the polytopes P_k defined in the previous section ,can be compactly defined as follows:

$$\min c_1^\top x_1 + c_2^\top x_2 + \ldots + c_m^\top x_m \tag{2.4a}$$
$$A_1x_1 + A_2x_1 + \ldots + A_mx_m = b \tag{2.4b}$$
$$x_k \in P_k \text{ for each } k = 1, \ldots, m \tag{2.4c}$$

Then, this bounding technique can be embedded into a Branch & Bound method (or in a more sophisticated Branch & Cut method) to force the satisfaction of the integrality constraint. However, as already said in Section 2.1.1, the success of these implicit enumeration methods is strictly linked to the strength of the underlying bounding techniques. A stronger lower bound for formulation (2.3) can be achieved by separately considering the following integer polytopes, derived introducing the integrality constraint into P_k, for each $k = 1, \ldots, m$:

$$\overline{P}_k = P_k \cap \mathbb{N}^{L_k} = \left\{ D_kx_k = u_k, x_k \in \mathbb{N}_k^L \right\}$$

If we replace each polytope P_k with the convex hull of $\overline{P}_k$ we obtain the following LP formulation:

$$\min c_1^\top x_1 + c_2^\top x_2 + \ldots + c_m^\top x_m \tag{2.5a}$$
$$A_1x_1 + A_2x_1 + \ldots + A_mx_m = b \tag{2.5b}$$
$$x_k \in conv(\overline{P}_k) \text{ for each } k = 1, \ldots, m \tag{2.5c}$$

This last LP formulation is not weaker than formulation (2.4) since, by definition, $conv(\overline{P}_k) \subseteq P_k$, for each $k = 1, \ldots, m$. Moreover, if exists $k \in \{1, \ldots, m\}$ such that P_k does not have the integrality property (i.e. $conv(\overline{P_k}) \subset P_k$), the lower bound obtained from formulation (2.5) is greater than the lower bound obtained from formulation (2.4).

To practically solve formulation (2.5) we need to provide an implicit way to enforce constraints (2.5c) since, in general, there is not a compact description of the convex hull of a discrete set. To obtain this result, similarly to what we have done in Section 2.2.1, applying the Minkowsky-Weyl theorem to polyhedron $conv(\overline{P}_k)$, we can describe it only by means of its extreme points $\left\{\overline{v}_1^k, \ldots, \overline{v}_{\overline{N}_k}^k\right\}$ as follows:

$$conv(\overline{P}_k) = \left\{ x_k = \sum_{i=1}^{\overline{N}_k} \lambda_i \overline{v}_i^k \mid \lambda_i \geq 0, \sum_{i=1}^{\overline{N}_k} \lambda_i = 1 \right\} \tag{2.6}$$

Notice that, similarly to what done in the previous application of the Minkowsky-Weyl theorem, we do not consider the set of extreme rays of $conv(\overline{P}_k)$ since we assume that P_k is a polytope.

Now, applying a procedure similar to the one described in Section 2.2.1 that transforms the convex combination coefficients used in definition (2.6) in decision variables, we obtain a LP formulation equivalent to formulation (2.5):

$$\min c_1^\top \sum_{i=1}^{\overline{N}_1} \lambda_i^1 \overline{v}_i^1 + c_2^\top \sum_{i=1}^{\overline{N}_2} \lambda_i^2 \overline{v}_i^2 + \ldots + c_m^\top \sum_{i=1}^{\overline{N}_m} \lambda_i^m \overline{v}_i^m \tag{2.7a}$$

$$A_1 \sum_{i=1}^{\overline{N}_1} \lambda_i^1 \overline{v}_i^1 + A_2 \sum_{i=1}^{\overline{N}_2} \lambda_i^2 \overline{v}_i^2 + \ldots + A_m \sum_{i=1}^{\overline{N}_m} \lambda_i^m \overline{v}_i^m = b \tag{2.7b}$$

$$\sum_{i=1}^{\overline{N}_1} \lambda_i^1 = 1 \tag{2.7c}$$

$$\sum_{i=1}^{\overline{N}_2} \lambda_i^2 = 1 \tag{2.7d}$$

$$\ddots \qquad \vdots \tag{2.7e}$$

$$\sum_{i=1}^{\overline{N}_m} \lambda_i^m = 1 \tag{2.7f}$$

$$\lambda_i^k \geq 0 \text{ for each } k = 1, \ldots, m \text{ and } i = 1, \ldots, \overline{N}_k \tag{2.7g}$$

This formulation is similar to the one obtained at the end of the previous section. It has a large amount of variables and a limited number of constraints. As a consequence, we cannot solve it using a standard LP method.

Note that in Section 4.3 we apply this decomposition technique to the compact ILP formulation for the MRWADC problem described in Section 4.2. By solving the obtained extended formulation, we are able to effectively tackle bigger instances w.r.t. the ones that can be solved using the compact formulation (see Section 4.6.4 for the computational results).

2.2.3 Identical Subsystems and symmetry breaking by Dantzig-Wolfe decomposition

A particular case of the Dantzig-Wolfe decomposition arises when we need to apply it to a problem having a diagonal structure, in which all the subsystems constituting the blocks of the diagonal are equivalent. If we make the following substitutions into formulation (2.3) for each $k = 1, \dots, m$:

$$c_k = c, \quad A_k = A, \quad D_k = D, \quad L^k = L', \quad u_k = u$$

We can simply state the resulting formulation as follows:

$$\min\ c^\top \sum_{k=1}^{m} x_k \tag{2.8a}$$

$$\sum_{k=1}^{m} A x_k = b \tag{2.8b}$$

$$D x_k = u \quad k = 1, \dots, m \tag{2.8c}$$

$$x^\top = (x_1^\top, x_2^\top, \dots, x_m^\top) \in \mathbb{N}^L \tag{2.8d}$$

If we want to directly solve this formulation using a standard Branch & Bound algorithm we need to take into account the large number of symmetries contained in the set of its feasible solutions: given a feasible solution $\tilde{x} = (\tilde{x}_1, \dots, \tilde{x}_m)$ we can produce an equivalent feasible solution by only permuting the order of its subcomponents $\{\tilde{x}_k\}_k$. Exploiting this idea, starting from each feasible solution we can easily obtain $m!$ equivalent feasible solutions. If we do not take care of these symmetries, a standard Branch & Bound algorithm may waste a large amount of time to analyze solutions that are equivalent to other already analyzed and pruned solutions (Margot, 2010).

On the contrary, if we carefully apply the Dantzig-Wolfe decomposition to formulation (2.8) we can get rid of all these symmetry issues. To obtain this result, we start by considering the convexification of $conv(\overline{P}_k)$:

$$conv(\overline{P}_k) = conv(\{Dx_k = u, x_k \in \mathbb{N}^{L_k}\}) = \left\{ x = \sum_{i=1}^{\overline{N}} \lambda_i^k \overline{v}_i^k \mid \sum_{i=1}^{\overline{N}} \lambda_i^k = 1, \lambda_i^k \in \mathbb{N} \right\}$$

Note that, since $\overline{P}_k$ is equivalent to $\overline{P} = \left\{ Dx = u, x \in \mathbb{N}^{L'} \right\}$ for each $k = 1, \dots, m$, the set of $\overline{P}_k$ extreme points is the same for each $k = 1, \dots, m$. We denote the common set of extreme points as $\{\overline{v}_1, \dots, \overline{v}_N\}$ and we exploit this equivalence by introducing the following aggregating variables:

$$\rho_i = \sum_{k=1}^{m} \lambda_i^k \quad \text{for each } i = 1, \dots, N$$

Using these new variables we can define the following extended LP formulation associated with the previous ILP formulation (2.8):

$$\min \; c^\top \sum_{i=1}^{\overline{N}} \rho_i \overline{v}_i \tag{2.9a}$$

$$A \sum_{i=1}^{\overline{N}} \rho_i \overline{v}_i = b \tag{2.9b}$$

$$\sum_{i=1}^{\overline{N}} \rho_i = m \tag{2.9c}$$

$$\rho_i \geq 0 \text{ for each } i = 1, \dots, \overline{N} \tag{2.9d}$$

Each equivalent feasible solution that can be obtained by permuting the order of indexes $k = 1, \dots, m$ of a particular feasible solution of the LP-Relaxation of formulation (2.8) is associated with the same solution of formulation (2.9), As a consequence, we have solved the symmetry issues existing in the previous formulation (2.8).

Note that this type of decomposition has been used to derive an extended formulation for the HAP (see Section 5.7). The derived formulation solves several symmetry issues afflicting both the compact formulations that we initially propose to solve the HAP (see Section 5.6).

2.2.4 The Column Generation method

In the previous section we have seen that, by applying the Dantzig-Wolfe decomposition, we can obtain a tighter extended formulation starting from a compact formulation having a particular diagonal structure. However, the number of variables of these extended formulations is, in general, exponential in the dimension of the original problem and we cannot solve them with a standard LP method.

To overcome these issues and to solve the LP-Relaxation of these extended formulations, we can employ the Column Generation method to implicitly enumerate all their variables. To simplify the presentation of this algorithm, we start by considering the last extended formulation (2.9) derived from the compact formulation (2.8) defined by m identical subsystems. In Column Generation terminology, the extended formulation (2.9) is called *Master Problem* (MP) and Column Generation starts by generating an initial *Reduced Master Problem* (RMP) obtained from MP considering only a subset of its variables, these variables are indexed by $\Omega_1 \subset \Omega = \{1, 2, \dots, N\}$. At each iteration $t > 1$, we define a new subset Ω_t that extends Ω_{t-1} with some of the indexes belonging to $\Omega \setminus \Omega_{t-1}$, and obtaining, as a consequence, a new RMP:

$$\text{RMP}(\Omega_t) : \min \; c^\top \sum_{i \in \Omega_t} \rho_i \bar{v}_i \tag{2.10a}$$

$$A \sum_{i \in \Omega_t} \rho_i \bar{v}_i = b \tag{2.10b}$$

$$\sum_{i \in \Omega_t} \rho_i = m \tag{2.10c}$$

$$\rho_i \geq 0 \text{ for each } i \in \Omega_t \tag{2.10d}$$

There are only two requirements that we need to satisfy during the generation of subsets $\Omega_1 \subset \Omega_2 \subset \cdots \subset \Omega_{t-1} \subset \Omega_t \subset \cdots \subset \Omega$:

- The number of indexes in the initial subset Ω_1 and, at each iteration $t > 1$, the number of indexes added to Ω_{t-1} to obtain Ω_t must not be too large: we need to be able to solve each RMP(Ω_t) with a standard LP method.
- The initial set Ω_1 must contain enough indexes: we need to guarantee the existence of a feasible solution for each RMP(Ω_t).

Note that the first requirement can be easily satisfied by introducing one or more dummy columns associated with high costs, similarly to what done in the two phase simplex method (see, for example, Vanderbei 2008). If both these requirements are satisfied, we can straightforwardly solve RMP(Ω_t) using a standard LP method. When this method terminates, it returns both a primal optimal solution $\tilde{\rho}^{(t)}$ and a dual optimal solution $(\tilde{\gamma}^{(t)}, \tilde{\eta}^{(t)})$, where vector $\tilde{\gamma}^{(t)}$ contains the optimal values of the dual variables associated with the linking constraints (2.10b) and $\tilde{\eta}^{(t)}$ is the optimal value of the dual variable associated with the convexity constraint (2.10c).

We can extend $\tilde{\rho}^{(t)}$ to derive a primal feasible solution $\tilde{\rho}$ for the full MP by simply setting $\tilde{\rho}_i = \tilde{\rho}_i^{(t)}$ if $i \in \Omega_t$ and $\tilde{\rho}_i = 0$ if $i \in \Omega \setminus \Omega_t$. To construct a feasible dual solution for the full MP, let us consider the dual problem associated with it:

$$\text{DMP} : \max \quad \gamma^\top b + m\eta \tag{2.11a}$$

$$(A\bar{v}_i)^\top \gamma + \eta \leq c^\top \bar{v}_i \quad i \in \Omega \tag{2.11b}$$

$$\gamma, \eta \text{ free} \tag{2.11c}$$

We can directly derive a valid solution for DMP by setting $(\tilde{\gamma}, \tilde{\eta}) = (\tilde{\gamma}^{(t)}, \tilde{\eta}^{(t)})$. This solution satisfies constraints (2.11b) for each $i \in \Omega_t$ and the primal-dual pair $\{\tilde{\rho}, (\tilde{\gamma}, \tilde{\eta})\}$ trivially satisfies the complementary slackness condition. As a consequence, $\{\tilde{\rho}, (\tilde{\gamma}, \tilde{\eta})\}$ is optimal for the full MP if and only if constraints (2.11b) are satisfied also for each $i \in \Omega \setminus \Omega_t$. To implicitly check this last condition, without evaluating one by one the satisfaction of all constraints (2.11b), we need to solve the following optimization problem, also called *Pricing Problem* (PP) in the Column Generation terminology:

$$\text{PP} : z_{\text{PR}} = \min_{i \in \Omega} c^\top \bar{v}_i - (A\bar{v}_i)^\top \tilde{\gamma} - \tilde{\eta}$$

If we now exploit the definition of $\{\overline{v}_1,\ldots,\overline{v}_N\}$ as the set of extreme points of the integer polyhedron $\overline{P} = \left\{Dx = u, x \in \mathbb{N}^{L'}\right\}$, we can solve PP using the following ILP formulation:

$$\text{PP} : z_{\text{PP}} = \min c^\top x - (Ax)^\top \tilde{\gamma} - \tilde{\eta} \tag{2.12a}$$
$$Dx = u \tag{2.12b}$$
$$x \in \mathbb{N}^{L'} \tag{2.12c}$$

If the optimal objective function value z_{PP} is nonnegative, then all constraints (2.11b) are satisfied by $(\tilde{\gamma}, \tilde{\eta})$. Thus, $\{\tilde{\rho}, (\tilde{\gamma}, \tilde{\eta})\}$ is optimal also for the full MP and we can stop the Column Generation method. Otherwise, there exists at least one index $i \in \Omega \setminus \Omega_t$ such that the associated constraint (2.11b) is not satisfied, we can insert any index associated with these violated constraints into Ω_t in order to obtain Ω_{t+1} and then solve RMP(Ω_{t+1}), in an iteratively fashion.

Note that, as reported in Chapter 4 for the MRWADC problem and in Chapter 5 for the HAP, when the PP itself is hard, we can avoid to solve it exactly during the first Column Generation iterations. However, to have a guaranteed lower bound at the end of the generation phase we need to solve the PP exactly at least once.

2.3 Column Generation based metaheuristics

As already discussed in Section 2.1.3, during the last years, in the Operations Research community there has been a growing interest in hybrid methods that combine exact and heuristic methods. In this context, some efforts have been made to define effective hybridization frameworks based on the Dantzig-Wolfe decomposition and the Column Generation method.

Some investigations in this area have been mainly focused on the development of efficient primal heuristics that can be embedded in a Branch & Price algorithm. In (Joncour et al., 2010) the authors start with a short survey on the classical and straightforward Column Generation based heuristics developed in the previous literature, like the rounding heuristic and the diving heuristic. In the second part of the paper, the authors propose the insertion of a diversification strategy based on the Limited Discrepancy Search (Harvey and Ginsberg, 1995) into the diving heuristic. Some computational results obtained applying the proposed heuristics to four different optimization problems are reported at the end of the paper. In (Lübbecke and Puchert, 2011) the authors describe the three different primal heuristics that they developed for their generic Branch & Price solver `GCG` (Gamrath and Lübbecke, 2010) based on the `SCIP` framework (Achterberg, 2009). The first heuristic they propose is the Extreme Points Crossover heuristic, this heuristic first selects a set of elements that are in common to a large set of columns that belong to the current RMP solution, then it solves a subproblem in which all the feasible solutions are enforced to contain all these elements. The second heuristic they propose is the Restricted Master heuristic in which a subproblem is defined and solved restricting the set of considered columns to the most promising ones. The last proposed heuristic is the Column Selection heuristic in which a feasible solution is built by iteratively fixing the values of some already generated columns, trying to satisfy all the MP constraints. In (Pesneau et al., 2012), the authors describe how to embed the Feasibility Pump method (Fischetti et al., 2005) in a Branch & Price algorithm. This is extension is not straightforward: to avoid the introduction of limitations to the effectiveness of the Pricing Problem solver, it is important to carefully define the objective function of the subproblem solved in each pumping cycle.

Other works in this area exploit the hierarchical partition of complex problems in master and slave subproblems, introduced by the Dantzig-Wolfe decomposition, in order to define effective coordination mechanisms to guide hybrid heuristics. In (Boschetti et al., 2010) the authors propose different ways to turn classical decomposition techniques in hybrid heuristics, they consider the Lagrangian Relaxation, the Benders decomposition and the Dantzig-Wolfe decomposition. To validate the proposed strategies they apply them to solve the Single Source Capacitated Facility Location Problem and then report the obtained results. In (Prescott-Gagnon et al., 2009), the authors exploit the Column Generation method to define an effective reconstruction step for their Large Neighborhood Search algorithm (Pisinger and Røpke, 2010) for the Vehicle Routing Problem with Time Windows. In (Cacchiani et al., 2012) the authors developed a Column Generation based heuristic to solve the Periodic Vehicle Routing Problem in which the Column Generation method is a basic step of a local search guided by Tabu Search.

The combination of the Column Generation method with heuristics is a recurring theme throughout the remaining part of the thesis. In both Column Generation implementations reported in Chapter 4 and in Chapter 5, to solve, respectively, the MRWADC problem and the HAP, we were able to terminate the Column Generation method, in a reasonable amount of time, only after the introduction of an effective Tabu Search heuristic to solve the Pricing Problem: in both cases the Pricing Problem is $\mathscr{N}\mathscr{P}$-hard and we can fast solve them with a commercial ILP solver only if we consider trivial instances. Moreover, The LP-Relaxation of the extended formulation that we developed for the MRWADC problem is very tight and the heuristic solution obtained by simply adding the integrality constraint to the final RMP is often optimal. On the contrary, the LP-Relaxation of the extended formulation that we developed for the HAP is less tight, thus, to improve the quality of the obtained feasible solutions, we implemented and tested three more complex Column Generation based heuristics, respectively, on the methods proposed in (Joncour et al., 2010), (Cacchiani et al., 2012) and (Prescott-Gagnon et al., 2009).

2.4 Simultaneous Row and Column Generation

Some works in the recent literature (see Zak, 2002; Avella et al., 2006; Feillet et al., 2010; Muter et al., 2012b,a, 2013) have been mainly focused on large scale ILP formulations that are quite different from the ones which are obtained applying the Dantzig-Wolfe decomposition to the particular compact formulations described in the previous section.

These formulations do not only have a huge number of variables, but they have also a huge number of constraints. Moreover, when we apply problem generation based methods to solve these formulations, we need to take into account the fact that the large part of these constraints are redundant unless some of the variables on which they are defined have been already generated. As a consequence, to solve these formulations, we need to simultaneously generate both columns and rows in an efficient way. These formulations are called *Column Dependent Row* (CDR, see Muter et al. 2012a) formulations and their general form can be stated as follows:

$$\mathrm{MP}: \min \sum_{j\in\Omega^Y} c_j y_j + \sum_{i\in\Omega^W} d_i w_i \tag{2.13a}$$

$$\sum_{j\in\Omega^Y} a_{kj} y_j = \hat{a}_k \quad k\in\Lambda^Y \tag{2.13b}$$

$$\sum_{i\in\Omega^W} b_{pi} w_i = \hat{b}_p \quad p\in\Lambda^W \tag{2.13c}$$

$$\sum_{j\in\Omega^Y} r_{qj} y_j + \sum_{i\in\Omega^W} d_{qi} w_i = \hat{r}_q \quad q\in\Lambda^{YW} \tag{2.13d}$$

$$y_j, w_i \geq 0 \text{ for each } j\in\Omega^Y, i\in\Omega^W \tag{2.13e}$$

In this formulation we suppose that the number of constraints (2.13b,2.13c) (indexed, respectively, by Λ^Y and Λ^W) is polynomial in the size of the considered instance. However, the number of constraints (2.13d) (indexed by set Λ^{YW}) and the number of variables $\{y_j, w_i : j\in\Omega^Y, i\in\Omega^W\}$ may be exponential in the size of the considered instance.

In the following, we sketch an efficient algorithm to solve CDR formulations that takes inspiration from the Column Generation method described in the previous section. We also discuss how a naive adaption of the Column Generation method to solve CDR formulations may be incorrect and prematurely terminate, returning a suboptimal solution.

As a first step, we define two subsets, $\Omega_1^Y \subset \Omega^Y$ and $\Omega_1^W \subset \Omega^W$, that define the initial pool of variables considered in the first Reduced Master Problem. At each iteration of the algorithm, we enlarge these subsets generating two subset collections:

$$\Omega_1^Y \subset \Omega_2^Y \subset \cdots \subset \Omega_{t-1}^Y \subset \Omega_t^Y \subset \cdots \subset \Omega^Y$$

$$\Omega_1^W \subset \Omega_2^W \subset \cdots \subset \Omega_{t-1}^W \subset \Omega_t^W \subset \cdots \subset \Omega^W$$

In order to generate subsets $\{\Omega_{t+1}^Y, \Omega_{t+1}^W\}$ from $\{\Omega_t^Y, \Omega_t^W\}$ we first need to solve the following Reduced Master Problem defined on $\{\Omega_t^Y, \Omega_t^W\}$:

$$\text{RMP}(\Omega_t^Y, \Omega_t^W) : \min \sum_{j \in \Omega_t^Y} c_j y_j + \sum_{i \in \Omega_t^W} d_i w_i \tag{2.14a}$$

$$\sum_{j \in \Omega_t^Y} a_{kj} y_j \geq \hat{a}_k \quad k \in \Lambda^Y \tag{2.14b}$$

$$\sum_{i \in \Omega_t^W} b_{pi} w_i \geq \hat{b}_p \quad p \in \Lambda^W \tag{2.14c}$$

$$\sum_{j \in \Omega_t^Y} r_{qj} y_j + \sum_{i \in \Omega_t^W} d_{qi} w_i \geq \hat{r}_q \quad q \in \Lambda^{YW}(\Omega_t^Y, \Omega_t^W) \tag{2.14d}$$

$$y_j, w_i \geq 0 \text{ for each } j \in \Omega_t^Y, i \in \Omega_t^W \tag{2.14e}$$

In the previous formulation we use $\Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$ to denote constraints (2.13d) that are defined by some variables indexed by Ω_t^Y or by Ω_t^W Note that each CDR formulation must also satisfy the two following assumptions on the structure of RMP(Ω_t^Y, Ω_t^W) :

- *Variables dependence*: if we add to RMP(Ω_t^Y, Ω_t^W) a new subset of y variable indexes denoted by $\tilde{\Omega}^Y \subset \Omega^Y \setminus \Omega_t^Y$, we have to immediately add a new subset of w variable indexes denoted by $S(\tilde{\Omega}^Y) \subset \Omega^W \setminus \Omega_t^W$ and a new subset of constraint indexes denoted by $\Delta(\tilde{\Omega}^Y) \subset \Lambda^{YW} \setminus \Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$. A simple example in which this assumption is satisfied occurs when variables w are used to model the quadratic costs associated with variables y (see Muter et al., 2013): for each pair of indexes $(j_1, j_2) \in \Omega^Y \times \Omega^Y$ exists a variable w_i such that its index $i \in \Omega^W$ is uniquely associated with pair (j_1, j_2), and it must be equal to the product of variables y_{j_1} and y_{j_2}. In this case, if we add $\tilde{\Omega}^Y = \{j_1, j_2\}$ to RMP(Ω_t^Y, Ω_t^W) we have to immediately add to RMP(Ω_t^Y, Ω_t^W) also variable w_i and all the constraints that link y_{j_1} and y_{j_2} with it (for example, $w_i \geq y_{j_1} + y_{j_2} - 1$).
- *Linking constraints redundancy*: all the constraints (2.13d) that have not yet been generated (i.e. the ones indexed by $\Lambda^{YW} \setminus \Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$) are redundant in formulation (2.13) if we set to zero all the variables that have not yet been generated (i.e. the variables indexed by $\Omega^Y \setminus \Omega_t^Y$ or by $\Omega^W \setminus \Omega_t^W$).

If we carefully generate both sets Ω_t^Y and Ω_t^W without increasing too much their sizes and the size of $\Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$, we can easily solve RMP(Ω_t^Y, Ω_t^W) with a standard LP method and, at the end of its execution, we obtain an optimal primal solution $(\tilde{y}^{(t)}, \tilde{w}^{(t)})$ and an optimal dual solution $(\tilde{\gamma}^{(t)}, \tilde{\nu}^{(t)}, \tilde{\mu}^{(t)})$ where dual variables vectors $\gamma \geq 0$, $\nu \geq 0$ and $\mu \geq 0$ are respectively associated with primal constraints (2.13b), (2.13c) and (2.13d).

From the primal optimal solution $(\tilde{y}^{(t)}, \tilde{w}^{(t)})$ of RMP(Ω_t^Y, Ω_t^W) we can obtain a primal feasible solution $(\tilde{y}, \tilde{w})$ for the full MP, i.e. for formulation (2.13), by simply defining $\tilde{y}_j = \tilde{y}_j^{(t)}$ and $\tilde{w}_i = \tilde{w}_i^{(t)}$ for each $j \in \Omega_t^Y, i \in \Omega_t^W$ and $\tilde{y}_j = \tilde{w}_i = 0$ for each $j \in \Omega^Y \setminus \Omega_t^Y, i \in \Omega^W \setminus \Omega_t^W$. Note that $(\tilde{y}, \tilde{w})$ does not violate any constraints (2.13d) indexed by $\Lambda^{YW} \setminus \Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$ since formulation (2.13d) satisfies the *Linking constraints redundancy* assumption and it does not violate the already generated constraints (2.13d) indexed by $\Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$ since it derives from a feasible RMP(Ω_t^Y, Ω_t^W) solution.

To understand if it is possible to generate a dual feasible solution for formulation (2.13) starting from the dual optimal solution $(\tilde{\gamma}^{(t)}, \tilde{\nu}^{(t)}, \tilde{\mu}^{(t)})$ of RMP(Ω_t^Y, Ω_t^W), let us consider the dual problem associated with formulation (2.13):

$$\text{DMP} : \min \sum_{k\in\Lambda^Y} \hat{a}_k\gamma_k + \sum_{p\in\Lambda^W} \hat{b}_p\nu_p + \sum_{q\in\Lambda^{WY}} \hat{r}_q\mu_q \tag{2.15a}$$

$$\sum_{k\in\Lambda^Y} a_{kj}\gamma_k + \sum_{q\in\Lambda^{YW}} r_{qj}\mu_q \le c_j \quad j\in\Omega^Y \tag{2.15b}$$

$$\sum_{p\in\Lambda^W} b_{pi}\nu_p + \sum_{q\in\Lambda^{YW}} d_{qi}\mu_q \le d_i \quad i\in\Omega^W \tag{2.15c}$$

$$\gamma_k, \nu_p, \mu_q \ge 0 \text{ for each } k\in\Lambda^Y, p\in\Lambda^W, q\in\Lambda^{YW} \tag{2.15d}$$

Similarly to what done for the primal solution, we can extend the dual solution $(\tilde{\gamma}^{(t)}, \tilde{\nu}^{(t)}, \tilde{\mu}^{(t)})$ of RMP(Ω_t^Y, Ω_t^W) to obtain a valid DMP solution by setting $\gamma_k = \gamma_k^{(t)}$ for each $k \in \Lambda^Y$, $\nu_p = \nu_p^{(t)}$ for each $p \in \Lambda^W$ and $\mu_q = \mu_q^{(t)}$ for each $q \in \Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$. However, this partial solution cannot be correctly completed since RMP(Ω_t^Y, Ω_t^W) does not contain the constraints (2.13d) indexed by $\Lambda^{YW} \setminus \Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$. Consequently, without these constraints we cannot evaluate the correct value for each dual variable μ_q with $q \in \Lambda^{YW} \setminus \Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$.

A simple and fast way to overcome this issue is to heuristically set to zero all the unknown dual variables: we set $\mu_q = 0$ for each $q \in \Lambda^{YW} \setminus \Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$. After these fixings, we can define and solve the two following Pricing Problems:

$$\text{PP}^Y : Z_{\text{PP}}^Y = \min_{j\in\Omega^Y} c_j - \sum_{k\in\Lambda^Y} a_{kj}\gamma_k - \sum_{q\in\Lambda^{YW}(\Omega_t^Y)} r_{qj}\mu_q \tag{2.16}$$

$$\text{PP}^W : Z_{\text{PP}}^W = \min_{i\in\Omega^W} d_i - \sum_{p\in\Lambda^W} b_{pi}\nu_p - \sum_{q\in\Lambda^{YW}(\Omega_t^Y)} d_{qi}\mu_q \tag{2.17}$$

Differently from what done in the standard Column Generation method described in Section 2.2.4, if the optimal objective function values of both problems PP^Y and PP^W are nonnegative (i.e. $Z_{\text{PP}}^Y \ge 0$ and $Z_{\text{PP}}^W \ge 0$) we cannot interrupt the generation process and declare the optimality of the MP solution obtained extending $(\tilde{y}^{(t)}, \tilde{w}^{(t)})$: it is possible that the correct values of the dual variables fixed at zero decrease the reduced costs of some of the primal variables indexed by $\Omega^Y \setminus \Omega_t^Y$ or $\Omega^W \setminus \Omega_t^W$ that have not been yet generated. On the contrary, if, by solving PP^Y or PP^W, we find a negative reduced cost variable we can add it to (Ω_t^Y, Ω_t^W) to obtain $(\Omega_{t+1}^Y, \Omega_{t+1}^W)$ and solve RMP$(\Omega_{t+1}^Y, \Omega_{t+1}^W)$ in an iterative fashion. This last step works correctly like in a Column Generation method since, by fixing some nonnegative dual variables to zero, we are underestimating the correct reduced costs of the primal variables

To overcome the problems introduced by the wrong estimation process of the values of the dual variables indexed by $\Lambda^{YW} \setminus \Lambda^{YW}(\Omega_t^Y, \Omega_t^W)$, the authors of (Muter et al., 2012a) propose a rather sophisticated algorithm to correctly anticipate these values. The framework they propose cannot be effectively applied *as is* to complex and large scale problems, however their deep analysis of CDR problems properties give us useful insights in the development of the Simultaneous Column and Row Generation heuristic which we developed to solve the TDO problem and which we describe in Chapter 6: in this heuristic (see Section 6.5.3), we introduce an effective strategy to obtain good estimates of the values of the missing dual variables that is strongly based on some problem specific considerations.

2.5 References

T. Achterberg. Scip: solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, 2009.

R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75 – 102, 2002.

P. Avella, B. D'Auria, and S. Salerno. A lp-based heuristic for a time-constrained routing problem. *European Journal of Operational Research*, 173(1):120 – 124, 2006.

C. Barnhart, E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, and P. H. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, 46(3):316–329, 1998.

J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische Mathematik*, 4(1):238–252, 1962.

A. Bettinelli, A. Ceselli, and G. Righini. A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research Part C: Emerging Technologies*, 19(5):723 – 740, 2011.

M. Boschetti and V. Maniezzo. Benders decomposition, lagrangean relaxation and metaheuristic design. *Journal of Heuristics*, 15(3):283–312, 2009.

M. Boschetti, V. Maniezzo, and M. Roffilli. Decomposition techniques as metaheuristic frameworks. In V. Maniezzo, T. Stützle, and S. Voß, editors, *Matheuristics*, volume 10 of *Annals of Information Systems*, pages 135–158. Springer, 2010.

V. Cacchiani, V. C. Hemmelmayr, and F. Tricoire. A set-covering based heuristic algorithm for the periodic vehicle routing problem. *Discrete Applied Mathematics*, 2012. doi: 10.1016/j.dam.2012.08.032.

A. Colorni, M. Dorigo, and V. Maniezzo. Distributed optimization by ant colonies. In *European Conference on Artificial Life*, pages 134–142, 1991.

E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

G. B. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Operations Research*, pages 393–410, 1954.

D. Feillet, M. Gendreau, A. L. Medaglia, and J. L. Walteros. A note on branch-and-cut-and-price. *Operations Research Letters*, 38(5):346 – 353, 2010.

M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104(1):91–104, 2005.

L. R. Ford and R. Fulkerson. A suggested computation for maximal multi-commodity network flows. *Management Science*, 5(1):97–101, 1958.

R. Fukasawa, H. Longo, J. Lysgaard, M. P. Aragão, M. Reis, E. Uchoa, and R. F. Werneck. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming*, 106(3):491–511, 2006.

G. Gamrath and M. E. Lübbecke. Experiments with a generic dantzig-wolfe decomposition for integer programs. In P. Festa, editor, *Experimental Algorithms*, volume 6049 of *Lecture Notes in Computer Science*, pages 239–252. Springer, 2010.

M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

M. Gendreau and J. Potvin, editors. *Handbook of Metaheuristics*. Springer, 2nd edition, 2010.

A. M. Geoffrion. Lagrangean relaxation for integer programming. In M. L. Balinski, editor, *Approaches to Integer Programming*, volume 2 of *Mathematical Programming Studies*, pages 82–114. Springer, 1974.

P. C. Gilmore and R. E. Gomory. A linear programming approach to the cutting-stock problem. *Operations research*, 9(6):849–859, 1961.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

W. D. Harvey and M. L. Ginsberg. Limited discrepancy search. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, IJCAI'95, pages 607–613, San Francisco, USA, 1995. Morgan Kaufmann.

C. Joncour, S. Michel, R. Sadykov, D. Sverdlov, and F. Vanderbeck. Primal heuristics for branch-and-price. In *European Conference on Operational Research (EURO'10)*, volume 1, page 2, 2010.

E. Lawler and D. Wood. Branch-and-bound methods: A survey. *Operations research*, 14(4):699–719, 1966.

M. E. Lübbecke and J. Desrosiers. Selected Topics in Column Generation. *Operations Research*, 53(6):1007–1023, 2005.

M. E. Lübbecke and C. Puchert. Primal heuristics for branch-and-price algorithms. In *Operations Research Proceedings*. Citeseer, 2011.

J. Lysgaard, A. N. Letchford, and R. W. Eglese. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, 100(2): 423–445, 2004.

V. Maniezzo, T. Stützle, and S. Voß, editors. *Matheuristics - Hybridizing Metaheuristics and Mathematical Programming*, volume 10 of *Annals of Information Systems*. Springer, 2010.

F. Margot. Symmetry in integer linear programming. In M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey, editors, *50 Years of Integer Programming 1958-2008*, pages 647–686. Springer, 2010.

İ. Muter, Ş. İ. Birbil, and K. Bülbül. Simultaneous column-and-row generation for large-scale linear programs with column-dependent-rows. *Mathematical Programming*, 2012a. doi: 10.1007/s10107-012-0561-8.

İ. Muter, Ş. İ. Birbil, K. Bülbül, and G. Şahin. A note on "a lp-based heuristic for a time-constrained routing problem". *European Journal of Operational Research*, 221(2):306 – 307, 2012b.

İ. Muter, Ş. İlker Birbil, K. Bülbül, G. Şahin, H. Yenigün, D. Taş, and D. Tüzün. Solving a robust airline crew pairing problem with column generation. *Computers & Operations Research*, 40(3):815 – 830, 2013.

S. N. Parragh and V. Schmid. Hybrid column generation and large neighborhood search for the dial-a-ride problem. *Computers & Operations Research*, 40(1):490 – 497, 2013.

P. Pesneau, R. Sadykov, and F. Vanderbeck. Feasibility pump heuristics for column generation approaches. *Experimental Algorithms*, pages 332–343, 2012.

D. Pisinger and S. Røpke. Large neighborhood search. *Handbook of metaheuristics*, pages 399–419, 2010.

E. Prescott-Gagnon, G. Desaulniers, and L. M. Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4):190–204, 2009.

E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

A. Schrijver. *Theory of linear and integer programming*. Wiley, 1998.

F. Vanderbeck. *Decomposition and column generation for integer programs*. PhD thesis, Université catholique de Louvain, 1994.

R. J. Vanderbei. *Linear programming*, volume 114. Springer, 2008.

E. J. Zak. Row and column generation technique for a multistage cutting stock problem. *Computers & Operations Research*, 29(9):1143 – 1156, 2002.

Chapter 3
Experimental analysis of different ILP approaches for the Knapsack Prize Collecting Steiner Tree Problem

3.1 Motivations

In this chapter we describe and compare different optimization methods for the *Knapsack Prize Collecting Steiner Tree Problem* (KPCSTP). We consider a Lagrangian Relaxation method (Cordone and Trubian, 2008) and several ILP formulations derived from the ones already proposed in the literature for similar problems (Magnanti and Wolsey, 1995; Ljubić et al., 2006).

The role of this survey in the thesis is to introduce the different ways in which we can tackle combinatorial optimization problems requiring the identification of a constrained tree (resp. arborescence) in a given undirected (resp. directed) graph that minimizes the difference between its cost and the prize of the spanned nodes. Such problems arise in the two following chapters when we need to solve the two Pricing Problems respectively associated with the MRWADC problem and with the HAP. The Pricing Problem associated with the MRWADC problem (see Section 4.5) differs from the KPCSTP for what concerns the definition of feasible solutions: it requires to limit the length of all the paths contained in each feasible arborescence and to limit the outdegree of the spanned nodes. On the other side, the Pricing Problem associated with the HAP (see Section 5.7.2) limits the set of feasible arborescences by introducing a non linear knapsack constraint on the spanned nodes and its cost function is not simply defined by the cost of the crossed arcs, but it depends on the cost associated with particular subsets intersected by the considered solution. Given the peculiarities of these two Pricing Problems, some of the considerations that we make in the following for the KPCSTP cannot be directly extended to them. Moreover, the methods having the best performance for the KPCSTP may perform badly on different problems and vice versa. However, as we discuss in the two following chapters both Pricing Problems can be efficiently solved with some of the methods described in this chapter.

In Section 3.2 we formally describe the KPCSTP and its applications. In Section 3.3 we provide a survey on the optimization methods proposed in the previous literature to tackle the KPCSTP and similar problems. In Section 3.4 we propose three different compact ILP formulations based on network flows. In Section 3.5 we discuss how to adapt a well-know extended formulation for the *Prize Collecting Steiner Tree Problem* (PCSTP) in order to handle the knapsack constraint. In Section 3.6 we describe a Relax-and-Cut method already proposed in the literature to solve the KPCSTP. Finally in Section 3.7 we compare the different optimization methods described in the previous sections.

3.2 The Knapsack Prize Collecting Steiner Tree Problem

The PCSTP is a well known optimization problem defined on an undirected graph $G=(V,E)$, where each vertex $v \in V$ is associated with a prize $p_v \in \mathbb{Q}$ and each edge $e \in E$ is associated with a nonnegative cost $c_e \in \mathbb{Q}^+$. The PCSTP requires to identify a connected subgraph $T=(V_T,E_T)$, with $V_T \subseteq V$ and $E_T \subseteq E$, that minimizes the difference $c(T)$ between the cost of the crossed edges and the profits of the spanned vertices:

$$c(T) = \sum_{e \in E_T} c_e - \sum_{v \in V_T} p_v$$

Note that, since the optimal solution of the PCSTP is a connected subgraph minimizing the cost of its edges and since each edge is associated with a nonnegative cost, we can restrict w.l.o.g. the set of the feasible PCSTP solutions to the set of trees contained in G. For practical reasons, the straightforward extension of the PCSTP called *Rooted Prize Collecting Steiner Tree Problem* (RPCSTP) has been often considered in the literature. This problem is very similar to the PCSTP, however in the RPCSTP we have a single root vertex $r \in V$ that must be contained in each feasible solution. Note that each PCSTP instance can be easily solved through an ILP formulation for the RPCSTP by simply introducing a dummy root vertex r that is connected to each other vertex $v \in V$ at zero cost (i.e. $c_{rv}=0$), fixing $p_r=0$ and by adding a new constraint that limits to one the number of edges exiting from r. Similarly, we can solve a RPCSTP instance with an ILP formulation developed for PCSTP by simply setting $p_r=+\infty$.

The interest in the PCSTP is mainly due to the different practical problems that can be modeled with it. Many of these problems arise from the introduction of deregulation in public utility sectors, such as electricity, telecommunication and gas. In this changed legal environment, the companies operating in these sectors need to focus their efforts on the maximization of the returns of their investments. For instance, when they decide to extend their network to reach new customers, they need to find the best trade-off between the cost that they need to pay to build the new infrastructures and the expected new profits that they gain from the newly reached customers.

The KPCSTP is a straightforward extension of the PCSTP where the set of feasible solutions is shrunk by a knapsack constraint on the set of vertices belonging to each feasible tree. More formally, with each vertex $v \in V$ we associate a nonnegative weight $w_v \in \mathbb{Q}^+$ and we define the weight of each tree $T=(V_T,E_T)$ as follows:

$$w(T) = \sum_{v \in V_T} w_v$$

A subtree T is feasible for the KPCSTP if and only if $w(T) \leq W$ where $W \geq 0$ is a given weight threshold. Also the KPCSTP admits a variant in which we have a mandatory root vertex r. We denote this variant by RKPCSTP and, similarly to what noted for the PCSTP, it is straightforward to transform an ILP formulation for the RKPCSTP to an ILP formulation for the KPCSTP. As a consequence, in the remaining part of this chapter, we focus our attention only on the rooted variant of the KPCSTP, since it allows us to simplify several notations.

The application areas of KPCSTP are similar to the ones previously described for the PCSTP. However, the KPCSTP provides more extended modeling capabilities. Consider, as examples, the extension of a telecommunication network based on a central broadcasting station having a limited power, or the extension of a network in a electricity market where the operators needs to comply with the government laws aiming to contrast monopolies: in both cases, we can limit the dimension of the final network and satisfy the practical requirements by introducing a knapsack constraint in the PCSTP.

3.3 Survey of the literature

The PCSTP has been deeply investigated in the literature. It was first proposed in (Bienstock et al., 1993).However, yet in (Segev, 1987), the author described a problem that is very similar to the RPCSTP, and in (Duin and Volgenant, 1987) have been developed some reduction procedures for the *Steiner Tree Problem* (STP) that can be extended and applied also to the PCSTP.

During the last two decades different exact methods have been proposed for the PCSTP. A Lagrangian Relaxation algorithm has been proposed in (Engevall et al., 1998), Branch & Cut methods have been developed in (Lucena and Resende, 2004) and in (Ljubić et al., 2006). The algorithm presented in the latter paper represents the current state of the art in solving the PCSTP.

At the best of our knowledge the KPCSTP has been studied only in (Cordone and Trubian, 2008) where the authors proposed an exact method based on the Lagrangian Relaxation method proposed in (Engevall et al., 1998) for the PCSTP. We describe in details this method in Section 3.6. In Section 3.7 we compare this approach with other optimization methods.

Finally, in (Haouari et al., 2008) the authors proposed a unifying framework that can be easily adapted to model both the PCSTP and other well-known tree optimization problems. Along with this framework the authors of the previous paper propose different optimization methods to solve the associated Lagrangian Dual Problem.

3.4 Compact formulations based on network flows

The first ILP technique we consider to correctly define the feasible region of the KPCSTP is based on the definition of feasible flows on a network derived from G. We consider two different ways of defining network flows starting from a KPCSTP instance: in the first one (see Section 3.4.1) we use a single-commodity flow that is generated from r and such that each vertex belonging to a feasible solution absorbs one unit of it; in the second one (see Section 3.4.2) we use a multi-commodity network flow where each vertex of the considered instance has its own commodity. We generate from r a single unit of flow for each commodity and each vertex belonging to the considered feasible solution absorbs one unit of the associated flow.

Both these network flow formulations are defined on the directed graph $\tilde{G} = (V,A)$ obtained from G by duplicating each edge in E into a pair of arcs having opposite directions, i.e. $A = \{(i,j),(j,i) : \{i,j\} \in E\}$ and by associating a cost c_a to each arc $a \in A$ defined as follows: $c_{ij} = c_{ji} = c_{\{i,j\}}$ for each $\{i,j\} \in E$.

Note that, by using $\tilde{G}$ instead of G, in the two following formulations, we take decisions on $\tilde{G}$ arcs and not on G edges. Consequently, the obtained solutions are arborescences and not trees. However, given an optimal arborescence it is straightforward to obtain the corresponding optimal tree: if the optimal arborescence contains arc (i,j), the corresponding optimal tree must contain edge $\{i,j\}$.

3.4.1 Single-commodity flow formulations

After introducing the following families of decision variables:

- x_a, for each arc $a \in A$, is equal to 1 if arc a belongs to the optimal arborescence, 0 otherwise.
- y_v, for each vertex $v \in V$, is equal to 1 if vertex v belongs to the optimal arborescence, 0 otherwise.
- f_a, for each arc $a \in A$, denotes the units of flow going through arc a.

We can state the single-commodity network flow formulation for the KPCSTP as follows:

$$\text{SCF}: \min \sum_{a\in A} c_a x_a - \sum_{v\in V} p_v y_v \tag{3.1a}$$

$$\sum_{a\in\delta^-(v)} x_a = y_v \qquad v \in V\setminus\{r\} \tag{3.1b}$$

$$y_r = 1 \tag{3.1c}$$

$$\sum_{v\in V} w_v y_v \le W \tag{3.1d}$$

$$f_a \le\mid V\mid x_a \qquad a\in A \tag{3.1e}$$

$$\sum_{a\in\delta^+(r)} f_a = \sum_{v\in V\setminus\{r\}} y_v \tag{3.1f}$$

$$\sum_{a\in\delta^-(v)} f_a - \sum_{a\in\delta^+(v)} f_a = y_v \qquad v\in V\setminus\{r\} \tag{3.1g}$$

$$x_a, y_v \in \{0,1\} \qquad a\in A, v\in V \tag{3.1h}$$

$$f_a \ge 0 \qquad a\in A \tag{3.1i}$$

The objective function (3.1a) minimizes the difference between the cost of the crossed arcs and the prizes of the spanned vertices. Constraints (3.1b) link the arc variables $\{x_a : a \in A\}$ with the vertex variables $\{y_v : v \in V\}$: if exists an arc $a \in A$ entering in vertex $v \in V$, vertex v must be spanned by the considered solution. Constraint (3.1c) requires that each feasible solution contains the root. Constraint (3.1d) guarantees that the solution respects the weight threshold W. Constraints (3.1e) link together the arc variables $\{x_a : a \in A\}$ and the flow variables $\{f_a : a \in A\}$: if a positive flow goes through an arc we need to take it in the solution. Constraint (3.1f) imposes the generation of the right amount of flow starting from r (one unit of flow for each spanned node, with the exception of the root). Constraints (3.1g) preserve the flow balance: each spanned vertex $v \in V \setminus \{r\}$ absorbs one unit of flow, propagating the remaining ones. Finally, constraints (3.1h) and (3.1i) define the correct domain of the decision variables.

Note that the number of vertices $\mid V\mid$ plays a big-M role in constraints (3.1e) and, as a consequence, it weakens the lower bound obtained by solving the LP-Relaxation of formulation (3.1). We can mitigate its effect by imposing the satisfaction of the knapsack constraint using a different flow definition: each vertex $v \in V$ absorbs w_v units of flow, propagating the remaining ones, and, starting from the root, we generate $\sum_{v\in V\setminus\{r\}} w_v$ units of flow. Following this scheme, we can substitute coefficient $\mid V\mid$ in constraints (3.1e) with threshold W and we can remove constraint (3.1d) since now it is implicitly satisfied by each feasible flow. The final obtained formulation can be stated as follows:

$$\text{SCF2} : \min \sum_{a\in A} c_a x_a - \sum_{v\in V} p_v y_v \tag{3.2a}$$

$$\sum_{a\in\delta^-(v)} x_a = y_v \qquad v\in V\setminus\{r\} \tag{3.2b}$$

$$y_r = 1 \tag{3.2c}$$

$$\sum_{a\in\delta^+(r)} f_a \le W - w_r \tag{3.2d}$$

$$f_a \le W x_a \qquad a\in A \tag{3.2e}$$

$$\sum_{a\in\delta^+(r)} f_a = \sum_{v\in V\setminus\{r\}} w_v y_v \tag{3.2f}$$

$$\sum_{a\in\delta^-(v)} f_a - \sum_{a\in\delta^+(v)} f_a = w_v y_v \qquad v\in V\setminus\{r\} \tag{3.2g}$$

$$x_a, y_v \in \{0,1\} \qquad a\in A, v\in V \tag{3.2h}$$

$$f_a \ge 0 \qquad a\in A \tag{3.2i}$$

With respect to the previous formulation (3.1), in this formulation we changed the flow definition constraints (3.2e - 3.2g), we removed the knapsack constraint (3.1d) and we introduced the constraint (3.2d) in order to limit the amount of flow exiting from the root.

3.4.2 Multi-commodity flow formulation

Similarly to what done in the two previous formulations, to define the multi-commodity network flow formulation, we use variables x_a for each $a \in A$ and y_v for each $v \in V$ to describe which arcs and vertices belong to the optimal solution. However, differently from what done in the previous formulations, in this new one, we define a commodity for each vertex $v \in V \setminus \{r\}$ and a binary variable $f_a^v \in \{0,1\}$ for each arc $a \in A$ that describes the amount of flow associated with commodity v crossing arc a.

Once defined these new variables, we can state the multi-commodity network flow formulation for the KPCSTP as follows:

$$\text{MCF} : \min \sum_{a \in A} c_a x_a - \sum_{v \in V} p_v y_v \tag{3.3a}$$

$$\sum_{a \in \delta^-(v)} x_a = y_v \qquad v \in V \setminus \{r\} \tag{3.3b}$$

$$y_r = 1 \tag{3.3c}$$

$$\sum_{v \in V} w_v y_v \leq W \tag{3.3d}$$

$$f_a^v \leq x_a \qquad a \in A, v \in V \tag{3.3e}$$

$$\sum_{a \in \delta^+(r)} f_a^v = y_v \qquad v \in V \setminus \{r\} \tag{3.3f}$$

$$\sum_{a \in \delta^-(w)} f_a^v - \sum_{a \in \delta^+(w)} f_a^v = 0 \qquad v, w \in V \setminus \{r\}, v \neq w \tag{3.3g}$$

$$\sum_{a \in \delta^-(v)} f_a^v - \sum_{a \in \delta^+(v)} f_a^v = y_v \qquad v \in V \setminus \{v\} \tag{3.3h}$$

$$x_a, y_v \in \{0, 1\} \qquad a \in A, v \in V \tag{3.3i}$$

$$f_a^v \geq 0 \qquad a \in A, v \in V \setminus \{r\} \tag{3.3j}$$

The objective function (3.3a) minimizes the difference between the cost of the crossed arcs and the prize of the spanned vertices. Constraints (3.3b) link the arc variables $\{x_a : a \in A\}$ with the vertex variables $\{y_v : v \in V\}$: if exists an arc $a \in A$ entering in vertex $v \in V$, it must be spanned by the solution. Constraint (3.3c) requires that each feasible solution contains the root. Constraints (3.3e) link together the arc variables $\{x_a : a \in A\}$ and the flow variables $\{f_a^v : a \in A, v \in V\}$: if the amount of the flow associated with any commodity v, crossing arc $a \in A$ is positive, we need to take a in the solution. Constraints (3.3f) impose the generation of one unit of the flow associated with commodity $v \in V \setminus \{r\}$ if and only if vertex v belongs to the solution. Constraints (3.3h) impose that each vertex $v \in V \setminus \{r\}$ absorbs one unit of flow of the corresponding commodity v, if and only if v belongs to the solution. Constraints (3.3g) impose that each vertex $w \setminus \{v, r\}$ propagates all the flow associated with commodity $v \neq w$ they received. Finally, constraints (3.3i) and (3.3j) define the correct domain of the decision variables.

3.5 Extended formulation based on the connectivity cuts

The second ILP technique we consider to correctly define the feasible region of the KPCSTP are the connectivity cuts. The ILP formulation stated in the following is the natural extension to the KPCSTP of the ILP formulation for the PCSTP proposed in (Ljubić et al., 2006). As already remembered in Section 3.3, the optimization method derived from the latter formulation is the best exact method proposed in the literature for the PCSTP.

The extension of that PCSTP formulation to the KPCSTP is straightforward. We start by defining the following decision variables:

- x_a, for each $a \in A$, is equal to 1 if arc a belongs to the optimal solution, 0 otherwise.
- y_v, for each $v \in V$, is equal to 1 if vertex v belongs to the optimal solution, 0 otherwise.

Then we can state the following ILP formulation:

$$\text{CTF}: \min \sum_{a\in A} c_a x_a - \sum_{v\in V} p_v y_v \tag{3.4a}$$

$$\sum_{a\in\delta^-(v)} x_a = y_v \qquad v \in V\setminus\{r\} \tag{3.4b}$$

$$y_r = 1 \tag{3.4c}$$

$$\sum_{v\in V} w_v y_v \le W \tag{3.4d}$$

$$\sum_{a\in\delta^+(S)} x_a \ge y_v \qquad v \in V\setminus\{r\}, S\subseteq V, r\in S, v\in V\setminus S \tag{3.4e}$$

$$x_a, y_v \in \{0,1\} \qquad a\in A, v\in V \tag{3.4f}$$

The objective function (3.4a) minimizes the difference between the cost of the crossed arcs and the prizes of the spanned vertices. Constraints (3.4b) link the arc variables $\{x_a : a \in A\}$ with the vertex variables $\{y_v : v \in V\}$: if exists an arc $a \in A$ entering in vertex $v \in V$, the vertex v must be spanned by the solution. Constraint (3.4c) requires that each feasible solution contains the root. For each vertex $v \in V$ belonging to the considered solution, constraints (3.4e) impose that at least one arc of each cut separating r from v belongs to the solution.

3.5.1 Separation algorithm

Since the number of constraints (3.4e) is exponential in the number of vertices $|V|$, to handle these inequalities we use the Branch & Cut framework. To solve the separation problem associated with these constraints, we use the same method proposed in (Ljubić et al., 2006). This method is based on the computation of at most n different maximum flows in a network derived from the current solution. Let $(\underline{x}', \underline{y}')$ be a solution that satisfies constraints (3.4b,3.4d) and the continuous relaxation of constraints (3.4f). For each $v \in V$ such that $y'_v > 0$, let ϕ be the value of the maximum flow from r to v in the network defined by $\tilde{G}$ and in which each arc a has a capacity equal to x'_a. There are only two possible cases:

- If $\phi < y_v$ then the MAX FLOW - MIN CUT theorem (see, for example, Lawler 1976) ensures the existence of a cut $(S, V \setminus S)$, with $r \in S$ and $v \in V \setminus S$ and with capacity equal to ϕ. Therefore, the constraint

$$\sum_{a\in\delta^-(S)} x_a \geq y_v$$

is violated and we can add it to the considered formulation, to cut the unfeasible solution $(\underline{x}', \underline{y}')$.

- If $\phi \geq y_v$ then the same theorem ensures that all the $r-v$ network cuts have a capacity not less than ϕ. Therefore, for each $S \subset V \setminus \{s\}$ with $r \in S$ and $v \notin S$, the constraint

$$\sum_{a\in\delta^-(S)} x_a \geq y_v$$

is satisfied by $(\underline{x}', \underline{y}')$ and we can stop the generation of the connectivity cuts constraints.

To increase the number of cuts added to the formulation at each iteration of the cutting plane algorithm we improved the separation algorithm with the two following techniques, already proposed in (Ljubić et al., 2006):

- *Back cuts*: in not trivial cases a feasible network flow is associated with two different minimum cuts, one near the source and another near the sink. Using an algorithm belonging to the push-relabel family (Goldberg and Tarjan, 1988), we can find these two cuts using only one max flow computation.
- *Nested cuts*: once discovered a violated cut, we set the capacities of the arcs contained in this cut to one and we repeat the violated cuts searching process on the modified network. In this manner, we can find more violated cuts in each separation algorithm execution.

3.5.2 Cuts pool initialization

As already noted in (Ljubić et al., 2006), a crucial aspect that need to be considered in order to obtain an effective Branch & Cut algorithm based on formulation CTF (3.4) is the way in which the pool of the connectivity cuts (3.4e) is initialized in the root vertex of the branching tree. In the computational campaign described in Section 3.7, we considered two different strategies:

- *Empty pool*: the initial LP formulation associated with the root vertex of the branching tree does not contain any constraint of family (3.4e).
- *2-cycle removal*: we initialize the pool of the connectivity cuts with all the constraints that impose the removal from each feasible solution of all the cycles having a length equal to two:

$$x_{(i,j)} + x_{(j,i)} \leq 1 \quad \{i,j\} \in E \setminus \delta(r) \tag{3.5}$$

3.6 A Relax-and-Cut method for the KPCSTP

In (Cordone and Trubian, 2008), the authors describe how to extend the methods proposed by Beasley (see, Beasley 1989) for the Steiner Tree Problem and by Engevall et al. (Engevall et al., 1998) for the PCSTP, in order to handle the knapsack constraint. Here we present a short survey of this method, and for its full details the interested reader is referred to the original paper.

The method is defined on an auxiliary undirected graph $\hat{G} = (\hat{V}, \hat{E})$ that is derived from $G = (V,E)$ by adding a dummy vertex v_0, linked to all the vertices in V (i.e. $\hat{V} = V \cup \{v_0\}$ and $\hat{E} = E \cup \{\{v_0, v\} : v \in V\}$). We associate a nonnegative cost $\hat{c}_e$ with each edge $e \in \hat{E}$ as follows:

$$\hat{c}_e = \begin{cases} c_e & \text{if } e \in E \\ 0 & \text{if } e = \{v_0, r\} \\ p_v & \text{if } e = \{v_0, v\} \text{ and } v \in V \setminus \{r\} \end{cases}$$

The idea behind this method is to uniquely associate each subtree of G which is a feasible solution for the KPCSTP to a special spanning tree $T = (U,X)$ defined on $\hat{G}$: T is feasible if and only if all the subtrees appended to the dummy vertex v_0 are leaves, with the exception of the subtree rooted in r.

Given a spanning tree $T = (U,X)$ defined on $\hat{G}$ and satisfying the previous condition, we can consider its subtree rooted in r denoted by $T' = (U', X')$ where $U' = U \setminus \{v_0\} \setminus \{v \in V : \{v_0, v\} \in X\}$ and $X' = X \setminus \{\{v_0, v\} : v \in V\}$. If T' satisfies the knapsack constraint, it is also a feasible solution for the KPCSTP.

The cost $\sum_{e \in X} \hat{c}_e$ of each special spanning tree $T = (U,X)$ and the cost of the corresponding KPCSTP solution $T' = (U', X')$ differ only by the constant term $\sum_{v \in V'} p_v$:

$$\sum_{e \in X} \hat{c}_e = \sum_{e \in X'} c_e + \sum_{v \in V \setminus U'} \hat{c}_{\{v_0, v\}} = \sum_{e \in X'} c_e + \sum_{v \notin U'} p_v = c(T) + \sum_{v \in V\prime} p_v$$

Thus, we can solve the KPCSTP by simply identifying the special spanning tree, defined on $\hat{G}$, having the minimum cost. To find this special spanning tree we introduce the binary variable x_e that is equal to 1 if and only if $e \in \hat{E}$ is contained in the optimal solution and we solve the following ILP formulation:

$$\text{SST}: \min \sum_{e\in\hat{E}} \hat{c}_e x_e \tag{3.6a}$$

$$\sum_{e\in\hat{E}} x_e = \mid \hat{V} \mid -1 \tag{3.6b}$$

$$\sum_{e\in\hat{E}(P)} x_e \le \mid P \mid -1 \qquad \emptyset \subset P \subset \hat{V} \tag{3.6c}$$

$$\sum_{e\in\hat{E}(P)} x_e + \sum_{v\in P\setminus\{u\}} x_{v_0 v} \le \mid P \mid -1 \qquad P \subseteq \hat{V}\setminus\{r\}, u \in P \tag{3.6d}$$

$$\sum_{v\in V\setminus\{r\}} w_v x_{v_0 v} \ge \sum_{v\in V} w_v - W \tag{3.6e}$$

$$x_{v_0 r} = 1 \tag{3.6f}$$

$$x_e \in \{0,1\} \qquad e \in \hat{E} \tag{3.6g}$$

Constraint (3.6b) imposes the correct number of edges that must belong to a feasible solution. The subtours elimination constraints (3.6c) impose the removal of all the cycles in each feasible solution. Constraints (3.6d) are a generalization of the subtours elimination constraints and impose that each subtree, directly connected with v_0 and having a vertex $v \in V \setminus \{r, v_0\}$ as root, is a leaf. Constraint (3.6e) imposes the satisfaction of the knapsack constraint (note that $\sum_{v\in V} w_v - \sum_{v\in V\setminus\{r\}} w_v x_{v_0 v}$ is equal to the sum of the weights of the vertices belonging to the subtree rooted in r). Finally, constraint (3.6f) ensures that r is directly connected with v_0 and constraints (3.6g) define the correct domain of the decision variables.

3.6.1 The Lagrangian Dual Problem

The number of constraints (3.6c,3.6d) is exponential in the number of $\hat{G}$ nodes. Thus, we cannot directly solve the LP-Relaxation of formulation (3.6) with a standard LP method. However, if we relax constraints (3.6d) and the knapsack constraint (3.6e), the remaining constraints define the polytope of the minimum cost spanning tree where the edge $\{v_0, r\}$ is imposed in each feasible solution. To exploit this peculiarity, we can associate a lagrangian multiplier λ_{Pu} with each constraint (3.6d), that is indexed by the vertices subset $P \subseteq \hat{V} \setminus \{r\}$ and by the reference vertex $u \in P$, and we can associate a lagrangian multiplier λ^W with the knapsack constraint (3.6e). Then, we can state the Lagrangian Dual Problem as follows:

$$DL: \max_{\underline{\lambda}\ge 0} \min_{\underline{x}\in\{0,1\}^{|E|}} \sum_{e\in\hat{E}} \overline{c}_e x_e - \sum_{u\in\hat{V}} \sum_{P\in\mathscr{P}_u} (\mid P \mid -1)\, \lambda_{Pu} + \lambda^W \left(\sum_{v\in V} w_v - W \right) \tag{3.7a}$$

$$\{x_e\} \text{ define a spanning tree on } \tilde{G} \tag{3.7b}$$

$$x_{v_0 r} = 1 \tag{3.7c}$$

Note that in this formulation, for each $u \in \hat{V}$ we use $\mathscr{P}_u$ to denote the family of sets containing the subsets of $\hat{V}$ that do not contain r and contain u, i.e. all the subsets of vertices associated with constraints (3.6d). Finally, the modified edge cost $\overline{c}_e$ is obtained by perturbing the original cost c_e with the lagrangian multiplier as follows:

$$\overline{c}_e = \begin{cases} \hat{c}_{rv} & \text{if } e = \{r,v\} \text{ and } v \in V \\ \hat{c}_{v_0 v} - \lambda^W w_v + \sum_{P \in \mathscr{P}_v} \sum_{u \in P \setminus \{v\}} \lambda_{Pu} & \text{if } e = \{v_0, v\} \text{ and } v \in V \setminus \{r\} \\ \hat{c}_{vw} + \sum_{P \in \mathscr{P}_w} \sum_{u \in P} \lambda_{Pu} & \text{if } e = \{v,w\} \text{ and } v,w \in V \setminus \{r\} \end{cases}$$

3.6.2 *Solving the Lagrangian Dual Problem*

The method proposed in (Cordone and Trubian, 2008) to solve the Lagrangian Dual Problem (3.7) is an implementation of the Relax-and-Cut approach already proposed in (Engevall et al., 1998). They started by noting that most of the relaxed constraints (3.6d) are inactive for the optimal solution and, as a consequence, the best value for the corresponding multipliers is zero. Hence, the main idea behind this approach is to consider only the constraints that are violated by the solutions of the lagrangian subproblems solved so far. The correct values for this small subset of lagrangian multipliers can be easily estimated using the subgradient ascent method (Held and Karp, 1970). Here we consider the implementation of the subgradient ascent method proposed in (Cordone and Trubian, 2008) that makes use of the modified subgradient updating strategy proposed in (Camerini et al., 1975),

3.6.3 *A lagrangian heuristic*

At each step of the subgradient algorithm we consider the subtree $T' = (U', X')$ rooted in r and directly connected with v_0. If T' satisfies the knapsack constraint and each other subtree connected to v_0 is a leaf, T' represents a feasible solution of the KPCSTP. Otherwise, we can apply the following three-steps heuristic to regain feasibility:

1. If $w(T') > W$, the heuristic regains the knapsack feasibility by removing from T' its leaf v having the largest ratio $w_v/(p_w - c_{uv})$ where the vertex u is the unique other T' vertex connected with v. This step is recursively executed until the weight of the considered tree satisfies the knapsack constraint.
2. If $w(T') < W$ the heuristic iteratively adds leaves to T' using a greedy procedure: at each iteration, it finds the edge $\{u,v\}$ with $u \in U'$ and $v \notin U'$ having the highest ratio $(p_v - c_{uv})/w_v$ among the edges that can be introduced into T' maintaining its feasibility and without decreasing the corresponding objective function value.
3. Once fixed the set of vertices U' contained in the subtree rooted in r, we can simply find the tree spanning these vertices at the minimum cost by using any

fast algorithm for the minimum spanning tree problem (here we use the Kruskal algorithm, see Kruskal 1956). Hence, during the last step of the heuristic procedure, we can substitute T' with the computed minimum spanning tree, possibly improving the objective function value.

3.6.4 A Tabu Search initialization heuristic

To fairly compare the Lagrangian Relaxation method with the optimization methods based on the ILP formulations described in the previous sections, we need to introduce a primal heuristic in the former method since the latter ones benefit of the advanced primal heuristics nowadays built in the commercial ILP solvers (Danna et al., 2005; Rothberg, 2007; Achterberg and Berthold, 2007; Lodi, 2013). Hence, in order to define an algorithm based on the previously introduced Lagrangian Relaxation method that is competitive with the methods based on the ILP formulations, described in the previous sections, that can be directly solved by a commercial ILP solver, we initialize the former method with an efficient primal heuristic. In (Cordone and Trubian, 2008) the authors describe a simple Tabu Search heuristic to determine a good initial feasible solution. The heuristic is initialized with the minimum spanning tree obtained by setting to zero all the constraint multipliers in formulation (3.7). The neighborhood explored at each iteration by the heuristic contains all the spanning trees that can be obtained by adding (resp. removing) a vertex to (resp. from) the current subtree rooted in r. The neighborhood definition determines which vertices are contained in the subtree rooted in r and we compute the best way to connect them by computing the tree that spans them all at the minimum cost. The heuristic avoids cycling behaviors by means of a tabu memory that forbids the reversal of executed moves for a certain number of iterations. This number is controlled by a tabu tenure that is dynamically adapted to the state of the search.

3.6.5 Embedding the Lagrangian Dual Problem in an exact method

It is possible that the spanning tree obtained at the end of the Lagrangian Dual Problem (3.7) resolution process violates some of the constraints (3.6d). However, since the minimum spanning tree polytope has the integrality property, the objective function value associated with this tree is equal to the objective function value of the optimal solution of the LP-Relaxation of the original formulation (3.6). Thus, we have an effective lower bounding procedure that can be embedded in a Branch & Bound algorithm in order to obtain an optimal solution for the KPCSTP.

Branching strategy

We define a branching strategy based on variable x_{v_0v} with $v \in V \setminus \{r\}$: on the first branch we impose the presence of edge $\{v_0, v\}$ in each feasible solution (i.e. removing vertex v in the corresponding KPCSTP feasible solutions), on the other branch we forbid the inclusion of the same edge in any feasible solution (i.e. including vertex v in the corresponding KPCSTP feasible solutions). The constraints added to problem (3.7) in both branches can be easily handled by simply modifying the Kruskal algorithm. Thus, the lagrangian subproblems obtained after the introduction of the branching constraints continue to be effectively solvable with the method described in Section 3.6.2.

Branching variable selection

The choice of the branching variable x_{v_0v} is based on the complementary slackness conditions. First, we denote with λ the best lagrangian multiplier vector found so far and we denote with s the slack vector of the corresponding lagrangian solution T^0_λ. Given these two vectors, among the constraints (3.6d) that are violated by T^0_λ, we select the one, indexed by i, for which $\mid \lambda_i s_i \mid$ is maximum and we branch on x_{v_0t}, where t is the root of the subtree associated with the violated constraint.

If all the constraints (3.6d) are satisfied by T^0_λ, we consider the constraint, indexed by i, for which $\mid \lambda_i s_i \mid$ is maximum: if this constraint is oversatisfied and belongs to the structural constraints (3.6d), we choose the branching variable x_{v_0u} where u is the reference vertex associated with it; otherwise, the selected constraint i must be the knapsack constraint (3.6e), and we branch on variable x_{v_0v} where v is the vertex adjacent to v_0 in T^0_λ associated with the minimum lagrangian cost $\overline{c}_{v_0v}$. Finally, if no vertex is adjacent to v_0 in T^0_λ, we select the branching variable x_{v_0v} where v is associated with the minimum lagrangian cost $\overline{c}_{v_0v}$.

Branching tree exploration

For the sake of simplicity, and to easily handle the generated cuts in the different branching nodes, we decide to adopt a depth-first strategy in which we first visit the branch associated with constraint $x_{v_0v} = 1$ (i.e. the one imposing the removal of vertex v from the KPCSTP solution) and then we visit the branch associated with constraint $x_{v_0v} = 0$ (i.e. the one imposing the presence of vertex v into the KPCSTP solution).

3.7 Computational Experiments

In this section we describe the computational experiments that we executed in order to asses the performances and the properties of the different methods we have described in the previous sections.

All the methods have been implemented in C++ and built with gcc 4.6, setting the -O3 optimization flag. To solve the LP/ILP problems, we used ILOG CPLEX 12.5 and the CONCERT libraries. Note that, among all the proposed methods, only the Relax-and-Cut method described in Section 3.6 does not require a LP/ILP solver.

All the experiments have been executed on a PC equipped with an Intel Xeon Processor E5-1620 Quad Core 3.60GHz and 16 GB of RAM. Since the Relax-and-Cut method have a single-threaded implementation, for the sake of fairness, we limited to one the number of threads that CPLEX could use.

3.7.1 Benchmark instances

In this computational campaign, we consider the same instances considered in (Cordone and Trubian, 2008). These instances can be divided in three different benchmarks:

- *Dense instances* (`D`): the instances in this benchmark have been derived from the instances for the PCSTP proposed in (Engevall et al., 1998), the number of vertices ranges from 10 to 100 by steps of 10 nodes, the underlying graph is complete and the edge costs are randomly extracted from a uniform distribution ranging in $[50;150]$. The prizes are chosen at random from a uniform distribution, and the range of this distribution depends on the group to which the instance belongs. There are three groups: in the *low prize* group the prizes range in $[40;100]$, in *medium prize* group the prizes range in $[40;150]$, and in *large prize* group the prizes range in $[40;200]$. For each combination of number of vertices and prize interval there are 5 different instances. Overall this benchmark contains 150 instances.
- *Sparse planar instances* (`P`): these instances have been generated extracting a given number of vertices from a 100×100 grid. The number of vertices ranges from 50 to 100 by steps of 10 and from 150 to 300 by steps of 50. The edges are randomly generated by extracting a couple of vertices and inserting the corresponding edge if and only if its length is less than or equal to $\alpha = 0.3$ of the length of the grid diagonal, and its insertion preserves the planarity of the graph. For each number of vertices, 5 different graphs have been generated, and, for each one of them, random prizes have been generated both in range $[-50;100]$ and in range $[0;100]$. Overall, the number of instances in this benchmark is 100.
- *Non planar instances* (`R`): the instances in this benchmark have been generated following a strategy similar to the one used for the generation of the instances of the previous benchmark. The unique difference between the two benchmarks

is that the graphs generated for benchmark `R` are not planar, the edge associated with a given couple of vertices belongs to the generated graph with probability π and if and only if its length is less than or equal to $\alpha = 0.3$ of the length of the grid diagonal. In order to generate instances with different degrees of sparsity, different values for π (0.05 and 0.1) have been considered . As before, for each number of vertices, 5 different graphs have been generated. Overall, the number of instances in this benchmark is 200.

In all the instances of the three benchmarks, all the vertex weights have been fixed to one (i.e. $w_v = 1$ for each $v \in V$), and we set $W = \lfloor \delta \mid V \mid \rfloor$ where δ is an instance parameter varying in $\{0.2, 0.4, 0.6, 0.8\}$. In this way we obtained instances having different levels of tightness in the knapsack constraint.

3.7.2 Network flow formulations comparison

In the first computational experiment, we compare the three different compact formulations based on network flows. In particular, we compare the single-commodity formulation SCF (3.1), its variant SCF2 (3.2) that handles the knapsack constraint in an implicit way and the multi-commodity formulation MCF (3.3). For each instance and for each formulation, we consider a time limit of 600 seconds. In Table 3.1, Table 3.2 and Table 3.3 we respectively report the average computational time in seconds required by the three network flow formulations to respectively solve the instances in benchmark `D`, benchmark `P` and benchmark `R`. The average computational time has been computed only on the instances that can be solved within the time limit. If some of the instances in a given class cannot be solved to optimality within the time limit, we report, within parenthesis, the number of unsolved instances. Note that, for benchmark `R`, using formulation MCF, CPLEX is not able to solve any instance having $n = 300$ and $\delta = 0.6$, as a consequence, we cannot compute the average computational time for these instances, and we denote this fact by a dash "-". Finally, for each class of instances, we highlight with a bold font the smallest average computational time.

Analyzing these tables we can observe that, when the number of vertices increases too much, formulation MCF is not competitive with the two single-commodity formulations. This behavior can be easily explained considering the different formulation sizes: formulation MCF has $O(\mid E \mid\mid V \mid)$ variables and $O(\mid E \mid\mid V \mid)$ constraints, while the two single-commodity formulation (SCF and SCF2) have only $O(\mid E \mid + \mid V \mid)$ variables and $O(\mid E \mid + \mid V \mid)$ constraints. Nonetheless, the number of branching nodes generated by CPLEX to solve formulation MCF is very low w.r.t. the number of branching nodes generated to solve formulations SCF and SCF2. As example, in Table 3.4 we report the average number of branching nodes generated by CPLEX to solve the three different network flow formulations on the instances belonging to benchmark `P` having $\delta = 0.8$ and varying the number of vertices. On these instances, the average number of branching nodes generated to solve MCF is always less than 1, this implies that most instances can be solved at the

	$\delta = 0.2$			$\delta = 0.4$			$\delta = 0.6$			$\delta = 0.8$		
n	MCF	SCF	SCF2	MCF	SCF	SCF2	MCF	SCF	SCF2	MCF	SCF	SCF2
10	**0.01**	**0.01**	**0.01**	**0.07**	0.13	**0.07**	**0.01**	**0.01**	**0.01**	0.07	**0.01**	**0.01**
20	0.33	0.20	**0.01**	0.53	0.27	**0.01**	0.47	**0.07**	**0.07**	0.40	0.13	**0.07**
30	3.33	**0.07**	0.20	2.33	0.53	**0.47**	2.40	**0.40**	**0.40**	2.93	**0.27**	**0.27**
40	9.80	0.60	**0.27**	7.93	**0.73**	1.13	8.00	0.60	**0.53**	7.20	0.60	**0.53**
50	29.93	1.33	**1.13**	19.27	1.67	**1.53**	21.67	**1.20**	**1.20**	22.20	**1.20**	1.40
60	73.33	2.40	**2.07**	45.20	3.13	**2.67**	39.87	2.47	**2.20**	44.60	**2.20**	**2.20**
70	110.93	5.20	**3.20**	102.40	**5.87**	**5.87**	104.40	4.33	**3.93**	88.80	**4.27**	4.73
80	196.87	9.07	**6.13**	164.60	11.93	**11.00**	185.80	7.80	**7.00**	177.80	**7.40**	7.73
90	227.09(4)	14.73	**9.93**	215.20(5)	15.20	**14.07**	232.82	11.47	**10.33**	254.64(1)	**10.53**	10.80
100	355.73(4)	24.07	**20.27**	395.54(3)	25.87	**21.67**	406.58(3)	20.07	**17.93**	387.60(5)	20.67	**19.13**

Table 3.1 Average computational time required by the different network flow formulations on dense instances (benchmark `D`).

	$\delta = 0.2$			$\delta = 0.4$			$\delta = 0.6$			$\delta = 0.8$		
n	MCF	SCF	SCF2	MCF	SCF	SCF2	MCF	SCF	SCF2	MCF	SCF	SCF2
50	1.20	0.60	**0.10**	0.90	0.90	**0.60**	0.90	0.60	**0.30**	1.00	0.50	**0.10**
60	1.90	0.60	**0.20**	1.20	**0.60**	0.70	1.00	0.40	**0.10**	1.40	0.50	**0.30**
70	2.70	1.90	**0.90**	2.20	1.30	**1.20**	1.60	**0.50**	0.60	1.80	0.50	**0.40**
80	3.40	1.40	**0.70**	3.30	1.90	**0.90**	2.70	**1.00**	**1.00**	2.30	**0.80**	0.90
90	6.20	7.00	**2.40**	4.70	6.10	**3.70**	3.90	1.20	**0.90**	2.80	**1.00**	**1.00**
100	7.50	4.50	**1.90**	6.70	2.00	**1.10**	4.10	1.40	**0.80**	4.10	**1.30**	1.70
150	33.20	10.30	**8.10**	19.40	**10.00**	13.40	18.60	7.70	**5.60**	17.50	**5.40**	6.00
200	92.40	41.60	**27.10**	95.10	29.50	**19.60**	86.60	**12.40**	13.10	63.30	**8.20**	**8.20**
250	261.20	124.11(1)	**54.70**	145.90	49.20	**26.90**	174.10	15.60	**12.80**	116.80	**9.50**	10.40
300	352.40	112.25(2)	**50.20**	296.40(2)	96.60(1)	**62.60**	259.30	21.30	**18.30**	280.60	**16.00**	17.30

Table 3.2 Average computational time required by the different network flow formulations on planar instances (benchmark `P`).

	$\delta = 0.2$			$\delta = 0.4$			$\delta = 0.6$			$\delta = 0.8$		
n	MCF	SCF	SCF2	MCF	SCF	SCF2	MCF	SCF	SCF2	MCF	SCF	SCF2
50	0.60	0.35	**0.15**	0.70	**0.30**	0.40	0.75	0.25	**0.20**	0.60	5.15	**0.10**
60	1.10	0.50	**0.15**	1.00	**0.30**	0.45	1.05	0.35	**0.15**	1.15	6.60	**0.35**
70	2.10	0.35	**0.20**	2.05	0.65	**0.55**	2.15	**0.50**	0.55	2.00	2.85	**0.40**
80	4.30	0.75	**0.60**	3.25	0.70	**0.40**	2.95	**0.50**	**0.50**	3.20	4.45	**0.40**
90	6.10	**0.95**	**0.95**	5.50	0.90	**0.60**	5.90	**0.50**	0.55	4.60	2.10	**0.45**
100	9.40	**1.20**	1.35	8.10	0.90	**0.75**	7.70	0.65	**0.35**	6.65	4.00	**0.40**
150	94.30	5.55	**3.35**	80.35	3.65	**2.85**	56.60	**2.20**	2.40	84.25	6.95	**2.80**
200	277.09(9)	15.05	**9.90**	201.64(6)	11.00	**8.50**	249.35(3)	6.00	**4.55**	242.00(5)	**4.95**	5.15
250	296.00(13)	**18.45**	19.55	322.78(11)	**15.30**	**15.30**	390.00(15)	11.10	**10.65**	412.00(13)	**3.25**	10.70
300	472.00(19)	42.05	**41.15**	575.00(19)	**33.05**	37.95	600.00(20)	26.40	**25.55**	600.00(20)	**4.60**	25.20

Table 3.3 Average computational time required by the different network flow formulations on non planar instances (benchmark `R`).

n	MCF	SCF	SCF2
50	**0.3**	1.9	2.8
60	0.9	**0.00**	7.9
70	**0.00**	1	3.5
80	**0.00**	119.6	120.8
90	**0.00**	523.10	193.60
100	**0.00**	465.30	557.30
150	**0.00**	899.40	1135.60
200	**0.00**	826.50	854.00
250	**0.00**	554.50	626.20
300	**0.00**	670.50	809.10

Table 3.4 Average number of branching nodes required to solve instances in benchmark P having $\delta = 0.8$ using the different network flow formulations.

root vertex using that formulation. On the contrary, if we exclude the smallest instances that can be solved generating less than ten branching nodes, using both SCF and SCF2, CPLEX generates hundreds of branching nodes. This different behaviors show that, as expected, the LP-Relaxation of formulation MCF is tighter than the LP-Relaxation of single-commodity formulations. However, the strength of formulation MCF is not sufficient to balance the higher computational resources required to handle the higher number of variables and constraints w.r.t. the single-commodity formulations. Note that we do not report the number of branching nodes generated to solve the other instances only for the sake of brevity, however the same behavior can be observed solving all the instances.

Regarding the different performance of the two single-commodity formulations, as expected, the optimization method based on formulation SCF2 outperforms the optimization method based on formulation SCF. However, the differences between these two formulations tend to vanish when we increase the δ value and, consequently, when we weaken the knapsack constraint. In figures 3.1-3.4 we plot the average computational time required by CPLEX to solve both SCF and SCF2 on instances in benchmark D varying $\delta \in \{0.2, 0.4, 0.6, 0.8\}$: when $\delta = 0.8$ the two plot are almost indistinguishable. This behavior can be easily explained considering the fact that the advantages of SCF2 w.r.t. SCF are due to the implicit handling of the knapsack constraint: increasing δ the knapsack constraint becomes weaker and the differences between the two formulations vanish.

3.7.3 The best way to initialize the connectivity cuts pool

In the second experiment, we compare the two different strategies proposed in Section 3.5.2 to initialize the connectivity cuts pool to solve formulation CTF (3.4). In Table 3.5, Table 3.6 and Table 3.7 we report the average computational time required to solve formulation CTF (3.4), respectively, for instances in benchmark D, benchmark R and benchmark P. For each number of vertices and each value of δ,

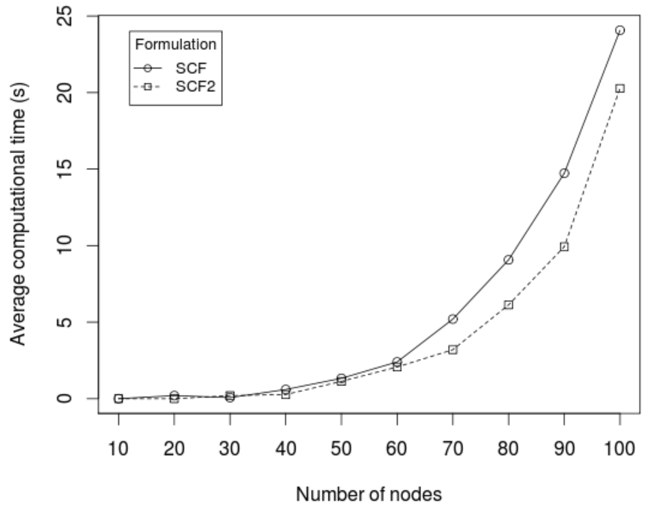


Fig. 3.1 Comparison of the average computational time required by CPLEX to solve instances in benchmark D using SCF and SCF2, setting $\delta = 0.2$.

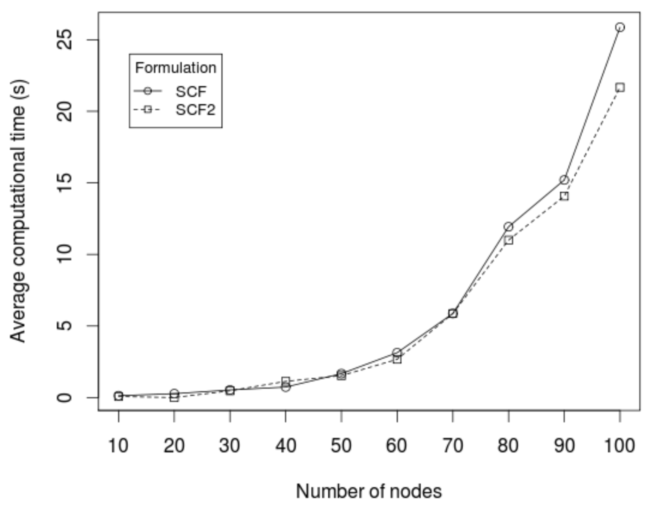


Fig. 3.2 Comparison of the average computational time required by CPLEX to solve instances in benchmark D using SCF and SCF2, setting $\delta = 0.4$.

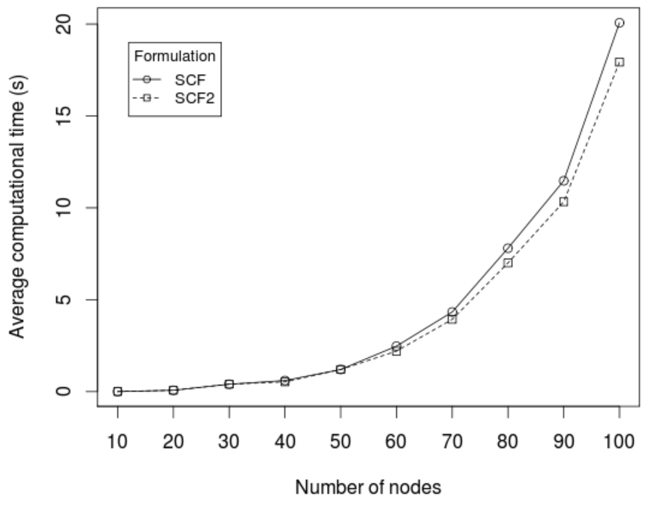


Fig. 3.3 Comparison of the average computational time required by CPLEX to solve instances in benchmark D using SCF and SCF2, setting $\delta = 0.6$.

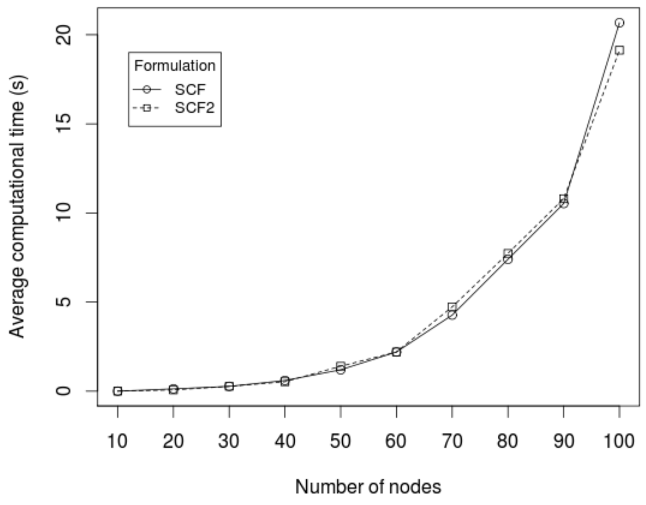


Fig. 3.4 Comparison of the average computational time required by CPLEX to solve instances in benchmark D using SCF and SCF2, setting $\delta = 0.8$.

we report the average computational time in seconds obtained both by initializing the connectivity cuts pool with the 2-Cycle removal constraints (3.5) and by not initializing it at all. On each instance we impose a time limit of 600 seconds and, if some of the instances in a given class cannot be solved within the time limit, the number of unsolved instances is reported within parenthesis.

	$\delta = 0.2$		$\delta = 0.4$		$\delta = 0.6$		$\delta = 0.8$	
n	(3.5)	Empty Pool	(3.5)	Empty Pool	(3.5)	Empty Pool	(3.5)	Empty Pool
10	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	0.13	**0.01**
20	**0.01**	**0.01**	**0.01**	0.07	**0.01**	0.07	0.07	**0.01**
30	**0.07**	0.27	**0.07**	0.13	**0.01**	0.13	0.00	**0.01**
40	**0.01**	0.27	**0.13**	0.47	**0.07**	0.33	**0.01**	0.33
50	**0.20**	0.87	**0.13**	1.27	**0.20**	1.13	**0.07**	0.87
60	**0.20**	2.60	**0.40**	3.33	**0.13**	3.07	**0.20**	2.33
70	**0.47**	5.60	**0.60**	7.73	**0.53**	7.07	**0.27**	4.47
80	**0.67**	11.27	**0.40**	17.07	**0.27**	14.07	**0.80**	10.33
90	**1.00**	23.93	**0.93**	29.53	**0.73**	23.40	**0.67**	15.60
100	**1.40**	47.33	**1.80**	48.60	**1.40**	39.33	**1.53**	29.47

Table 3.5 Average computational time required to solve formulation CTF (3.4) on dense instances (benchmark D), initializing the connectivity cuts pool with the 2-Cycle removal constraints (3.5) or with the empty pool.

	$\delta = 0.2$		$\delta = 0.4$		$\delta = 0.6$		$\delta = 0.8$	
n	(3.5)	Empty Pool	(3.5)	Empty Pool	(3.5)	Empty Pool	(3.5)	Empty Pool
50	0.40	**0.30**	**0.30**	0.40	**0.10**	0.40	**0.10**	0.30
60	0.30	**0.10**	**0.20**	0.70	**0.10**	0.40	**0.20**	0.20
70	**0.40**	0.80	**0.70**	0.80	**0.20**	0.60	**0.10**	0.70
80	**0.40**	0.60	**0.50**	1.50	**0.30**	1.10	**0.20**	0.50
90	**0.70**	2.10	**0.90**	2.70	**0.50**	1.60	**0.10**	0.90
100	**0.50**	1.80	**1.20**	2.00	**0.70**	3.30	**0.40**	1.20
150	**5.60**	48.00	**5.90**	67.90	**2.00**	11.10	**1.20**	5.90
200	**29.70**	180.78(1)	**20.30**	207.44(1)	**6.70**	170.90	**3.10**	20.30
250	**75.50**	280.33(7)	**40.40**	321.67(4)	**9.60**	88.30	**7.40**	40.40
300	**76.40**	384.50(8)	**150.40**	450.00(9)	**31.30**	176.00(4)	**17.60**	100.44(1)

Table 3.6 Average computational time required to solve formulation CTF (3.4) on planar instances (benchmark P), initializing the connectivity cuts pool with the 2-Cycle removal constraints (3.5) or with the empty pool.

Analyzing these results, we can observe that the initialization strategy based on the 2-Cycle removal constraints (3.5) outperforms the initialization strategy that does not use them: using the latter initialization strategy, we cannot solve, within the time limit, 35 instances in benchmark P and 15 instances in benchmark R. On all other instances, by initializing the connectivity cuts pool with the 2-Cycle re-

	$\delta = 0.2$		$\delta = 0.4$		$\delta = 0.6$		$\delta = 0.8$	
n	(3.5)	Empty Pool	(3.5)	Empty Pool	(3.5)	Empty Pool	(3.5)	Empty Pool
50	**0.01**	0.05	**0.01**	0.10	**0.01**	0.20	**0.01**	0.00
60	**0.10**	0.30	**0.01**	0.15	**0.01**	0.20	**0.01**	0.10
70	**0.10**	0.25	**0.05**	0.45	**0.01**	0.20	**0.01**	0.25
80	**0.05**	0.35	**0.01**	0.50	**0.01**	0.35	**0.01**	0.30
90	**0.20**	0.45	**0.01**	0.80	**0.01**	0.85	**0.10**	0.55
100	**0.05**	1.05	**0.10**	1.20	**0.10**	1.05	**0.05**	0.50
150	**0.35**	9.05	**0.35**	9.10	**0.50**	7.85	**0.50**	4.05
200	**1.70**	30.67(2)	**1.70**	44.70	**1.65**	32.25	**1.85**	19.00
250	**3.85**	86.21(1)	**2.75**	100.75	**5.15**	79.85	**5.65**	51.30
300	**5.60**	196.00(4)	**5.75**	244.22(3)	**11.75**	189.06(3)	**22.35**	173.50(2)

Table 3.7 Average computational time required to solve formulation CTF (3.4) on non planar instances (benchmark R), initializing the connectivity cuts pool with the 2-Cycle removal constraints (3.5) or with the empty pool.

moval constraints (3.5) we can significantly reduce the required computational time. This behavior can be explained by considering the high amount of computational resources saved by adding only once all the 2-Cycle removal constraints (3.5) w.r.t. the computational burden of the continuous generation of the same cuts in the different branching nodes.

3.7.4 Comparison of the different exact methods

In the last computational experiment, in order to finally understand which one of the proposed methods is the most efficient in solving the KPCSTP, we compare the three following methods:

- SCF2: formulation (3.2) directly solved by CPLEX.
- CTF: formulation (3.4) solved by CPLEX with the dynamic generation of the connectivity cuts constraints and initializing the connectivity cuts pool with the 2-Cycle removal constraints (3.5).
- SST: formulation (3.6) solved using the Relax-and-Cut method described in Section 3.6.

We focus our attention on these three methods since in Section 3.7.2 we have shown that formulation SCF2 (3.2) outperforms the two others network flow formulations (i.e. SCF and MCF), and in Section 3.7.3 we have shown that, by initializing the connectivity cuts pool with the 2-Cycle removal constraints, we significantly decrease the required computational time w.r.t. not initializing it at all.

Thus, in Table 3.8, Table 3.9 and Table 3.10 we report the average computational time in seconds required to solve the KPCSTP using these three optimization methods, respectively, on benchmark D, benchmark P and benchmark R. Similarly to what done in the previous experiments, we impose a 600 seconds time limit. If some instances in a given class cannot be solved within the time limit, we report in parenthesis the number of unsolved instances.

	$\delta = 0.2$			$\delta = 0.4$			$\delta = 0.6$			$\delta = 0.8$		
n	SCF2	SST	CTF	SCF2	SST	CTF	SCF2	SST	CTF	SCF2	SST	CTF
10	**0.01**	**0.01**	**0.01**	0.07	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	0.13
20	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	**0.01**	0.07	**0.01**	**0.01**	0.07	**0.01**	0.07
30	0.20	**0.01**	0.07	0.47	**0.01**	0.07	0.40	**0.01**	**0.01**	0.27	**0.01**	**0.01**
40	0.27	0.04	**0.01**	1.13	**0.03**	0.13	0.53	0.02	0.07	0.53	**0.01**	**0.01**
50	1.13	**0.01**	0.20	1.53	**0.01**	0.13	1.20	**0.01**	0.20	1.40	**0.01**	0.07
60	2.07	**0.04**	0.20	2.67	**0.01**	0.40	2.20	**0.01**	0.13	2.20	0.02	0.20
70	3.20	**0.03**	0.47	5.87	**0.06**	0.60	3.93	**0.01**	0.53	4.73	**0.01**	0.27
80	6.13	**0.02**	0.67	11.00	**0.03**	0.40	7.00	0.03	0.27	7.73	0.02	0.80
90	9.93	**0.04**	1.00	14.07	**0.72**	0.93	10.33	0.03	0.73	10.80	**0.01**	0.67
100	20.27	**0.11**	1.40	21.67	**0.03**	1.80	17.93	0.11	1.40	19.13	**0.28**	1.53

Table 3.8 Average computational time required to solve dense instances (benchmark D) using formulation SCF2 (3.2), formulation CTF (3.4) and formulation SST (3.6).

	$\delta = 0.2$			$\delta = 0.4$			$\delta = 0.6$			$\delta = 0.8$		
n	SCF2	SST	CTF	SCF2	SST	CTF	SCF2	SST	CTF	SCF2	SST	CTF
50	**0.10**	0.98	0.40	0.60	2.29	**0.20**	0.30	0.82	**0.10**	**0.10**	23.69	**0.10**
60	0.20	**0.14**	0.30	0.70	**0.06**	0.30	0.10	**0.04**	0.10	0.30	39.74(1)	**0.20**
70	0.90	1.12	**0.40**	1.20	0.64	**0.50**	0.60	**0.12**	0.20	0.40	21.58	**0.10**
80	0.70	2.70	**0.40**	0.90	0.54	**0.50**	1.00	0.99	**0.30**	0.90	1.13(1)	**0.20**
90	2.40	14.29	**0.70**	3.70	2.44	**0.90**	0.90	0.69	**0.50**	1.00	24.57	**0.10**
100	1.90	23.90	**0.50**	1.10	0.48	**0.40**	0.80	**0.51**	0.70	1.70	**0.40**	**0.40**
150	8.10	76.44(3)	**5.60**	13.40	33.00	**2.90**	5.60	47.58	**2.00**	6.00	10.06	**1.20**
200	**27.10**	55.91(8)	29.70	19.60	73.18(4)	**9.50**	13.10	69.14(1)	**6.70**	8.20	23.50	**3.10**
250	**54.70**	525.04(9)	75.50	26.90	7.11(8)	**25.80**	12.80	48.66(2)	**9.60**	10.40	49.37	**7.40**
300	**50.20**	600.00(10)	76.40	62.60	484.00(9)	**58.00**	**18.30**	229.22(5)	31.30	**17.30**	173.66(3)	17.60

Table 3.9 Average computational time required to solve planar instances (benchmark P) using formulation SCF2 (3.2), formulation CTF (3.4) and formulation SST (3.6).

Analyzing these results, we can immediately note that the performances of the different optimization methods change drastically if we consider dense instances (benchmark D), or sparse instances (benchmark P and benchmark R). In particular, SST is very effective on dense instances and in many cases it outperforms the other two competing methods. However, if we consider the sparse instances, SST is not able to solve many instances within the time limit (for example, it is not able to solve any instance in benchmark P having $n = 300$). CTF is a more robust method w.r.t. SST: on many instance classes in benchmark R and benchmark P it outperforms the two other methods, while, if we consider benchmark D, it is competitive with SST and outperforms SCF2 on most instances. Finally, it is interesting to note that, de-

	$\delta = 0.2$			$\delta = 0.4$			$\delta = 0.6$			$\delta = 0.8$		
n	SCF2	SST	CTF	SCF2	SST	CTF	SCF2	SST	CTF	SCF2	SST	CTF
50	0.15	0.02	**0.01**	0.40	0.02	**0.01**	0.20	**0.01**	**0.01**	0.10	**0.01**	**0.01**
60	0.15	0.16	0.10	0.45	0.16	**0.01**	0.15	0.84	**0.01**	0.35	0.64	**0.01**
70	0.20	**0.05**	0.10	0.55	0.10	**0.05**	0.55	0.29	**0.01**	0.40	0.24	**0.01**
80	0.60	**0.04**	0.05	0.40	0.04	**0.01**	0.50	0.04	**0.01**	0.40	0.02	**0.01**
90	0.95	**0.03**	0.20	0.60	0.07	**0.01**	0.55	0.13	**0.01**	0.45	**0.08**	0.10
100	1.35	0.77	**0.05**	0.75	0.30	**0.10**	0.35	**0.06**	0.10	0.40	**0.05**	**0.05**
150	3.35	1.75	**0.35**	2.85	9.24	**0.35**	2.40	2.50	**0.50**	2.80	2.16	**0.50**
200	9.90	23.03(2)	**1.70**	8.50	16.96	**1.70**	4.55	1.77	**1.65**	5.15	**1.29**	1.85
250	19.55	10.76(3)	**3.85**	15.30	10.06	**2.75**	10.65	**1.83**	5.15	10.70	**2.59**	5.65
300	41.15	75.50(4)	**5.60**	37.95	12.98	**5.75**	25.55	42.90(2)	**11.75**	25.20	49.59(3)	**22.35**

Table 3.10 Average computational time required to solve non planar instances (benchmark R) using formulation SCF2 (3.2), formulation CTF (3.4) and formulation SST (3.6).

spite its simplicity, SCF2 has often performances that are comparable with the other more complex algorithms and outperforms the SST method on sparse instances.

3.8 Conclusion

In this chapter we investigated the different approaches that can be exploited to solve the KPCSTP. The computational results reported in Section 3.7 showed that:

- The ILP extended formulation based on the connectivity cuts, on average, outperforms the other proposed methods.
- The Lagrangian Relaxation method is competitive with the previous one on dense instances, but on sparse instances it is outperformed also by the methods based on the single-commodity network flow formulations.
- The the single-commodity network formulation that implicitly handles the knapsack constraint using a particular flow definition, despite its simplicity, is competitive with other more complex methods.
- Despite the tightness of its LP-Relaxation, the multi-commodity flow formulation cannot be effectively used to solve the KPCSTP. The time required to process a single branching node is too high.

These conclusions can be used to develop effective methods for combinatorial optimization problems that are similar to the KPCSTP. However, in different situations the investigated methods may show different behaviors. As example, in Chapter 4 we show that the method based on the multi-commodity formulation outperforms the method based on the connectivity cuts when solving the Pricing Problem associated with the extended formulation that we developed to solve the MRWADC problem. These contradictory results are deeply discussed in Section 4.5.

3.9 References

T. Achterberg and T. Berthold. Improving the feasibility pump. *Discrete Optimization*, 4(1):77 – 86, 2007.

J. E. Beasley. An sst-based algorithm for the steiner problem in graphs. *Networks*, 19(1):1–16, 1989.

D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical programming*, 59(3): 413–420, 1993.

P. M. Camerini, L. Fratta, and F. Maffioli. On improving relaxation methods by modified gradient techniques. In M. L. Balinski and P. Wolfe, editors, *Nondifferentiable Optimization*, volume 3 of *Mathematical Programming Studies*, pages 26–34. Springer, 1975.

R. Cordone and M. Trubian. A Relax-and-Cut Algorithm for the Knapsack Node Weighted Steiner Tree Problem. *Asia-Pacific Journal of Operational Research*, 25(3):373–391, 2008.

E. Danna, E. Rothberg, and C. L. Pape. Exploring relaxation induced neighborhoods to improve mip solutions. *Mathematical Programming*, 102(1):71–90, 2005.

C. W. Duin and A. Volgenant. Some generalizations of the steiner problem in graphs. *Networks*, 17(3):353–364, 1987.

S. Engevall, M. Göthe-Lundgren, and P. Värbrand. A strong lower bound for the node weighted steiner tree problem. *Networks*, 31(1):11–17, 1998.

A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, October 1988.

M. Haouari, S. Layeb, and H. Sherali. The prize collecting steiner tree problem: models and lagrangian dual optimization approaches. *Computational Optimization and Applications*, 40(1):13–39, 2008.

M. Held and R. M. Karp. The traveling-salesman problem and minimum spanning trees. *Operations Research*, 18(6):1138–1162, 1970.

J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

E. Lawler. *Combinatorial Optimizations: Networks and Matroids*. Holt, Rinehar and Winston, 1976.

I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem. *Mathematical programming*, 105(2-3):427–449, 2006.

A. Lodi. The heuristic (dark) side of mip solvers. In E. Talbi, editor, *Hybrid Metaheuristics*, volume 434 of *Studies in Computational Intelligence*, pages 273–284. Springer, 2013.

A. Lucena and M. Resende. Strong lower bounds for the prize collecting steiner problem in graphs. *Discrete Applied Mathematics*, 141(1–3):277 – 294, 2004.

T. L. Magnanti and L. A. Wolsey. Optimal trees. In C. L. Monma M. O. Ball, T. L. Magnanti and G. L. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503 – 615. Elsevier, 1995.

E. Rothberg. An evolutionary algorithm for polishing mixed integer programming solutions. *INFORMS Journal on Computing*, 19(4):534–541, 2007.

A. Segev. The node-weighted steiner tree problem. *Networks*, 17(1):1–17, 1987.

Chapter 4
Multicast Routing and Wavelength Assignment with Delay Constraint in WDM networks with heterogeneous capabilities

4.1 Introduction

A great number of people worldwide use intensively computer networks, and, in the last few years, new bandwidth-intensive computer networks applications have been developed: consider, for example, video conferences, video streaming, VOIP telephony, web applications and cloud computing services (Mukherjee, 2006; Tzanakaki et al., 2011). Consequently, an acute need for very high-bandwidth network infrastructures emerges. To meet this need, backbone networks based on optical fibers have been built: a single mode optical fiber has a potential bandwidth of nearly 50 terabits per second, that is much higher than the bandwidth supported by electronic equipments (few tens of gigabits per second today). Hence a bandwidth mismatch arises on networks which have optical technologies on links and electronic devices on nodes. This bandwidth mismatch can be exploited by Wavelength Division Multiplexing (WDM) optical networks (see, for example, Mukherjee 2000, 2006 for an introduction on this technology) to enable parallel transmissions. A WDM optical network consists of a set of optical fibers, the optical links, connected by switch nodes mainly equipped with electronic devices. Multiple wavelengths are available in each fiber, which provide a large transmission bandwidth. Each different wavelength corresponds to a different *communication channel*.

Consider now a piece of data that needs to go from a source node s to a destination node t in a WDM network: if the optical signal carrying the data has different wavelengths (i.e. it uses different communication channels) on different optical network links, in the intermediate switch nodes we need to convert the optical signal using an electronic device. This conversion process can be very slow w.r.t. the speed of optical communications and, consequently, routing data in a WDM network using a standard packet switching strategy brings to a waste of the optical links bandwidth.

Hence, the best way to send data from a source node s to a destination node t in a WDM network is to use the same wavelength (i.e. the same communication channel) on all the optical links crossed by the optical signal carrying the data. If we model the optical network as a directed graph, we need to find a directed path connecting s to t such that we can assign the same wavelength to all the links crossed by the path.

A single path from s to t corresponds to a *unicast* or point-to-point transmission. *Multicast transmissions* provide a means of point-to-multipoint communication. In the multicast transmission setting we need to connect a single source node s with a set of destinations nodes D. This requires the capability of the switch nodes to reproduce the data received in input on different output links. Multicast transmissions have several data networks applications including video-conferencing, distributed computing, Internet TV, stock prices updates, software distribution, and so on. A single stream is shared by a batch of common recipients/users, which makes this system more advantageous.

Extending the idea introduced above for the unicast transmission setting, to efficiently use WDM networks in a multicast setting, given the directed graph modeling the network, we need to find an arborescence (resp. set of arborescences), rooted in s and covering all the nodes in D, in which, for all the links defining the arborescence (resp. each arborescence), we can use the same wavelength. The problem of finding a multicast arborescence and assigning a suitable wavelength to it is known as the Multicast Routing and Wavelength Assignment (MCRWA) problem (Chen and Wang, 2002). It plays a key role in supporting multicasting over WDM networks.

In this work we address the Multicast Routing Wavelength Assignment with Delay Constraints (MRWADC) problem on heterogeneous WDM networks (Chen et al., 2008). With respect to the MCRWA problem three more features arising in real world applications are considered:

1. *Delay bound constraints:* the time required to carry a signal from the source to each destination node along directed paths is limited by a threshold value, say Δ (useful to guarantee the Quality-of-Service of the transmissions).
2. *Heterogeneous multicast capabilities*: for budget constraints, in a real WDM network not all the nodes are equipped with the same facilities (i.e. the considered network is heterogeneous) and, consequently, the capability to reproduce the data received in input on different output links varies from node to node.
3. *Network congestion*: certain wavelengths on certain network links are unavailable.

In order to choose between feasible arborescence sets, we associate with each optical link a cost that we need to pay each time we use one of its wavelengths. The objective function of the problem requires to minimize a convex combination of the sum of these costs (*communication cost*) and the number of used arborescences (*wavelength consumption*).

The MRWADC problem on heterogeneous WDM networks, has been already studied in Chen et al. (2008). In this paper the authors propose a compact ILP formulation and a constructive heuristic that uses the minimum cost paths in the network. In Chen and Tseng (2005) the authors propose a genetic algorithm to solve the MRWADC problem without considering network congestion. Other related problems have been studied in literature. In Zhu et al. (1995) the authors describe a local search heuristic that allows to make efficient multicast transmission in a non-optical network. In Zhang et al. (2000) the authors study how to efficiently manage WDM networks with sparse splitting capabilities (i.e. only few nodes in the network have splitting capabilities) in a multicast setting. The authors of Sreenath et al. 2001 investigate the efficient use of a WDM network containing some special nodes called virtual sources. In Jia et al. (2001) the authors develop optimization algorithms to satisfy delay bound constraints in WDM network having unlimited splitting capabilities. Finally, the problem studied in Yan et al. (2001) requires to minimize the wavelengths used by a WDM network with sparse splitting capabilities in a multicast setting.

In this work, we introduce a new ILP formulation which allows a simple Dantzig-Wolfe decomposition and leads to an extended formulation amenable to a Column Generation approach. The related Pricing Problem is very similar to the KPCSTP introduced in Chapter 3. Thus, here we derives two different exact methods for the Pricing Problem starting from, respectively, the multi-commodity flow formulation (see Section 3.3) and the connectivity cuts formulation (see Section 3.5) for the KPCSTP. In Chen et al. (2008), the only previous approach in literature to solve the MRWADC problem, the authors say that "*The elapsed execution time is more than 34 h [...] for some request (q = 3) in the network with 60 nodes (n = 60), and the ILP formulation cannot be used to solve the network with more than 70 nodes or requests with more than 3 destinations in a reasonable time.*". Using the method described in this chapter, we obtain optimal solutions for instances having up to one hundred nodes within two hours of computation time. Moreover, to solve instances within the same class, our algorithm requires far less computation time w.r.t. the method proposed in Chen et al. (2008). Notice that the computational experiments reported in Chen et al. (2008) had been executed using an older version of the commercial ILP solver and a slower computer w.r.t. those we have used in our tests. Hence, to fairly compare our approach with that based on solving a compact ILP formulation, we tested the latter method on our machine and using our version of the ILP solver. The obtained results (see, Section 4.6.4) confirm that a compact formulation can be used with success on instances up to 40 nodes, but it fails on an increasing number of instances as n increases beyond that threshold.

In the next section we formally introduce the MRWADC problem and we present our new ILP compact formulation. In Section 4.4, by applying the Dantzig-Wolfe decomposition technique to the compact formulation, we introduce an extended formulation and we describe how to solve its LP-Relaxation using a Column Generation method. In Section 4.5 we describe the two exact methods and the Tabu Search heuristic we developed to solve the Pricing Problem. In Section 4.6 we report some implementation issues and present the experimental results we obtained. The last section reports our concluding remarks. Finally, note that the whole content of this chapter derives from our previously published work (Colombo and Trubian, 2013).

4.2 The MRWADC problem and its compact ILP formulation

To formally describe the MRWADC problem we need to introduce some WDM network terminology.

Definition 1. A **WDM network** is a tuple $\mathscr{N} = (G, M, c, t, \theta, w)$, where:

- $G = (V, A)$ ($| V | = n$ and $| A | = m$) is a directed graph which defines the underlying network structure. The nodes in V are the switch nodes and the arcs in A are the optical links. There are multiple wavelengths (communication channels) available in each optical link.
- M ($| M | = k$) is the discrete set of the wavelengths available in each optical link.
- The function $c : A \to \mathbb{R}^+$ (resp. $t : A \to \mathbb{R}^+$) determines the cost (resp. time) associated with each arc.
- The function $\theta : V \to \mathbb{N}$ determines the maximum number of reproductions that a network node can generate (i.e. it determines the splitting capabilities of the nodes, as explained below).
- The function $w : A \times M \to \{0, 1\}$ models the network congestion and associates with each communication channel $(a \in A, \lambda \in M)$ the value 0 (resp. 1) if the communication channel is busy (resp. free)

A WDM network needs nodes capable to reproduce the received input on many output links:

Definition 2. A ***multicast incapable** (MI)* node is a node of a WDM network that is not able to reproduce the data it received in input on more than one output link. On the contrary, a ***multicast capable** (MC)* node is a node of a WDM network that is able to reproduce the data on more than one output link. The ***splitting capability*** of a node is the maximum number of reproductions of the received data that a node can generate.

The WDM networks studied in this chapter are heterogeneous since not all the network nodes have the same multicast capabilities: only a subset of the nodes are MC and, generally, these nodes have different splitting capabilities.
In the previous section we introduced the requirements of an efficient unicast transmission on WDM networks. These requirements have been formalized by the light-path concept (Chlamtac et al., 1992):

Definition 3. Given a WDM network, a ***light-path*** is a path in G which uses the same wavelength on all the crossed optical links.

In the multicast setting, the light-path concept has been extended to the light-tree concept (Sahasrabuddhe and Mukherjee, 1999):

Definition 4. Given a WDM network, a ***light-tree*** is an arborescence, say T, defined on G, which uses the same wavelength, say λ, on all the crossed optical links. We will denote a light-tree with the corresponding couple (T, λ).

Observation. Note that, since the underlying network structure of the problem studied in this chapter is a directed graph, we need to find arborescences, not trees. Despite this, to conform with the previous literature on this problem we continue to use the somewhat misleading optical network terminology.

Given the delay bound constraints, the heterogeneous multicast capabilities and the network congestion, a single light-tree could not span all the destinations of a multicast transmission request respecting all the constraints. In these situations, we need to find different light-trees on different wavelengths, spanning different destination subsets; we need a light-forest (Zhang et al., 2000):

Definition 5. Given a WDM network, a ***light-forest*** is a set of light-trees defined on G and denoted by $\{(T_1,\lambda_1),(T_2,\lambda_2),\ldots,(T_p,\lambda_p)\}$ such that each light-tree in this set uses a different wavelength (i.e. if $i \neq j$ then $\lambda_i \neq \lambda_j$).

The input data of the problem is defined by a WDM network and by a transmission request:

Definition 6. A **transmission request** is a 3-tuple $\mathscr{R} = (s,D,\Delta)$, where $s \in V$ is the transmission source, $D \subseteq V \setminus \{s\}$ is the destination set (with $\mid D \mid = q$) and $\Delta \in \mathbb{R}^+$ is the delay bound threshold.

For convenience we introduce the following standard notations:

Notation. Given a a directed graph T, $A(T)$ is the set of its arcs, $V(T)$ is the set of its nodes and $D(T) = V(T) \cap D$ is the set of the destinations spanned by T (notice that the definition of $D(T)$ depends on the considered transmission request) . Given a node subset $S \subset V(T)$ we define $\delta_T^-(S) = \{(i,j) \in A(T) | i \notin S \wedge j \in S\}$ and similarly $\delta_T^+(S) = \{(i,j) \in A(T) | i \in S \wedge j \notin S\}$. For sake of simplicity, we write $\delta_T^-(v)$ and $\delta_T^+(v)$ instead of $\delta_T^-(\{v\})$ and $\delta_T^+(\{v\})$. Finally, for each $\lambda \in M$ we define $A^\lambda = \{a \in A \mid w(a,\lambda) = 1\}$ and $G^\lambda = (V, A^\lambda)$.

The feasible solutions of our problem are constrained light-forests:

Definition 7. Given a WDM network $\mathscr{N} = (G,M,c,t,\theta,w)$ and a transmission request $\mathscr{R} = (s,D,\Delta)$, a **feasible light-tree** is a light-tree (T,λ) such that: $\lambda \in M$, T is rooted in s and contained in G^λ, the outdegree of each node $v \in V(T)$ is at most equal to $\theta(v)$ and if $P = (a_1,\ldots,a_l)$, with $a_i \in A^\lambda$ for each $i = 1,\ldots,l$, is the unique path connecting s with $v \in V(T)$, then $\sum_{j=1}^{l} t(a_j) \leq \Delta$.

Definition 8. Given a WDM network $\mathscr{N} = (G,M,c,t,\theta,w)$ and a transmission request $\mathscr{R} = (s,D,\Delta)$, a **feasible light-forest** is a set $\{(T_1,\lambda_1),(T_2,\lambda_2),\ldots,(T_p,\lambda_p)\}$ that satisfies the following constraints: for each $i = 1,2,\ldots,p$, (T_i,λ_i) is a feasible light-tree, and each destination node in D is spanned at least by one light-tree (i.e. $\bigcup_{i=1}^{p} D(T_i) = D$).

Our problem can be now stated as follows:

MRWADC problem:

Input: A WDM network $\mathscr{N} = (G, M, c, t, \theta, w)$, a transmission request $\mathscr{R} = (s, D, \Delta)$ and parameter $\alpha \in [0, 1]$.
Output: A feasible light-forest $\{(T_1, \lambda_1), (T_2, \lambda_2), \ldots, (T_p, \lambda_p)\}$ that minimizes $\alpha \sum_{i=1}^{p} c_{T_i} + (1-\alpha)p$, where, for each $i = 1, \ldots, p$, we define the communication cost of the light-tree (T_i, λ_i) as $c_{T_i} = \sum_{a \in A(T_i)} c(a)$

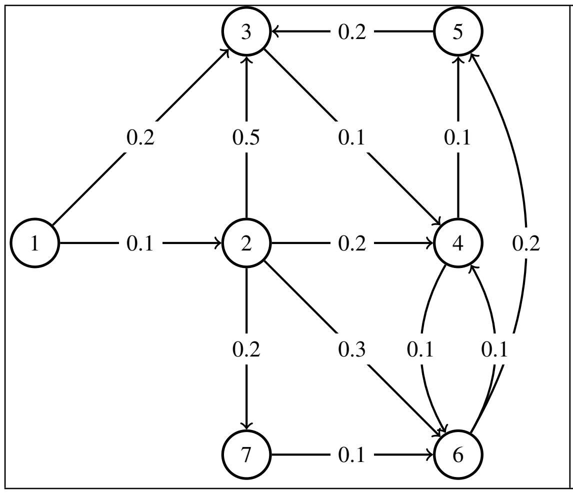
As already done in the literature (Chen et al., 2008), we choose to use a convex combination of the communication cost with the wavelength consumption as objective function. With this choice we want to give to the decision maker the possibility to increment/decrement α in order to face different WDM network states. For example, if there are few free wavelengths the decision maker can decrement α to preserve them. On the other side, if the WDM network has many free wavelengths the decision maker may strictly control the communication cost incrementing α.

Since every instance of the Minimum Steiner Tree Problem can be polynomially transformed into a MRWADC problem instance (see Chen et al. 2008) and the Minimum Steiner Tree Problem is $\mathscr{N}\mathscr{P}$-hard (see, for example, Garey and Johnson 1979), the studied problem is $\mathscr{N}\mathscr{P}$-hard.

Figures 4.1 and 4.2 illustrate, respectively, an example of a MRWADC instance and one of its optimal solutions. In the optimal solution shown in Figure 4.2, two wavelengths are required. Indeed, even if the solution which uses the paths $(1-2-7)$, $(1-2-4-6)$ and $(1-2-4-5-3)$ respects the splitting capabilities of the involved nodes, the last path, requiring 0.6 units of time, violates the delay bound constraint. The same happens to the solution which uses the paths $(1-2-7-6)$ and $(1-2-3)$. Even in this case the last path, requiring 0.6 units of time, violates the delay bound constraint. On the other hand, we cannot merge light-trees T_1 and T_2 since the source node 1 is a multicast incapable node (i.e. $\theta(1) = 1$).
The ILP formulation, to which we will refer in the following as the *compact formulation* (CF) makes use of the following variables:

- $f_a^{\lambda,d}$ is equal to 1 if $a \in A^\lambda$ is used in G^λ to connect s with $d \in D$, 0 otherwise.
- x_a^λ is equal to 1 if $a \in A^\lambda$ is used in G^λ, 0 otherwise; variable z^λ is equal to 1 if at least one of the G^λ arcs is used, 0 otherwise.
- u_d^λ is equal to 1 if $d \in D$ is spanned by the light-tree defined in G^λ, 0 otherwise.

Once defined these variables, we can define CF formulation as follows:



$c(1,2)=2$ $\theta(1)=1$
$c(1,3)=4$ $\theta(2)=2$
$c(2,3)=3$ $\theta(3)=1$
$c(2,4)=1$ $\theta(4)=2$
$c(2,6)=2$ $\theta(5)=1$
$c(2,7)=5$ $\theta(6)=1$
$c(3,4)=4$ $\theta(7)=1$
$c(4,5)=3$ $\Delta=0.5$
$c(4,6)=2$ $\alpha=0.9$
$c(5,3)=2$ $s=1$
$c(6,4)=1$
$c(6,5)=2$
$c(7,6)=3$
$M=\{1,2\}$
$D=\{3,6,7\}$
$w((3,4),1)=0$
$w((4,5),2)=0$

Fig. 4.1 Example of a MRWADC instance. On the left we report the WDM network structure. The arc labels define the function t. On the right we report the other parameters describing the WDM network, the transmission request and the combination coefficient α.

$$\mathrm{CF}: \min\ \alpha \sum_{\lambda\in M}\sum_{a\in A^\lambda} c(a)x_a^\lambda + (1-\alpha)\sum_{\lambda\in M} z^\lambda$$

$$\sum_{a\in\delta^-_{G^\lambda}(s)} f_a^{\lambda,d} - \sum_{a\in\delta^+_{G^\lambda}(s)} f_a^{\lambda,d} = -z^\lambda \quad d\in D, \lambda\in M \tag{4.1a}$$

$$\sum_{a\in\delta^-_{G^\lambda}(d)} f_a^{\lambda,d} - \sum_{a\in\delta^+_{G^\lambda}(d)} f_a^{\lambda,d} = u_d^\lambda \quad d\in D,\ \lambda\in M \tag{4.1b}$$

$$\sum_{a\in\delta^-_{G^\lambda}(v)} f_a^{\lambda,d} - \sum_{a\in\delta^+_{G^\lambda}(v)} f_a^{\lambda,d} = 0 \quad d\in D, v\in V\setminus\{s,d\}, \lambda\in M \tag{4.1c}$$

$$x_a^\lambda \geq f_a^{\lambda,d} \quad d\in D,\ \lambda\in M,\ a\in A^\lambda \tag{4.1d}$$

$$\sum_{a\in\delta^-_{G^\lambda}(v)} x_a^\lambda \leq 1 \quad v\in V,\ \lambda\in M \tag{4.1e}$$

$$\sum_{a\in\delta^+_{G^\lambda}(v)} x_a^\lambda \leq \theta(v) \quad v\in V,\ \lambda\in M \tag{4.1f}$$

$$\sum_{a\in A^\lambda} t(a) f_a^{\lambda,d} \leq \Delta \quad d\in D,\ \lambda\in M \tag{4.1g}$$

$$\sum_{\lambda\in M} u_d^\lambda \geq 1 \quad d\in D \tag{4.1h}$$

$$f_a^{\lambda,d},\ x_a^\lambda,\ z^\lambda,\ u_d^\lambda \in \{0,1\} \quad d\in D,\ \lambda\in M,\ a\in A^\lambda$$

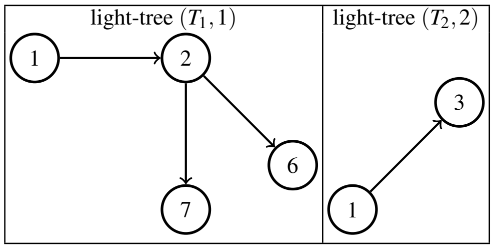


Fig. 4.2 An optimal solution for the instance defined in Figure 4.1. Since $c_{T_1} = 2+5+2 = 9$ and $c_{T_2} = 4$, its cost is equal to $0.9 \times (9+4) + 0.1 \times 2 = 11.9$.

Constraints (4.1a,4.1b,4.1c) determine the used wavelengths, the spanned destinations and ensure the light-trees connectivity. Constraints (4.1d) identify the used links on the different wavelengths. Constraints (4.1e) limit to one the indegree of each node in each light-tree. Constraints (4.1f) ensure that the nodes splitting capabilities are respected. Constraints (4.1g) ensure that the delay bound threshold is respected. Constraints (4.1h) ensure the spanning of all the destinations. This formulation is quite different from the one proposed in Chen et al. (2008), the purpose of this new formulation is to make explicit the diagonal blocks structure of the problem. This formulation is compact since its definition requires a number of variables and constraints that is polynomial in the instance size. In particular, it requires $(nq+2n+q)k + q\sum_{\lambda \in M} \mid A_\lambda \mid + q$ constraints and $(q+1)(k+\sum_{\lambda \in M} \mid A_\lambda \mid)$ variables.

4.3 An extended formulation

Applying the Dantzig-Wolfe decomposition (see Section 2.2.1) to formulation CF (4.1), we can obtain a tighter extended formulation for the MRWADC problem. We start by considering the set $\mathscr{T}^\lambda$ that contains all the feasible light-trees defined on wavelength λ. This set can be defined starting from the polytope defined by constraints (4.1a-4.1g), i.e. all the constraints of formulation CF with the exception of the destination covering constraints (4.1g), for a fixed $\lambda \in M$:

$$
\begin{aligned}
\mathscr{T}^{\lambda}_{LP} = \{\ (\underline{x},\underline{u},z) : & \sum_{a\in\delta^-_{G^\lambda}(s)} f^d_a - \sum_{a\in\delta^+_{G^\lambda}(s)} f^d_a = z \text{ for each } d\in D;\\
& \sum_{a\in\delta^-_{G^\lambda}(d)} f^d_a - \sum_{a\in\delta^+_{G^\lambda}(d)} f^d_a = u_d \text{ for each } d\in D;\\
& \sum_{a\in\delta^-_{G^\lambda}(v)} f^d_a - \sum_{a\in\delta^+_{G^\lambda}(v)} f^d_a = 0 \text{ for each } d\in D, v\in V\setminus\{s,d\};\\
& x_a \geq f^d_a \text{ for each } d\in D, a\in A^\lambda; \quad \sum_{a\in\delta^+_{G^\lambda}(v)} x_a \leq \theta(v) \text{ for each } v\in V;\\
& \sum_{a\in A^\lambda} t(a) f^d_a \leq \Delta \text{ for each } d\in D; f^d_a, x_a, u_d, z \in [0,1] \text{ for each } d\in D, a\in A^\lambda\ \}
\end{aligned}
$$

Using this polytope we can rewrite the LP-Relaxation of formulation CF as follows:

$$
\begin{aligned}
\mathrm{CF_{LP}} : \min\ \alpha \sum_{\lambda\in M}\sum_{a\in A^\lambda} c(a) x^\lambda_a + (1-\alpha)\sum_{\lambda\in M} z^\lambda & \qquad (4.2a)\\
\sum_{\lambda\in M} u^\lambda_d \geq 1 \quad d\in D & \qquad (4.2b)\\
(\underline{x}^\lambda, \underline{u}^\lambda, z^\lambda) \in \mathscr{T}^\lambda_{LP} \quad \lambda\in M & \qquad (4.2c)
\end{aligned}
$$

However, since the MRWADC solution must be integral we can obtain a tighter LP-Relaxation by substituting $\mathscr{T}^\lambda_{LP}$ with the convex hull of its integral points $conv(\mathscr{T}^\lambda_0)$ where the set $\mathscr{T}^\lambda_0$ contains all the elements of $\mathscr{T}^\lambda$ plus the empty solution:

$$
\begin{aligned}
\mathrm{ECF_{LP}} : \min\ \alpha \sum_{\lambda\in M}\sum_{a\in A^\lambda} c(a) x^\lambda_a + (1-\alpha)\sum_{\lambda\in M} z^\lambda & \qquad (4.3a)\\
\sum_{\lambda\in M} u^\lambda_d \geq 1 \quad d\in D & \qquad (4.3b)\\
(\underline{x}^\lambda, \underline{u}^\lambda, z^\lambda) \in conv(\mathscr{T}^\lambda_0) \quad \lambda\in M & \qquad (4.3c)
\end{aligned}
$$

Note that if we set $D = V$, $\theta(v) = n$ for each $v\in V$ and $t(a) = \infty$ for each $a\in A^\lambda$, the set $\mathscr{T}^\lambda$ contains all the feasible solutions for the PCSTP (see Chapter 3). Thus, since the PCSTP is $\mathscr{N}\mathscr{P}$-hard, $\mathscr{T}^\lambda_{LP}$ does not have the integrality property, unless $\mathscr{P} = \mathscr{N}\mathscr{P}$. As a consequence, if we exclude some particular instances of the MRWADC problem, we have $conv(\mathscr{T}^\lambda_0) \neq \mathscr{T}^\lambda_{LP}$ and ECF is, in general, tighter than CF.

The extreme points of $conv(\mathscr{T}^\lambda_{LP})$ coincide with the feasible light-trees in $\mathscr{T}^\lambda$ thus we can associate a tuple $(\overline{\underline{x}}^\lambda_T, \overline{\underline{u}}^\lambda_T, \overline{z}^\lambda_T)$ to each feasible light-tree $T\in\mathscr{T}^\lambda$ and we can express $conv(T^\lambda_0)$ as follows:

$$
conv(\mathscr{T}^\lambda_0) = \left\{ (\underline{x},\underline{u},z) : (\underline{x},\underline{u},z) = \sum_{T\in\mathscr{T}^\lambda} \gamma^\lambda_T (\overline{\underline{x}}^\lambda_T, \overline{\underline{u}}^\lambda_T, \overline{z}^\lambda_T),\ \sum_{T\in\mathscr{T}^\lambda} \gamma^\lambda_T \leq 1,\ \gamma^\lambda_T \geq 0 \right\}
$$

Note that when we define the value of the combination coefficients we use the inequality $\sum_{T\in\mathscr{T}^\lambda}\gamma_T^\lambda \leq 1$ instead of the standard equality $\sum_{T\in\mathscr{T}^\lambda}\gamma_T^\lambda = 1$, since $\mathscr{T}_0^\lambda$ contains the empty solution while $\mathscr{T}^\lambda$ does not. Introducing this new definition of $conv(\mathscr{T}^\lambda)$ and defining the cost of each feasible light-tree $T \in \mathscr{T}^\lambda$ as $c_T = \sum_{a\in A^\lambda} c(a)\bar{x}_{a,j}^\lambda$, we can rewrite formulation (5.22) using the γ coefficients as variables:

$$\mathrm{ECF_{LP}} : \min \sum_{\lambda\in M}\sum_{T\in\mathscr{T}^\lambda}(\alpha c_T + (1-\alpha))\gamma_T^\lambda \tag{4.4a}$$

$$\sum_{\lambda\in M}\sum_{T\in\mathscr{T}^\lambda}\bar{u}_{d,T}^\lambda\gamma_T^\lambda \geq 1 \quad d\in D \tag{4.4b}$$

$$\sum_{T\in\mathscr{T}^\lambda}\gamma_T^\lambda \leq 1 \quad \lambda\in M \tag{4.4c}$$

$$\gamma_T^\lambda \geq 0 \quad \lambda\in M, T\in\mathscr{T}^\lambda \tag{4.4d}$$

4.4 Column Generation

By adding the integrality constraint to formulation ECF$_{\mathrm{LP}}$, we obtain the following extended formulation for the MRWADC problem:

$$\mathrm{EF} : \min \sum_{\lambda\in M}\sum_{T\in\mathscr{T}^\lambda}(\alpha c_T + (1-\alpha))\gamma_T^\lambda$$

$$\sum_{\lambda\in M}\sum_{T\in\mathscr{T}^\lambda}\bar{u}_{d,T}^\lambda\gamma_T^\lambda \geq 1 \quad d\in D \tag{4.5a}$$

$$\sum_{T\in\mathscr{T}^\lambda}\gamma_T^\lambda \leq 1 \quad \lambda\in M \tag{4.5b}$$

$$\gamma_T^\lambda \in \{0,1\} \quad \lambda\in M,\ T\in\mathscr{T}^\lambda$$

Remember that coefficient $\bar{u}_{d,T}^\lambda$ is equal to 1 if the light-tree T on wavelength λ spans the destination d (note that this coefficient is linked with the u_d variable introduced to solve the Pricing Problem in Section 4.5). Constraints (4.5a) ensure the spanning of the destinations and constraints (4.5b) limit to one the light-trees defined on each wavelength λ.

The extended formulation, when compared to the compact one, contains less constraints $(q+k)$ but, except special and simple cases, has a huge number of variables (one for each feasible light-tree in the network). To solve its LP-Relaxation, i.e. formulation ECF$_{\mathrm{LP}}$ (4.4), we use the Column Generation method (see Section 2.2). In this context, formulation ECF$_{\mathrm{LP}}$ (4.4) plays the role of the Master Problem (MP).

At each iteration of the Column Generation method we need to solve an instance of the Reduced Master Problem (RMP) that is derived from MP by considering only a subset of the feasible light-trees. Given the optimal solution of the RMP, let $\pi_d \geq 0$ and $\nu_\lambda \leq 0$ denote the optimal values of the dual variables respectively associated with the destinations spanning constraints (4.5a) and with the wavelength consumption constraints (4.5b). Then, we can define the reduced cost associated with each feasible light-tree $T \in \mathscr{T}^\lambda$ as:

$$\bar{c}_T = \alpha c_T + (1-\alpha) - \sum_{d \in D} \bar{u}^\lambda_{T,d} \pi_d - \nu_\lambda$$

The *Pricing Problem* (PP) associated with the optimal solution of the current RMP requires to identify a light-tree having a negative reduced cost, or to prove that such a light-tree does not exist. Consequently, the Pricing Problem can be stated as:

$$\mathrm{PP} : z = \min_{T \in \bigcup_\lambda \mathscr{T}^\lambda} \{\bar{c}_T\}$$

If the optimal objective function value of the Pricing Problem is nonnegative, the optimal solution of the RMP is optimal also for the MP and we can stop the Column Generation method; otherwise, we can add any set of feasible light-trees having a negative reduced cost to the RMP constraint matrix, obtaining a new RMP instance, and iterate the process.

The Pricing Problem can be further decomposed by wavelengths obtaining k problems like the following one:

$$\mathrm{PP}^\lambda : z^\lambda = \min_{T \in \mathscr{T}^\lambda} \left\{ \alpha c_T - \sum_{d \in D} \bar{u}^\lambda_{T,d}\, \pi_d \right\} \tag{4.6}$$

If we solve the problem PP^λ for each $\lambda \in M$, we can solve the Pricing Problem simply by computing:

$$z = \min_{\lambda \in M} \{z^\lambda + (1-\alpha) - \nu_\lambda\}$$

4.5 Pricing Problem

In this section we describe the methods used to solve the Pricing Problem PP^λ introduced in the previous section. Hereafter, we refer to the PP^λ problem as the *Constrained Prize Collecting Steiner Arborescence Problem* (CPCSAP) and to simplify the notations, we refer to the graph $G^\lambda = (V, A^\lambda)$ as $G = (V, A)$, while $\mathscr{T}$ contains all the feasible light-trees in G^λ (i.e. $\mathscr{T} = \mathscr{T}^\lambda$). Moreover, we introduce the α multiplicative factor directly into the costs of the arcs.

Since any PCSTP (see Section 3.2) instance can be polynomially transformed in a CPCSAP instance (by adding a dummy root node and giving directions to arcs), CPCSAP is a $\mathscr{NP}$-hard problem. To the best of our knowledge CPCSAP has not yet been studied in the previous literature, however similar extensions of the Prize-Collecting Steiner Tree problem have been investigated (see Chapter 3 and Costa et al. 2008).

As discussed in Section 3.1, the CPCSAP is very similar to the *Knapsack Prize Collecting Steiner Tree problem* (KPCSTP). CPCSAP extends KPCSTP by introducing the arc directions, the source node and the splitting capabilities. Moreover, the delay bound constraint can be seen as a bunch of knapsack constraints (one for each directed path in the considered solution) that are similar to the one contained in the KPCSTP definition. As a consequence, to exactly solve the CPCSAP, we developed two exact methods based on two formulations introduced in Chapter 3 for the KPCSTP. The first exact method is based on a multi-commodity flow formulation similar to the one described in Section 3.4.2 and the second exact method is based on the formulation using the connectivity cuts constraints described in Section 3.5. In the first formulation the delay bound constraint can be easily introduced using a family of constraints whose number is polynomial in the size of the instance (see Section 4.5.1). The introduction of the same constraint in the second formulation is more challenging since, to satisfy it, we need to insert in the formulation a second family of constraints (in addition to the connectivity cuts one) whose number is exponential in the size of the instance (see Section 4.5.2). To efficiently identify the violated cuts belonging to this new family of constraints we developed an *ad hoc* separation algorithm. Nonetheless, as discussed in the following Section 4.6, the computational time required to solve the formulation based on the connectivity cuts is greater than the time required to solve the one based on the multi-commodity flow. This result is in contrast with what we obtained for the KPCSTP in Section 3.7.4, where the optimization method based on the connectivity cuts formulation outperformed all the competing algorithms. This different results can be explained considering both the slowdowns introduced by the second separation algorithm in resolution process for the connectivity cuts based formulation and the simplifications that can be obtained by introducing, in the multi-commodity flow formulation for the CPCSAP, only one commodity for each destination and not for each node.

We complete the analysis of the resolution methods for the CPCSAP by presenting a new Tabu Search heuristic that is used in cooperation with the previously described exact methods in order to speed the whole Column Generation method. At each iteration of the generation process, we first execute the Tabu Search heuristic for a limited number of iterations and, when it fails to find a feasible light-tree having a negative reduced cost, we switch to an exact method.

4.5.1 Multi-commodity flow formulation

In the following we describe a CPCSAP formulation derived from the multi-commodity flow formulation for the KPCSTP described in Section 3.4.2. We start by defining the following decision variables:

- f_a^d, equal to 1 if $a \in A$ is used and it belongs to the path connecting s with $d \in D$, 0 otherwise.
- x_a, equal to 1 if $a \in A$ is used, 0 otherwise.

- u_d, equal to 1 if $d \in D$ is spanned by the light-tree, 0 otherwise.

We introduce the following formulation:

$$\text{MCF} : \min \sum_{a \in A} c(a) x_a - \sum_{d \in D} \pi_d u_d \tag{4.7a}$$

$$\sum_{a \in \delta^-(s)} f_a^d - \sum_{a \in \delta^+(s)} f_a^d = -u_d \quad d \in D \tag{4.7b}$$

$$\sum_{a \in \delta^-(d)} f_a^d - \sum_{a \in \delta^+(d)} f_a^d = u_d \quad d \in D \tag{4.7c}$$

$$\sum_{a \in \delta^-(v)} f_a^d - \sum_{a \in \delta^+(v)} f_a^d = 0 \quad d \in D,\ v \in D \setminus \{s,d\} \tag{4.7d}$$

$$x_a \geq f_a^d \quad d \in D, a \in A \tag{4.7e}$$

$$\sum_{a \in \delta^-(v)} x_a \leq 1 \quad v \in V \tag{4.7f}$$

$$\sum_{a \in \delta^+(v)} x_a \leq \theta(v) \quad v \in V \tag{4.7g}$$

$$\sum_{a \in A} t(a) f_a^d \leq \Delta \quad d \in D \tag{4.7h}$$

$$u_d, x_a, f_a^d \in \{0,1\} \quad a \in A, d \in D$$

Constraints (4.7b, 4.7c, 4.7d) ensure that the solution is connected. Constraints (4.7e) define the used arcs. Constraints (4.7f,4.7g) limit respectively the indegree and the outdegree of the spanned nodes. Constraints (4.7h) ensure that the solution respects the delay bound threshold. Note that the objective function (4.7a) is equivalent to the one used in (4.6) since the α multiplicative coefficient has been introduced in the costs of the arcs in a preprocessing phase.

We consider the following modifications of MCF in order to strengthen it:

- *Delay bound constraints*: constraints (4.7h) can be strengthened by adding the u_d variable:

$$\sum_{a \in E} t(a) f_a^d \leq \Delta u_d \quad d \in D$$

- *Outdegree constraints*: as before, constraints (4.7g) can be strengthened by adding the u_d variable:

$$\sum_{a \in \delta_G^+(d)} x_a \leq \theta(d) u_d \quad d \in D \tag{4.8}$$

- *Flow balancing in non-destination nodes:* as already noticed in Ljubić et al. (2006), if $v \in V \setminus D$ is contained in an optimal solution then its outcut cannot be empty. Otherwise, the cost required to connect this node to other nodes in the solution is not balanced by any profit. Consequently, since the indegree of each spanned node is equal to 1, the following constraints are valid:

$$\sum_{a \in \delta_G^-(v)} x_a \leq \sum_{a \in \delta_G^+(v)} x_a \quad v \in V \setminus D \tag{4.9}$$

4.5.2 Connectivity cuts formulation

In the following we describe the CPCSAP formulation based on the connectivity cuts constraints derived from the similar formulation for the KPCSTP described in Section 3.5. We define the following decision variables:

- u_v, equal to 1 if $v \in V$ is used, 0 otherwise.
- x_a, equal to 1 if $a \in A$ is used, 0 otherwise.

At first we introduce the Connectivity Cuts Formulation (CCF). This formulation is a relaxation of CPCSAP since it does not consider the delay bound constraints. Later we show how to strengthen the formulation taking into account also the missing constraints.

$$\text{CCF}: \min \sum_{a \in A} \tilde{c}(a) x_a \tag{4.10a}$$

$$\sum_{a \in \delta_G^-(v)} x_a = u_v \quad v \in V - \{s\} \tag{4.10b}$$

$$\sum_{a \in \delta_G^+(v)} x_a \leq \theta(v) \quad v \in V \tag{4.10c}$$

$$\sum_{a \in \delta_G^-(S)} x_a \geq u_v \quad v \in V - \{s\}, S \subset V, v \in S, s \notin S \tag{4.10d}$$

$$x_a, u_v \in \{0,1\} \quad a \in A, v \in V \tag{4.10e}$$

Constraints (4.10b) link variables $\{x_a : a \in A\}$ with variables $\{u_d : d \in D\}$ and limit the indegree of the solution nodes to one. Constraints (4.10c) ensure that the nodes splitting capabilities are respected. Constraints (4.10d) ensure that the solution is connected. In the objective function (4.10a), we modify the cost of each arc $a = (i, j)$ in order to consider also the prize associated with j: $\tilde{c}(a)$ is equal to $\alpha c(a) - \pi_j$ if $j \in D$, otherwise it is simply equal to $\alpha c(a)$. Hence, the objective function is equivalent to the one used in (4.6).

Since the number of constraints (4.10d) is exponential in the number of WDM network nodes, to handle these inequalities we use the Branch & Cut framework (see, for example, Wolsey 1998). To solve the separation problem we use the same strategy described in Section 3.5. As mentioned above, the CCF formulation provides a relaxation of the CPCSAP since it does not consider the delay bound constraints. To satisfy these constraints, we developed an iterative row generation method based on the solution of minimum cost flow problems. Let $(\underline{x}', \underline{u}')$ be a solution of the LP relaxation of the CCF formulation, and let $d \in D$ be a destination node such that $u'_d > 0$. We consider the following minimum cost flow problem instance:

$$\text{PMCF}(\underline{x}',\underline{u}',d): z_d = \min \sum_{a\in A} t(a) f_a$$

$$\sum_{a\in\delta_G^-(s)} f_a - \sum_{a\in\delta_G^+(s)} f_a = -u'_d \tag{4.11a}$$

$$\sum_{a\in\delta_G^-(v)} f_a - \sum_{a\in\delta_G^+(v)} f_a = 0 \quad v\in V\setminus\{s,d\} \tag{4.11b}$$

$$\sum_{a\in\delta_G^-(d)} f_a - \sum_{a\in\delta_G^+(d)} f_a = u'_d \tag{4.11c}$$

$$0 \le f_a \le x'_a \quad a\in A \tag{4.11d}$$

PMCF($\underline{x}',\underline{u}',d$) requires to find the minimum cost flow that sends u'_d units of flow from the source s to the destination d, in the subgraph of G currently identified by $(\underline{x}',\underline{u}')$. The cost of each arc $a \in A$ is equal to the time $t(a)$ spent by a signal to go along the arc a. Hence, if $(\underline{x}',\underline{u}')$ is a feasible CPCSAP solution, the inequality $z_d \le \Delta$ must be satisfied. Otherwise, we need to find a valid inequality that cut the infeasible solution $(\underline{x}',\underline{u}')$ and add it to the CCF formulation.

In order to obtain this inequality, let ρ_v and $\nu_a \ge 0$ be the dual variables, associated, respectively, with the flow conservation constraints (4.11a,4.11b,4.11c) and with the capacity constraints (4.11d). We introduce the PMCF($\underline{x}',\underline{u}',d$) dual problem:

$$\text{DMCF}(\underline{x}',\underline{u}',d): z_d = \max \ u'_d\rho_d - u'_d\rho_s - \sum_{a\in A} \nu_a x'_a$$

$$\rho_j - \rho_i - \nu_{ij} \le t_{ij} \quad (i,j)\in A$$

$$\nu_a \ge 0 \quad a\in A$$

Theorem 1. *Let $(\underline{x}',\underline{u}')$ be a solution of the LP relaxation of the CCF formulation and d a destination such that $u'_d > 0$. Moreover, let $(\underline{\rho}^*,\underline{\nu}^*)$ and z_d be, respectively, an optimal solution and the optimal objective function value of* DMCF($\underline{x}',\underline{u}',d$). *If $z_d > \Delta u'_d$ then $(\underline{x}',\underline{u}')$ is not a feasible solution for the CPCSAP problem, and the inequality:*

$$-\sum_{a\in A} \nu_a^* x_a \le (\Delta + \rho_s^* - \rho_d^*) u_d \tag{4.12}$$

is a valid inequality for the CCF formulation which cuts $(\underline{x}',\underline{u}')$ and does not cut any feasible solution of CPCSAP.

Proof. First we prove that, if $z_d > \Delta u'_d$, then $(\underline{x}',\underline{u}')$ is unfeasible. By contradiction, suppose $(\underline{x}',\underline{u}')$ is a feasible CPCSAP solution, since $u'_d > 0$ and since $(\underline{x}',\underline{u}')$ is integral, we have $u'_d = 1$ and consequently $z_d > \Delta u'_d$ can be restated as $z_d > \Delta$. From what we said before for the problem PMCF($\underline{x}',\underline{u}',d$) and using the strong duality theorem, the delay bound constraint on the destination d is not satisfied contradicting the hypothesis.

Since z_d is the optimal objective function value of the problem DMCF$(\underline{x}', \underline{u}', d)$ and $(\underline{\rho}^*, \underline{\nu}^*)$ is its optimal solution we can restate $z_d > \Delta u'_d$ as:

$$z_d = u'_d \rho^*_d - u'_d \rho^*_s - \sum_{a \in A} \nu^*_a x'_a > \Delta u'_d$$

The previous inequality can be rewritten as follow:

$$-\sum_{a \in A} \nu^*_a x'_a > (\Delta + \rho^*_s - \rho^*_d) u'_d$$

Consequently $(\underline{x}', \underline{u}')$ violates cut (4.12).

Now we show that inequality (4.12) does not cut any CPCSAP feasible solution. To prove this, observe that the feasibility of a DMCF$(\underline{x}', \underline{u}', d)$ solution does not depend on $(\underline{x}', \underline{u}')$ values. Consequently, each feasible DMCF$(\underline{x}', \underline{u}', d)$ solution is feasible also for the different problem DMCF$(\underline{\tilde{x}}, \underline{\tilde{u}}, d)$ obtained considering any CPCSAP feasible solution $(\underline{\tilde{x}}, \underline{\tilde{u}})$. Since $(\underline{\tilde{x}}, \underline{\tilde{u}})$ is integral, there are only two possible cases:

- $\tilde{u}_d = 1$: let $\tilde{z}_d$ be the optimal objective function value of the DMCF$(\underline{\tilde{x}}, \underline{\tilde{u}}, d)$ problem. From what we said before, $(\underline{\rho}^*, \underline{\nu}^*)$ is a feasible solution of the dual problem DMCF$(\underline{\tilde{x}}, \underline{\tilde{u}}, d)$ and since $(\underline{\tilde{x}}, \underline{\tilde{u}})$ respects the delay bound constraints, we know that $\tilde{z}_d \leq \Delta \tilde{u}_d$. Consequently, by applying the weak duality theorem we obtain:

  $$\tilde{u}_d \rho^*_d - \tilde{u}_d \rho^*_s - \sum_{a \in A} \nu^*_a \tilde{x}_a \leq \tilde{z}_d \leq \Delta \tilde{u}_d$$

  The previous inequality can be rewritten as follow:

  $$-\sum_{a \in A} \nu^*_a \tilde{x}_a \leq (\Delta + \rho^*_s - \rho^*_d) \tilde{u}_d$$

- $\tilde{u}_d = 0$: the inequality (4.12) becomes $-\sum_{a \in A} \nu^*_a \tilde{x}_a \leq 0$ and, since, for each $a \in A$, $\nu^*_a \geq 0$ and $x_a \in \{0, 1\}$, the considered inequality is satisfied.

Finally, we observe that even the CCF formulation can be strengthened by introducing the flow balancing constraints (4.9) and, since there is a u_v variable for each $v \in V$, we can add the constraints (4.8) even for the non-destination nodes.

4.5.3 Tabu search

We do not need to solve each Pricing Problem exactly: to improve the RMP definition, it is sufficient to find a light-tree having a negative reduced cost. We need an exact pricing solver only to guarantee that light-trees with negative reduced cost do not exist. Consequently, to speed up the Column Generation method, we developed a Tabu Search heuristic to which we will refer in the following as PPTS.

Tabu search is a well-known local search metaheuristic approach which allows the visit of non-improving solutions, and is controlled by memory mechanisms to avoid the insurgence of cyclic behaviors. It was introduced by Glover (Glover, 1986) and the interested reader can find in Glover and Laguna (1997) a detailed treatment of its applications and variants. In the following, we mainly focus on the specific aspects of our implementation.

To simplify the description of PPTS, we introduce the set $C = \{v \in V \mid \theta(v) > 1\}$ containing all the multicast capable nodes in the WDM network. PPTS is initialized using the light-tree T that contains only the source node s. At each iteration PPTS executes a move composed by one or both the following steps:

1. Add to the current solution T a path connecting a node in the set $S_A(T) = \{v \in V(T) \mid \theta(v) >\mid \delta^+_T(v) \mid\}$ (i.e. the set that contains every node v for which we can add arcs to v outcut maintaining the feasibility of T) with a node in the set $R_A(T) = \{v \notin V(T) \mid v \in D \cup C\}$ (i.e. the set that contains every node v not spanned by T and such that v is a destination or a MC node).
2. Delete from the current solution T a path connecting a node in the set $S_D(T) = \{v \in V(T) \mid v \in D \cup C\} \cup \{s\}$ (i.e. the set that contains the source and every node v spanned by T such that v is a destination or a MC node) with a node in the set $R_D(T) = \{v \in V(T) \mid \mid \delta^+_T(v) \mid = 0\}$ (i.e. the set of the T leaves).

Therefore, the PPTS solution space does not contain every feasible light-tree T, but it contains those trees whose leaves are either destination nodes or MC nodes and the starting tree containing only the source node.

The number of paths that we can add to the current solution maintaining its feasibility can be huge. In order to speed up the evaluation of the moves, we restrict the neighborhood definition: we heuristically define, for each $n_1 \in \bigcup_{T \in \mathscr{T}} S_A(T)$ and $n_2 \in \bigcup_{T \in \mathscr{T}} R_A(T)$, a small subset of paths that can be added to a solution to connect n_1 with n_2. In the following we denote this subset as $\mathscr{P}(n_1, n_2)$. When PPTS evaluates moves to connect n_1 with n_2, it can use only the paths contained in $\mathscr{P}(n_1, n_2)$.

The heuristic we develop to define $\mathscr{P}(n_1, n_2)$ uses two different objective functions to evaluate paths: the *cheapest* objective function, in which the cost of a path arc $a \in A$ is equal to $c(a)$ and the *fastest* objective function, in which the cost of a path arc $a \in A$ is equal to $t(a)$. The heuristic produces a mixture of the cheapest and the fastest paths using the following method that is parametrized by `lPath` and `wIt`:

1. Let $P = \{p_1, \dots, \dots, p_{\texttt{lPath}}\}$ be the set of the `lPath` cheapest paths not already generated. Then set $\mathscr{P}(n_1, n_2) = \mathscr{P}(n_1, n_2) \cup \{p \in P \mid t(p) \leq \Delta\}$. If $\mid \mathscr{P}(n_1, n_2) \mid \geq$ `lPath` then stop, else go to the next step.
2. Let p be the fastest path not already generated. If $t(p) > \Delta$, stop. Otherwise, set $\mathscr{P}(n_1, n_2) = \mathscr{P}(n_1, n_2) \cup \{p\}$. If $\mid \mathscr{P}(n_1, n_2) \mid =$ `lPath` then stop, else return to the previous step.

The $\mathscr{P}(n_1, n_2)$ size is dynamically updated: initially it is equal to `lPath` and after `wIt` iterations without any best solution improvement, PPTS adds one new path to each set $\mathscr{P}(n_1, n_2)$.

The tabu mechanism has the purpose to avoid reversing recently performed moves. We implemented this mechanism using three different tabu memories: MFIRST, MLAST and MPAIR. Each tabu memory is implemented as a list of move attributes and it is updated using a *first in first out* strategy. In MFIRST the move attribute is defined by the first arc of the path added by the move, in MLAST the move attribute is defined by the last arc of the path added by the move, and in MPAIR the move attribute is defined by the pair composed by the first and the last arc of the path added by the move.

The size of MLAST and MFIRST is equal to half the size of MPAIR. The MPAIR size changes dynamically in order to increase the algorithm freedom while it is near to a local minimum.

4.6 Computational experiments

4.6.1 Algorithms for the subproblem

We developed the algorithm based on the multi-commodity flow formulation (denoted by MCALG in the following) using the ILOG CONCERT 2.9 and ILOG CPLEX 12.4. The formulation we give to the solver contains all the strengthenings and the new constraints described in Section 4.5.1. To explore the branching tree, we used the best bound strategy and the CPLEX default branching rule. The algorithm based on the connectivity cuts formulation (denoted by CCALG in the following) has been developed using the same ILOG technologies used by MCALG and also CCALG uses the best bound exploration strategy and the CPLEX default branching rule. To increase the number of cuts added during the separation phase of the connectivity cuts constraints (4.10d), we used both the *back cuts* and the *nested cuts* strategies described in Section 3.4.2 and, to avoid slowing down the algorithm, we limit the number of added cuts in each separation phase to `MNestedCuts`. We inserted in the formulation all the constraints strengthenings introduced in Section 4.5.2.

Finally, in Section 3.7.3 we have seen that directly avoiding cycles of length 2 is very effective when we need to solve KPCSTP, thus, we applied the same strategy for the CPCSAP. However, in preliminary tests we have seen that better results can be achieved if we consider only the 2-cycles that contain the destination nodes. As a consequence, both in MCALG and in CCALG we introduced the following constraints.

$$x_{ij} + x_{ji} \leq u_j \qquad\qquad j \in D, (i,j) \in A, (j,i) \in A \tag{4.13}$$

We implemented the maximum flow algorithm following Cherkassky and Goldberg (1997). We solve Minimum Cost Flow problems with the A. Löbel's implementation of the network simplex algorithm (available at `http://www.zib.de/Optimization/Software/Mcf/`) called through the MCFClass framework (developed by A. Frangioni and C. Gentile, and available at `http://www.di.unipi.it/optimize/Software/MCF.html`). Finally, in the PPTS implementation, to generate the fastest and the cheapest paths, we used Yen's algorithm to find the k shortest paths (see, for example, Lawler 1976).

4.6.2 Column Generation

To ensure the initial RMP feasibility, we use a dummy column representing a light-tree that spans all the destination nodes without using any wavelength. We associate with this column a very high cost; therefore, if the corresponding variable has a nonzero optimal value at the end of the Column Generation method, then the given MRWADC instance is unfeasible. At each iteration of the Column Generation method, we split the search for a new light-tree in two steps:

- *Heuristic search*: for each $\lambda \in M$, we execute PPTS for `nTS` iterations on the CPCSAP instance derived from PP^{λ}. If PPTS finds a solution with a reduced cost less than the negative threshold `tsTH`, we add the corresponding light-tree to RMP and we reoptimize it.
- *Exact search*: if we do not find any new column in the previous step, for each $\lambda \in M$, we solve the associated CPCSAP instance using either MCALG or CCALG (in this way we have two different Column Generation algorithms which we need to compare). If during this process we find a light-tree with a negative reduced cost, we add it to RMP and we reoptimize it.

The sequence of tested wavelengths during this procedure is based on a dynamically updated wavelength list. At the end of each Column Generation iteration we append to this list the wavelength associated with the last light-tree inserted in RMP.

4.6.3 Parameters and Instances generation

After few initial tests we chose the following values for the algorithms parameters. These values allow us to obtain fast Column Generation convergence and execution:

- CCALG: `MNestedCuts` $= 50$.
- PPTS: `wIt` $= 50$ and `lPath` $= 5$.
- Column Generation: `nTS` $= 200$ and `tsTH` $= -0.1$.

We generated the tested MRWADC instances using the process described in Chen et al. (2008). The network structure generation method has been introduced in Waxman (1988): n nodes are uniformly distributed on a $w \times h$ discrete rectangular grid. Given two nodes i and j, arc (i,j) belongs to the network with a probability equal to $P(i,j) = \mu e^{\frac{p(i,j)}{\delta\gamma}}$, where $p(i,j)$ is the euclidean distance between i and j, δ is the maximum distance between two network nodes, μ and γ are two parameters which, respectively, determine the arcs density and the mean arc length. For each generated arc $a = (i, j)$ we set $c(a) = p(i, j)$ and we generate $t(a)$ uniformly between 0.1 and 3. We generate the node splitting capabilities (i.e. the function θ) by randomly selecting 15% of the nodes as multicast capable and for these nodes we generate the maximum number of reproductions uniformly between 2 and the node outdegree. We generate the source and the destinations of the transmission requests uniformly among the network nodes; we set the delay bound threshold Δ to $\chi\tau$, where τ is the minimum time required to reach all the destinations from the source and χ is a parameter that allows us to determine the tightness of the delay bound constraints. We define the congestion function $w : E \to \{0, 1\}$ generating b random paths using the following method:

1. Choose uniformly the wavelength $\lambda \in M$ on which the path is generated and the starting node v.
2. Choose uniformly the next path arc $(v, w) \in \delta^+_{G^\lambda}(v)$ that does not create a cycle with the already inserted arcs. If no arc can be added to the path then stop. Otherwise, go to the next step.
3. Stop with probability 0.7. Otherwise, set $v = w$ and go to the previous step.

If the communication channel $((i, j), \lambda)$ belongs to any of the generated paths we set $w((i, j), \lambda) = 0$, otherwise we set $w((i, j), \lambda) = 1$. The instances tested for this chapter are obtained with parameters $\mu = 0.7$, $\gamma = 0.9$, $w = 100$, $h = 100$ and $b = \lfloor \frac{n}{3} \rfloor$.

4.6.4 Experimental results

To test our methods, we generated random instances having $n \in \{20, 40, 60, 80, 100\}$, $q \in \{3, 4, 5\}$, $\chi \in \{1.2, 2, 3\}$, $(1-\alpha)/\alpha \in \{0.1, 1, 10, 50\}$ and considering $k = 5$ wavelengths. For each typology, we generated 5 instances, using the process described above. Therefore, the total number of tested instances is 900. We executed the following tests using a PC equipped with an Intel Core2 Quad-core 2.66Ghz and 4GB of RAM. In each test we imposed a time limit of two hours.

The first computational experiments compare the results obtained by directly solving with CPLEX the compact formulation CF (see (4.1a)-(4.1h) in section 4.2) with the ones obtained by the Column Generation method to solve the LP-Relaxation of formulation EF (see section 4.4). Moreover, in these experiments we also compare the two different algorithms (CCALG and MCALG) we developed to exactly solve the instances of CPCSAP (see section 4.5) generated during the Column Generation phase. We test all these three methods on all the instances. Table 4.1 contains, for each n, q and χ value, the average computation time required by the Column Generation algorithm using MCALG (columns T_{MC}) and CCALG (columns T_{CT}), and the average computation time required by CPLEX to solve CF (columns T_{CF}).

If one of the three algorithms fails to solve some instances within the time limit or consumes all the available memory, we report in parenthesis the number of unsolved instances. When considering triplets of comparable values we report in bold the best of the three. Columns T_{CF} show how formulation CF can solve all the 360 instances

		$\chi = 3$			$\chi = 2$			$\chi = 1.2$	
n q	T_{MC}	T_{CT}	T_{CF}	T_{MC}	T_{CT}	T_{CF}	T_{MC}	T_{CT}	T_{CF}
20 3	0.75	0.60	**0.28**	0.75	0.65	**0.22**	0.35	0.25	**0.17**
20 4	0.95	0.75	**0.70**	0.85	**0.60**	0.63	0.50	**0.45**	0.60
20 5	1.15	**0.75**	1.99	1.60	**1.20**	4.18	**1.00**	1.10	2.91
40 3	1.80	**1.25**	1.53	3.60	3.35	**1.20**	2.95	2.90	**1.20**
40 4	1.35	1.00	**0.88**	2.45	**2.15**	2.90	2.25	2.25	**1.18**
40 5	2.30	**1.25**	5.72	5.70	**4.65**	11.77	**4.35**	9.60	141.81
60 3	**9.00**	10.85	34.29	**20.85**	43.15	23.43	13.95	65.65	**11.27**
60 4	**15.85**	46.20	107.78	**23.40**	144.20	380.84(2)	**13.55**	57.15	54.75(1)
60 5	**27.00**	100.75	748.55(3)	**50.25**	199.50	839.74(3)	**19.50**	379.45(1)	52.88(8)
80 3	**15.05**	17.80	141.43	**32.60**	73.20	395.18	**24.50**	445.30	652.32(1)
80 4	**20.80**	28.15	248.63	**36.05**	58.05	241.16	**27.90**	779.50	148.70(6)
80 5	**286.55**	334.50(1)	814.36(8)	**278.75**	544.50(3)	1356.87(7)	**63.85**	1090.19(5)	1483.62(14)
100 3	**18.45**	104.95	190.43	**22.05**	240.80	315.70(2)	**8.35**	26.90	368.65
100 4	**19.15**	27.55	324.03	**24.00**	83.65	697.72	**18.35**	42.75	136.05
100 5	**148.55**	283.62(2)	201.07(4)	**103.45**	1218.25(2)	1067.92(5)	**32.75**	180.19(4)	336.20(10)

Table 4.1 Mean execution time comparison between MCALG, CCALG and formulation CF.

with up to 40 nodes. However, it cannot be effectively used when solving instances with $n \geq 60$ and $q \geq 4$. Overall, using formulation CF, we cannot solve 74 instances out of the 540 with $n \geq 60$. Moreover, on the solved instances having at least 60 nodes, the time required by CPLEX is considerable larger w.r.t. the time required by the Column Generation method. Finally, in some preliminary tests, (not reported here for the sake of brevity), we compared the results obtained using CF with the ones obtained using the similar ILP compact formulation proposed in Chen et al. (2008): we did not find any significant differences between the performance of the two compact formulations.

Comparing columns T_{MC} and T_{CT}, we see that, on average, MCALG is the fastest algorithm. The performance gap between MCALG and CCALG increases if we increase the size of the instances: if $n \geq 60$ the average time required by MCALG is less then the average time required by CCALG on all instances classes. Moreover, MCALG is the only algorithm that terminates within the time limit on all the instances.

This behavior is different from the one we observed in Chapter 3 for the KPCSTP and in our preliminary tests (the results are not reported here for the sake of brevity). In these tests we compared the different Pricing Problem resolution strategies using a sample of all the pricing instances generated from the Column Generation process. As a result, CCALG generated less branching nodes, better root node bounds and required less computation time, w.r.t. MCALG. Despite these results, MCALG remains the best choice when we need an exact solver CPCSAP embedded in a

Column Generation algorithm that uses also a heuristic to solve the Pricing Problem. We think these somewhat conflicting results can be explained by the fact that MCALG is more suitable in solving the particular CPCSAP instances generated at the end of the Column Generation phase that, in our experience, are the hardest ones among the instances generated during the whole generation phase. The other CPCSAP instances are easily solved using the PPTS heuristic.

Since in the previous test we observed that MCALG is the best choice to solve the Pricing Problem, we used it to analyze how the different components of the Column Generation algorithm contribute to the resolution process and how different $(1-\alpha)/\alpha$ values determine the difficulty of the instances. In Table 4.2 we report for each class of instances, now aggregated by $(1-\alpha)/\alpha$ values, the average computation time required by MCALG (column T) and the average number of light-trees (i.e. columns) contained in the optimal solutions (column C_{SOL}). For each row we report

	$(1-\alpha)/\alpha = 50$		$(1-\alpha)/\alpha = 10$		$(1-\alpha)/\alpha = 1$		$(1-\alpha)/\alpha = 0.1$	
n q	T	C_{SOL}	T	C_{SOL}	T	C_{SOL}	T	C_{SOL}
20 3	0.47	1.33	**0.67**	1.60	**0.67**	1.67	**0.67**	1.67
20 4	**0.93**	2.67	0.80	2.93	0.67	3.00	0.67	3.00
20 5	**1.33**	2.20	1.27	3.00	1.20	3.20	1.20	3.20
40 3	**3.60**	1.60	2.13	1.73	2.40	1.73	3.00	1.80
40 4	1.67	2.13	1.80	2.33	2.20	2.33	**2.40**	2.33
40 5	3.20	2.20	4.20	2.33	**4.53**	2.33	**4.53**	2.33
60 3	**15.20**	1.73	13.60	2.07	14.93	2.27	14.67	2.27
60 4	**18.60**	2.07	16.40	2.40	17.47	2.47	17.93	2.47
60 5	**40.07**	2.00	31.00	2.53	28.93	2.73	29.00	2.73
80 3	**27.13**	1.60	23.67	1.93	22.27	1.93	23.13	1.93
80 4	**37.20**	1.80	26.53	2.33	25.47	2.67	23.80	2.80
80 5	**622.67**	1.93	83.87	2.33	69.20	2.73	63.13	2.73
100 3	13.33	2.07	16.60	2.13	**17.73**	2.33	17.47	2.33
100 4	**24.07**	1.53	21.67	1.80	16.53	1.93	19.73	1.93
100 5	**166.47**	2.07	82.07	2.53	58.33	2.73	72.80	2.73

Table 4.2 Time required by MCALG and number of light-trees in the optimal solutions varying $(1-\alpha)/\alpha$ ratio.

in bold the highest average computation time. Considering this table and Table 4.1, we can observe how the difficulty of the MRWADC problem varies w.r.t. both χ and $(1-\alpha)/\alpha$. The instances having $\chi = 2$ require more computation time w.r.t. instances having $\chi = 3$ or $\chi = 1.2$. This behaviour can be explained by the fact that by increasing χ we weaken the delay bound and by decreasing χ we reduce the number of feasible light-trees and, as a result, we simplify the problem. As for the $(1-\alpha)/\alpha$ value, passing from $(1-\alpha)/\alpha = 50$ to $(1-\alpha)/\alpha = 10$ we can significantly decrease the required computation time but further decreases in $(1-\alpha)/\alpha$ value do not further simplify the instances. Perhaps, while moving from $(1-\alpha)/\alpha = 50$ to $(1-\alpha)/\alpha = 10$ we pass through a phase transition, but the only thing that we can say is that if our objective is the one that requires primarily to minimize the transmission cost we can solve the MRWADC problem more effectively. Since by increasing $(1-\alpha)/\alpha$, we increase the relative cost of using a new light-tree we expect that increasing $(1-\alpha)/\alpha$ we decrease the number of light trees in the optimal solutions. The values contained in columns C_{SOL} of Table 4.2 validate this statement. Moreover, even in this quantity there is probably a phase transition: the average decrement in C_{SOL} is equal to 17.7% if we move from $(1-\alpha)/\alpha = 50$ to $(1-\alpha)/\alpha = 10$ while it is only 5.97% and 0.59% if we, respectively, move from $(1-\alpha)/\alpha = 10$ to $(1-\alpha)/\alpha = 1$ and from $(1-\alpha)/\alpha = 1$ to $(1-\alpha)/\alpha = 0.1$.

In order to analyze how the different parts of our algorithm interact with each other and how good the obtained feasible solutions are, in Table 4.3 we report for each pair (n,q) the following values:

- $\%T_{EX}$ is the average percentage of the overall computation time used by the exact pricing solver (i.e. MCALG).
- $\%T_{HE}$ is the average percentage of the overall computation time used by the heuristic for the pricing solver (i.e. PPTS).
- C is the average number of the total generated columns.
- C_{EX} is the average number of the columns generated by the exact pricing solver.
- $\%\Delta_{CG}$ is the average percentage gap between the lower bound obtained by the Column Generation algorithm and the upper bound obtained solving the final Reduced Master Problem with the integrality constraint. In parenthesis we report the number of instances for which this gap is greater than zero. Notice that this value is equal to the number of instances that we cannot solve to optimality.
- $\%\Delta_{CF}$ is the average percentage gap between the best lower bound and the best upper bound found by CPLEX when we use it to solve formulation CF. In parenthesis we report the number of instances for which this gap is greater than zero. Notice that this value is equal to the number of the instances that CPLEX cannot solve within time limit and without consuming all the available memory.

Notice that columns $\%T_{EX}$ and $\%T_{HE}$ do not sum to 100% since the remaining time is used by CPLEX to solve the generated RMP instances and the final RMP instance with the integrality constraint. Considering the values reported in Table 4.3, we see how it is important to have a good exact solver for the Pricing Problem considering the percentage of time required by the exact solver ($\%T_{EX}$) and the number of overall columns it generates (C_{EX}). For many instance classes (i.e. the ones for which $C_{EX} = 0.0$) we need to use the exact solver only to prove the optimality of the final Reduced Master Problem and, as for the other instances, the exact solver generates very few columns. Nonetheless the time required by the exact solver is always greater than 22.8% and increases up to 60.49% for instances having $n = 80$ and $q = 5$.

Finally, a comparison between columns $\%\Delta_{CG}$ and $\%\Delta_{CF}$ shows how the light-trees generated by the Column Generation phase are good even if we consider the integrality constraint: on 886 out of the 900 tested instances our algorithm is able to find an optimal solution. Moreover it is able to find very good quasi-optimal solutions for the remaining 14 instances. On the other hand, using formulation CF we cannot solve to optimality 74 instances and the feasible solutions found by CPLEX in two hours of computation time are worse w.r.t. the ones obtained by the Column Generation method in less time. Considering both Table 4.1 and Table 4.3 we

n	q	$\%T_{EX}$	$\%T_{HE}$	C	C_{EX}	$\%\Delta_{CG}$	$\%\Delta_{CF}$
20	3	37.18	53.89	6.23	0.00	0.00	0.00
20	4	40.46	48.97	6.02	0.00	0.00	0.00
20	5	42.06	53.43	10.87	0.02	0.03(1)	0.00
40	3	33.92	56.30	6.27	0.02	0.00	0.00
40	4	37.46	56.84	9.18	0.00	0.00	0.00
40	5	36.94	52.25	14.93	0.90	0.06(3)	0.00
60	3	27.93	71.46	7.23	0.15	0.00	0.00
60	4	32.01	67.99	10.52	0.05	0.08(1)	1.31(3)
60	5	48.02	51.67	15.67	0.57	0.12(4)	8.58(14)
80	3	27.43	71.52	5.85	0.05	0.09(1)	0.63(1)
80	4	41.01	58.09	10.23	0.18	0.00	4.55(6)
80	5	60.49	39.19	17.40	0.83	0.13(3)	19.01(29)
100	3	22.80	74.34	5.95	0.00	0.00	0.70(2)
100	4	35.77	60.92	12.72	0.13	0.00	0.00
100	5	50.19	47.01	15.68	1.73	0.03(1)	14.19(19)

Table 4.3 Summary of MCALG performance and average percentage gaps obtained with formulation CF.

can conclude that the standard method based on a compact formulation is competitive with our Column Generation algorithm only when we consider small instances ($n \leq 40$), when considering larger instances our algorithm outperforms the standard one.

In theory, using the Branch and Price framework, we could extend our Column Generation approach to solve to optimality also the instances for which the optimal solution of the LP-Relaxation of formulation EF are fractional. However, the intrinsic difficulty of the Pricing Problem makes quite complex the development of a fast exact method that uses our decomposition strategy. Nonetheless, the quasi-optimal solutions obtained for the unsolved instances could be improved using a more sophisticated Column Generation based heuristic (see Section 2.3).

4.7 Conclusion

In this chapter we have developed a Column Generation method that can be used to efficiently solve the MRWADC problem. The LP relaxation of the new extended formulation that we proposed is very strong and on most of the tested instances the solutions obtained by solving this relaxation are integral. On the other instances, by adding the integrality constraint to the last Reduced Master Problem obtained by the Column Generation method we can find near optimal solutions. The time required by the Column Generation method to obtain these results is significantly lower than the time required to solve a compact ILP formulation for the MRWADC problem.

To solve the $\mathcal{NP}$-hard Pricing Problem that arises from our decomposition approach we developed two exact methods and a Tabu Search heuristic. Both exact methods have been developed by adapting an exact method for the KPCSTP proposed in Chapter 4. In particular, the first method derives from the multi-commodity flow formulation described in Section 3.4.2 while the other one derives from the connectivity cuts formulation described in Section 3.5. Using computational experiments we showed that, differently from what obtained for the KPCSTP (see Section 3.7), the multi-commodity flow formulation outperforms the one based on the connectivity cuts.

For what concerns the cooperation between the exact and the heuristic methods to solve the Pricing Problem instances, since the total number of generated columns is very low and since the heuristic generates most of the columns, in many situations, the exact method needs to solve only the final Pricing Problem instances. Nonetheless, since a great part of computation time is used to prove the optimality of the solution of the final Reduced Master Problem, it is important to have a fast exact method. Finally, the results that can be obtained by adding the integrality constraint to the last Reduced Master Problem generated by the Column Generation method are very close to the computed lower bound.

4.8 References

B. Chen and J. Wang. Efficient routing and wavelength assignment for multicast in wdm networks. *Selected Areas in Communications, IEEE Journal on*, 20(1):97 –109, 2002.

M. T. Chen and S. S. Tseng. A genetic algorithm for multicast routing under delay constraint in WDM network with different light splitting. *Journal of information science and engineering*, 21:85–108, 2005.

M. T. Chen, B. M. T. Lin, and S. S. Tseng. Multicast routing and wavelength assignment with delay constraints in WDM networks with heterogeneous capabilities. *Journal of Network and Computer Applications*, 31(1):47–65, 2008.

B. V. Cherkassky and A. V. Goldberg. On Implementing the Push-Relabel Method for the Maximum Flow Problem. *Algorithmica*, 19(4):390–410, 1997.

I. Chlamtac, A. Ganz, and G. Karmi. Lightpath communications: an approach to high bandwidth optical WAN's. *Communications, IEEE Transactions on*, 40(7): 1171–1182, 1992.

F. Colombo and M. Trubian. A column generation approach for multicast routing and wavelength assignment with delay constraints in heterogeneous wdm networks. *Annals of Operations Research*, 2013. doi: 10.1007/s10479-013-1403-7.

A. M. Costa, J. F. Cordeau, and G. Laporte. Fast heuristics for the steiner tree problem with revenues, budget and hop constraints. *European Journal of Operational Research*, 190(1):68–78, 2008.

M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

X. Jia, D. Du, and X. Hu. Integrated algorithms for delay bounded multicast routing and wavelength assignment in all optical networks. *Computer Communications*, 24(14):1390–1399, 2001.

E. Lawler. *Combinatorial Optimizations: Networks and Matroids*. Holt, Rinehar and Winston, 1976.

I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem. *Mathematical programming*, 105(2-3):427–449, 2006.

B. Mukherjee. WDM Optical Communication Networks: Progress and Challenges. *IEEE Journal on selected areas in communications*, 18(10):1810–1824, 2000.

B. Mukherjee. *Optical WDM Networks*. Springer, 2006.

L. H. Sahasrabuddhe and B. Mukherjee. Light-Trees: Optical Multicasting for Improved Performance in Wavelength-Routed Networks. *Communications Magazine, IEEE*, 37(2):67–73, 1999.

N. Sreenath, C. Siva Ram Murthy, and G. Mohan. Virtual Source Based Multicast Routing in WDM Optical Networks. *Photonic Network Communications*, 3(3): 213–226, 2001.

A. Tzanakaki, K. Katrinis, T. Politi, A. Stavdas, M. Pickavet, P. Van Daele, D. Simeonidou, M. O'Mahony, S. Aleksic, L. Wosinska, and P. Monti. Dimensioning the future pan-european optical network with energy efficiency considerations. *Optical Communications and Networking, IEEE/OSA Journal of*, 3(4):272 – 280, april 2011.

B. M. Waxman. Routing of Multipoint Connections. *Selected areas in communications, IEEE journal on*, 6(9):1617–1622, 1988.

L. A. Wolsey. *Integer Programming*. Wiley, 1998.

S. Yan, M. Ali, and J. Deogun. Route Optimization of Multicast Sessions in Sparse Light-splitting Optical Networks. *IEEE GLOBECOM'01*, 4:22–29, 2001.

X. Zhang, J. Y. Wei, and C. Qiao. Constrained Multicast Routing in WDM Networks with Sparse Light Splitting. *Journal of Ligthwave Technology*, 18(12): 1917–1927, 2000.

Q. Zhu, M. Parsa, and J. J. Garcia-Luna-Aceves. A source-based algorithm for delay-constrained minimum-cost multicasting. *Annual Joint Conference of the IEEE Computer and Communications Societies*, 1:377–385, 1995.

Chapter 5
The Homogenous Areas Problem

5.1 Introduction

Italy has adopted a federal and decentralized model of state administration. Town councils, in particular, are in charge of managing a large amount of matters involving their own territory. Several of these matters, however, involve different towns at the same time, and therefore can be managed efficiently only with a certain deal of coordination. The role to coordinate them is played by the *province*, an intermediate level of government between the towns and the central government. Each province periodically writes a Land Coordination Plan (*Piano Territoriale di Coordinamento*), which builds a framework for the overall policy of cooperation among the towns, and defines the procedures to implement coordinated actions. Then, the province works in joint with the single town administrations, providing them information and supporting their interactions on the matters of common interest.

Some provinces are rather large and involved in hundreds of different activities. This poses a problem of work organization, namely:

How to achieve an effective interaction between the personnel of the towns and the experts of specific fields who work at the province.

The solution adopted by the province of Milan, which includes 134 towns, is to create a "customer care" layer of about 20 employees, with the task to support the employees of the towns. For the sake of efficiency, a certain degree of specialization in this layer is desirable, in order to limit the expertise required from each employee. To this purpose, the province is partitioned into "homogeneous areas", whose towns share a large number of common activities, different from the ones of the other areas. Then, a team of employees is assigned to each area and specifically trained on the corresponding activities. A by product of this structure is the improved cooperation by means of personal relationships, since each team becomes a friendly and competent reference for the towns of the associated area. The workload imposed on the single teams should be similar for the sake of equity and is limited by the number of working hours available for the coordination activities. In general, this partition implies some redundancies, because a few activities involve towns from different areas, whose teams have to gain the same expertise and to perform the same operations. Such redundancies should be minimized. This gives rise to a special Graph Partitioning Problem (GPP) with a nonlinear objective function and additional side constraints which, to the best of our knowledge, has not yet been taken into account in the huge literature on GPPs. The solution provides a reference structure for the actual workflow organization, which of course should also take into account local agreements, as well as historical, social and political issues.

In the following section we define formally the *Homogeneous Areas Problem* (HAP). In Section 5.3 we investigate the computational complexity of the problem. In Section 5.4 we describe the instances used throughout the chapter to test and compare our algorithms. In Section 5.5 we discuss the relationship between the classical GPPs and the HAP. In Section 5.6 we introduce and compare two different compact formulations for the problem and some valid inequalities to tighten them. In Section 5.7 we describe a Column Generation method for the HAP and the algorithms that we developed to efficiently solve the associated Pricing Problem. In Section 5.8 and Section 5.9 we introduce and compare, respectively, two local search based heuristics and three Column Generation based heuristics. Finally, in Section 5.10 we provide the results that we obtained by executing the proposed methods to solve two real world HAP instances.

Finally, note that this chapter summarizes and extends our previous works related to the HAP and described in (Colombo et al., 2011, 2012; Ceselli et al., 2013; Colombo et al., 2013).

5.2 Problem definition

The HAP can be formulated as follows. Let $G = (V, E)$ be an undirected graph with $V = \{1, \ldots, n\}$, $\mathscr{S} \subseteq 2^V$ a collection of subsets of vertices, $q : \mathscr{S} \to \mathbb{R}^+$ a cost function defined on $\mathscr{S}$ and Q a cost threshold. Finally, let k be an integer number. Given any subset of vertices $U \subseteq V$, define the set of subsets in $\mathscr{S}$ intersecting at least one of these vertices as:

$$\mathscr{S}_U = \{S \in \mathscr{S} : S \cap U \neq \emptyset\}$$

In the following, to simplify the notations when we consider the singleton $\{v\}$ we will write $\mathscr{S}_v$ instead of $\mathscr{S}_{\{v\}}$ and, for each $S \in \mathscr{S}$, we introduce the set of its vertices, denoted by $V_S = \{v \in V : S \in \mathscr{S}_v\}$. Given these definitions and notations, we define the cost of the subgraph of G induced by vertices in U as follows:

$$c_U = \sum_{S \in \mathscr{S}_U} q_S$$

The HAP requires to partition G into at most k vertex-disjoint connected subgraphs $G_i = (U_i, E_i)$ such that the cost of G_i does not exceed Q for all i and the total cost

$$\phi = \sum_i c_{U_i} - \sum_{S \in \mathscr{S}} q_S \tag{5.1}$$

is minimum. Notice that, by minimizing ϕ we minimize the cost of the subsets that are split among different subgraphs and, since $\sum_{S \in \mathscr{S}} q_S$ is a constant term, we can remove it from the definition of ϕ, without modifying the relationship occurring between any two feasible solutions.

As described in Section 5.1, the HAP arises from a practical requirement of partitioning the province of Milan into "homogeneous areas". In that case, vertices correspond to towns, edges to pairs of adjacent towns, each subset $S \in \mathscr{S}$ represents an activity involving a subset of towns and q_S represents the amount of hours that a province officer need to work in order to be trained on activity corresponding to S.

In Figure 5.1 we report a sample instance and some of its solutions. The top-left subfigure 5.1 (a) provides a graph G with 7 vertices and a subset collection $\mathscr{S}$ containing three subsets: $S_1 = \{1,6,7\}$ with cost $q_{S_1} = 11$, $S_2 = \{1,2,3,4\}$ with cost $q_{S_2} = 10$ and $S_3 = \{3,4,5\}$ with cost $q_{S_3} = 9$. In this sample we consider a cost threshold equal to $Q = 25$ and a maximum number of subgraphs $k = 3$. If the nodes could be partitioned such that each set in $\mathscr{S}$ is intersected by only one subgraph, the overall cost would hit the theoretical lower bound $\phi = 0$. However, on this particular sample instance, this theoretical lower bound cannot be achieved since the cost threshold forces us to split at least one subset in $\mathscr{S}$. The top-right subfigure 5.1(b) depicts an optimal solution, with two subgraphs induced by $U_1 = \{1,6,7\}$ and $U_2 = \{2,3,4,5\}$. Its cost is equal to $\phi = q_{S_1} + 2q_{S_2} + q_{S_3} - \sum_{S \in \mathscr{S}} q_S = q_{S_2} = 10$, because subset S_2 is the only subset intersected by both the subgraphs. The bottom-left subfigure 5.1(c) depicts a suboptimal solution with three subgraphs induced by $U_1 = \{1\}$, $U_2 = \{2,3,4,5\}$ and $U_3 = \{6,7\}$. Its cost is equal to $\phi = q_{S_1} + q_{S_2} = 21$ since both S_1 and S_2 are intersected by two subgraphs. The bottom-right subfigure 5.1(d) depicts a solution that is unfeasible since the subgraph induced by $U_1 = \{1,2,3,4,5\}$ is associated with a cost $C_{U_1} = q_{S_1} + q_{S_2} + q_{S_3} = 30$ greater than cost threshold $Q = 25$

5.3 Computational complexity

Theorem 2. *It is $\mathscr{NP}$-complete to determine whether a given instance of the HAP with $k = 2$ has optimum lower than a given threshold, even if: a) all subsets in $\mathscr{S}$ have cardinality not larger than two b) the connection constraint is relaxed.*

Proof. The proof is based on a reduction from the decision version of the *equicut* problem, which is defined as follows. Let $G' = (V', E')$ be an undirected graph, $c : E' \rightarrow \mathbb{N}$ a cost function defined on the edges, ϕ' a positive integer. The decision version of the equicut problem amounts to determining whether there exists a partition of V' into two disjoint subgraphs such that the number of vertices in the two subgraphs differs at most by one, and the sum of the costs for the edges whose extreme vertices belong to different subgraphs does not exceed ϕ'.

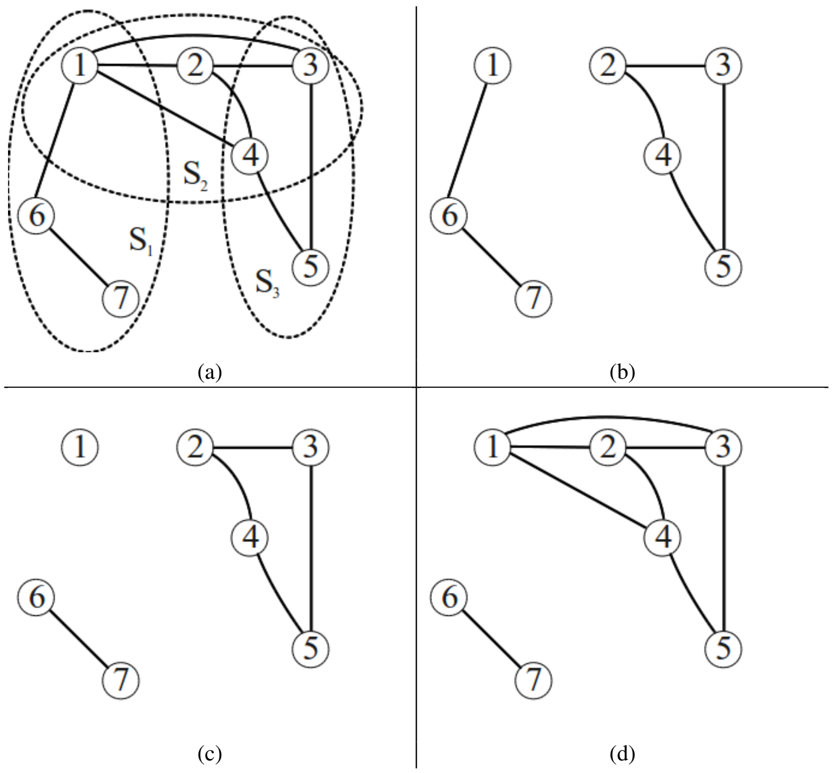


Fig. 5.1 a) sample instance having $q_{S_1} = 11, q_{S_2} = 10, q_{S_3} = 9$ and $Q = 25$, b) optimal solution associated with objective function value $\phi = 10$ c) sub-optimal solution associated with objective function value $\phi = 21$ d) a solution that is unfeasible since $q_{S_1} + q_{S_2} + q_{S_3} = 30 > Q = 25$.

We now show that, given any instance of the equicut problem, it is possible to build an instance of the HAP with no connectivity constraint, such that their optimal solutions correspond one-to-one. First, the set V coincides with V'. Then, for each edge $e = \{u, v\} \in E'$ introduce a subset $S_e \in \mathscr{S}$ including only vertices u and v, with a cost q_{S_e} equal to c_{uv}. As well, for each vertex $v \in V$ introduce another subset $S_v \in \mathscr{S}$ including only v, with a cost $q_{S_v} = (|E'|+1)\, c_{\max}$, where $c_{\max} = \max_{\{u,v\} \in E'} c_{uv}$. Set $k = 2$ and the cost threshold to $Q = \{(|E'|+1)\lceil |V'|/2 \rceil + |E'|\}\, c_{\max}$.

Now consider the instance of the HAP thus obtained. The cost threshold forces both subgraphs to include exactly $|V'|/2$ vertices if $|V'|$ is even, $\lceil |V'|/2 \rceil$ and $\lfloor |V'|/2 \rfloor$ vertices if odd. Any unbalanced partition, in fact, would have a shore with at least $(\lceil |V'|/2 \rceil + 1)$ vertices, with a cost at least $(|E'|+1)\, c_{\max}\, (\lceil |V'|/2 \rceil + 1)$, which is strictly larger than Q. On the contrary, all balanced partitions are feasible, since the cost of each shore is $(|E'|+1)\, c_{\max} |V'|/2$ plus the sum of the costs of the subsets split in two different subgraphs. This second term is the sum of the edge costs for all edges whose extreme vertices fall in different subgraphs; such edges are at most $|E'|$ and their cost is at most $c_{\max}$. Thus, the feasible solutions of both problems correspond to the partitions of the vertices into two subgraphs of equal, or nearly equal, cardinality.

The cost of the *equicut* solution is the sum of the costs for the edges whose extreme vertices belong to different subgraphs. The cost of the HAP solution with respect to function ϕ is the sum of the costs q_S for the subsets containing vertices in different subgraphs. Since the two costs are identical, the optimal solutions of the two problems correspond one-to-one.

5.4 Instances

In the following we describe the four different benchmarks we will use through all this chapter to test the algorithms developed to solve the HAP. We choose to introduce here the benchmarks that we use in the remaining part of the chapter in order to avoid the introduction of a final section containing a too large and too complex comparison of many different variants of each algorithm we developed. We decide to compare variants of a given method immediately after its description. This strategy allows us to immediately discriminate which variants are effective and which are not and, as a consequence, the reader can focus his attention on the most promising variants. These benchmarks are also used in Section 5.5 to investigate the relationship between the HAP and the standard GPPs.

5.4.1 Real world instances

We consider two different real world instances respectively derived from the province of Milan and the province of Monza. The instance derived from Milan has 134 vertices and 774 subsets, while the instance derived from Monza is smaller and has 55 vertices and 426 subsets. In both real world instances the adjacency graph G derives from geographical data. The maximum number k of subgraphs, the subsets $\mathscr{S}$, their costs $\{q_S : S \in \mathscr{S}\}$ and the cost threshold Q have been provided by the staff of the corresponding province.

5.4.2 Benchmark `A` *- the realistic instances*

The instances e belonging to benchmark `A` have been generated extracting as follows a subset of towns from the graph obtained by unifying the two graphs associated with Milan and Monza that are geographically adjacent. First, we selected, as seed towns, the five towns involved in the most costly activities, namely the ones associated with the largest values of $\sum_{S \in \mathscr{S}_v} q_S$. Starting from each seed town, we extracted the first n vertices found during a breadth- first visit of the adjacency graph, setting $n = 50$, 60, 70, 80 and 90. This method produced 5 different instances with a shape reasonably similar to a standard province and centered on a reasonable main town. We include in each instance all the subsets associated with the activities in which the extracted towns are involved. The cost threshold Q is equal to the one associated with the whole province and we use the heuristic *VLNS-TS*, described in Section 5.8.2, to identify a reasonable value for the maximum number k of subgraphs. Overall, this benchmark contains 25 instances.

5.4.3 Benchmark `B` - random instances with realistic subset structure

The instances in benchmark `B` have been generated in order to mirror the real world ones. To define G we generate a random planar graph using the strategy described in the following. First, we uniformly generate n points in a euclidean 100×100 grid. Then we build a triangulation of these points: we consider, in turn, all pairs of points by non decreasing distances, and we draw the corresponding segment if and only if it does not cross the previous ones. Each point turns into a node and each segment turns into an edge of graph G.

The number of subsets in $\mathscr{S}$ and their costs have been generated by a quite complex mechanism that tries to mimic the distribution of the subsets in the real world instances. Since the structure of the real world instances is based on the distribution of activities among towns (see Section 5.2), to generate the subsets belonging to $\mathscr{S}$, we first generate the activities and their costs. Given the number of nodes, n, the number of activities is set to $2n$, and the maximum number of subgraphs is set to $k = n/5$. The set of activities is randomly generated and the average number of activities associated with each town is set to $2\alpha n$ with $\alpha \in \{0.05; 0.1\}$. To fix this value, we analyzed the real world instances: the average number of activities for each town is ≈ 0.03 for Milan and ≈ 0.06 for Monza.

The cost of each activity a is defined by two typologies of tasks that an officer supporting a town involved in a must provide:

- The elementary tasks which are performed just once for each town in the area associated with the same officer, they require a fixed amount of work w_a.
- The elementary tasks that need to be repeated for each town in a given area, they are associated with a variable amount of work π_a that needs to be done for each town of a given area involved in a.

The structure of the activities and the amount of work they require can be modeled as a HAP instance by correctly defining the structure and the cost of the subsets belonging to $\mathscr{S}$: for each town v (that corresponds to a vertex in the considered HAP instance) we introduce a singleton subset $S_v = \{v\} \in \mathscr{S}$ associated with a cost $q_{S_v} = \sum_a \pi_a$ where the previous sum is defined on all the activities in which the town

v is involved and, for each activity a, we introduce a subset $S_a \in \mathscr{S}$ containing all the towns involved in a and having $q_{S_a} = w_a$.

For each activity a, the fixed cost w_a has been generated uniformly in $[0,100]$, while π_a have been generated either in $[0;10]$ or in $[0;100]$. In this way we have obtained two different classes of instances: when $\pi_a \in [0;10]$, the cost of each feasible subgraph depends more strongly on the fixed costs than on the variable costs, when $\pi_a \in [0;100]$ the cost threshold constraint equally depends on fixed and variable costs.

Starting from the costs of the subsets belonging to $\mathscr{S}$, we defined the cost threshold Q as follows:

$$Q = \beta \, \frac{\sum\limits_{S \in \mathscr{S}} q_S \min(k, |V_S|)}{k}$$

which is a rough estimate of the average cost of a feasible subgraph. Note that, changing coefficient $\beta \in \{1.00, 1.15\}$, we can produce instances having tight or loose cost threshold constraint.

Overall, for benchmark `B` we generated 88 instances having n that ranges from 20 to 70 by steps of 5, two different sizes for each $\mathscr{S}_v$ (i.e. $\alpha \in \{0.05, 0.1\}$), two distributions of variable costs (i.e. $\pi_a \in [0,10]$ or $\pi_a \in [0,100]$), and two levels of cost threshold (i.e. $\beta \in \{1.00, 1.15\}$).

5.4.4 *Benchmark `C` - random instances with generic subset structure*

We generated the instances in benchmark `C` following a strategy similar to the one used for benchmark `B`. The graph G and the cost threshold Q have been generated following the procedure described in the previous section. The differences between benchmarks `B` and `C` arise in the definition of the subsets belonging to $\mathscr{S}$ and their costs. In this benchmark, we randomly generate these subsets devoting special care to guarantee that each node belongs to at least one subset and that no subset is empty. For instances having n nodes we generated $|\mathscr{S}| = 2n$ subsets, each subset, on average, has a size equal to $\alpha|\mathscr{S}|$ with $\alpha \in \{0.05, 0.1\}$ and its cost directly depends on the vertices it contains: for each $v \in V$ we define a cost w_v that is either fixed to 1 or randomly generated (with a discrete uniform distribution) in $\{1, \ldots, 10\}$ or in $\{1, \ldots, 100\}$ and we define $q_S = \sum_{v \in S} w_v$. Moreover, note that in this benchmark not all the towns are associated with a singleton subset, this subset is part of $\mathscr{S}$ only if it has been chosen by the random subset generation process.

Overall, for benchmark `C` we generated 60 instances having n that ranges from 30 to 70 by steps of 10, two different sizes for each $\mathscr{S}_v$ (i.e. $\alpha \in \{0.05, 0.1\}$), three different distributions of node cost (i.e. $\pi_v = 1$, $\pi_v \in \{1, \ldots, 10\}$ or $\pi_v \in \{1, \ldots, 100\}$), and two levels of cost threshold (i.e. $\beta \in \{1.00, 1.15\}$).

5.5 On the relationship with Graph Partitioning Problems

In this section we provide some references to the huge literature on GPPs, a small example to illustrate the specific features of the HAP and a more detailed explanation of the differences between our problem and the GPPs. Given an undirected edge-weighted graph $G = (V, E)$, the most common formulation of GPPs asks for a division of V into a given number k of nonempty, pairwise disjoint subsets, such that the edge-cut, i. e. the total weight of the edges that connect vertices in different subsets, is minimized. This basic problem admits a number of variation (see e. g. Fjallström, 1998). Several different approaches have been proposed to solve them, such as hierarchic multi-level heuristics (Sanders and Schulz, 2011), geometry-based and flow-based methods (Arora et al., 2008), genetic approaches (Kim et al., 2011), spectral methods (Donath and Hoffman, 1973), mathematical programming approaches (Fan and Pardalos, 2010), local search metaheuristics and integrated approaches (Osipov et al., 2012). The HAP differs from these classical GPPs both in the constraints and in the objective function.

Cardinality constraint

In classical GPPs, the number k of vertex-disjoint subsets is usually given, and the subsets are required to be nonempty, since their cardinalities, $n_1, \ldots, n_k$, with $\sum_{j=1}^{k} n_j = |V|$ are explicitly imposed (Guttmann-Beck and Hassin, 2000) or constrained to be approximately of the same size, (see e. g. Osipov et al., 2012). In the HAP, *empty subsets of vertices U are allowed* and k is just an upper threshold. In fact, merging two subsets into a single subset is always profitable, if the obtained subset does not exceed the cost threshold Q.

Cost threshold

In this respect, the HAP is similar to the *Node-capacitated Graph Partitioning Problem* (Ferreira et al., 1998) in which the vertices in V are weighted and the sum of the weights in each subset of the partition is limited by a capacity threshold. However, the cost threshold is managed differently in the HAP, since the cost of a subset U does not increase linearly as new vertices are included, but stepwise as new subsets $s \in \mathscr{S}$ are intersected.

Connectivity constraint

The connectivity constraint is usually not imposed in GPPs, where the edges of the graph are taken into account only when computing the objective function. Quite commonly, the edge costs model a proximity measure, and the subsets end up naturally to be connected in the optimal solution. In the HAP, on the contrary, the edges determine the feasibility of solutions, since each subset must induce a connected subgraph on G, but they have no relation with the objective function, as shown by the following example. Let graph G consists of a path of 4 vertices: $1-2-3-4$, and define the following subset collection:

$$\mathscr{S} = \{\{1\},\{2\},\{3\},\{4\},\{1,4\},\{2,3\}\}$$

each singleton subset has a cost equal to 3 (i.e. $q_{\{1\}} = q_{\{2\}} = q_{\{3\}} = q_{\{4\}} = 3$) and the remaining sets have unitary costs (i.e. $q_{\{1,4\}} = q_{\{2,3\}} = 1$). If $k = 2$ and $Q = 8$, the connectivity constraint allows a single feasible solution: $U_1 = \{1,2\}$ and $U_2 = \{3,4\}$, whose cost is $\phi = 2$, because both $\{1,4\}$ and $\{2,3\}$ have been split. This is also the worst possible situation. If the connectivity constraint is relaxed, all 6 partitions into two subsets of two vertices are feasible solutions. The best one is $U_1 = \{1,4\}$ and $U_2 = \{2,3\}$, whose cost is $\phi = 0$. So, the ratio between the optimum of the original problem and that of the relaxed one can be arbitrarily large.

To understand how the relaxation of connectivity constraint impacts on the benchmarks described in Section 5.4 we compute the optimal objective function value with and without the connectivity constraint for some of them (namely, the real world instance of Monza, the instances in benchmark `A` up to $n = 60$, the instances in benchmark `B` up to $n = 25$ and the instances in benchmark `C` up to $n = 30$). We have accurately chosen these instances in order to limit the computational resources required to solve them. The obtained results are reported in Table 5.1,Table 5.2 and Table 5.3, respectively for benchmark `A`, `B` and `C` (we report the results obtained on Monza within benchmark `A`). Considering these results, we can observe that the lower bounds obtained relaxing the connectivity constraints is, on average, strictly lower than the optimum objective value of the HAP. In particular, relaxing the connectivity constraints implies, on average, a decrement in the objective function value equal to 32.95% on Monza, equal to 6.13% for the tested instances in benchmark `A`, equal to 41.11% for the tested instances in benchmark `B` and equal to 30.82% for the tested instances in benchmark `C`. This suggests that neglecting the connectivity constraint would not provide meaningful information on the original problem and that classical methods ignoring this constraint would not provide useful solutions.

Objective function

The objective function of the classical GPPs depends linearly on the cost of the edges whose extreme vertices belong to different subsets. Sometimes, this cost is tuned by some function of the cardinality of the subsets, (see e. g. Matula and Shahrokhi, 1990). The objective function of the HAP is completely independent from the edge set E, because it depends nonlinearly on the intersections between the subsets in $\mathscr{S}$ and the sets of vertices of the subgraphs.

The following example illustrates the computation of this objective function and discusses the possibility to replace it with an auxiliary one, defined on the edges of an auxiliary complete graph, in such a way that the optimal edge-cut cost would also

Ins.	ϕ^*	ϕ^*_{relax}	$\%\Delta$
50_1	3864.21	3864.21	0.00
50_2	5035.11	5010.99	-0.48
50_3	5847.16	5030.17	-13.97
50_4	5136.93	4955.74	-3.53
50_5	5460.25	5307.18	-2.80
60_1	4438.98	4175.9	-5.93
60_2	6036.54	5371.83	-11.01
60_3	6696.81	5421.31	-19.05
60_4	5703.16	5527.75	-3.08
60_5	6121.01	6033.64	-1.43
Monza	419.21	281.07	- 32.95

Table 5.1 Comparison between the optimal objective function values of the HAP on benchmark A with (ϕ^*) and without (ϕ^*_{relax}) the connectivity constraint.

n	α	$\pi_{\max}$	β	ϕ^*	ϕ^*_{relax}	$\%\Delta$
20	0.05	10	1.00	0.00	0.00	0.00
20	0.05	10	1.15	0.00	0.00	0.00
20	0.05	100	1.00	45.53	0.00	-100.00
20	0.05	100	1.15	0.00	0.00	0.00
20	0.10	10	1.00	703.45	516.64	-26.56
20	0.10	10	1.15	556.29	399.62	-28.16
20	0.10	100	1.00	1004.64	601.07	-40.17
20	0.10	100	1.15	801.96	518.49	-35.35
25	0.05	10	1.00	170.14	51.47	-69.75
25	0.05	10	1.15	95.58	0.00	-100.00
25	0.05	100	1.00	198.86	0.00	-100.00
25	0.05	100	1.15	122.50	51.47	-57.98
25	0.10	10	1.00	1906.34	1330.24	-30.22
25	0.10	10	1.15	1478.70	1145.45	-22.54
25	0.10	100	1.00	2240.17	1720.41	-23.20
25	0.10	100	1.15	1977.81	1504.67	-23.92

Table 5.2 Comparison between the optimal objective function values of the HAP on benchmark B with (ϕ^*) and without (ϕ^*_{relax}) the connectivity constraint.

provide the optimum of the HAP. This would allow to directly apply the methods proposed in the literature to solve the GPPs, provided that the cardinality, capacity and connectivity constraints could be somehow managed correctly.

Figure 5.2(a) shows an instance of the HAP defined by a complete graph with seven vertices $V = \{1,2,\dots,7\}$. For the sake of simplicity and since in a complete graph the connectivity constraint can be neglected, in the figure we do not report any original arc. The subsets collection $\mathscr{S}$ contains all the singleton subsets and three non singleton subsets: $S_1 = \{1,3,2\}$, $S_2 = \{1,3,4,5\}$ $S_3 = \{5,6,7\}$. The costs of the subsets are defined as follows: $q_{\{1\}} = q_{\{3\}} = 44$, $q_{\{2\}} = q_{\{4\}} = 22$, $q_{\{5\}} = 52$, $q_{\{6\}} = q_{\{7\}} = 30$, $q_{S_1} = 9$, $q_{S_2} = 10$ and $q_{S_3} = 8$. The cost threshold and the maximum number of subgraphs are respectively equal to $Q = 130$ and $k = 3$. Given this instance, all the feasible solutions have at most three vertices in each subgraph.

n	α	$w_{\max}$	β	ϕ^*	ϕ^*_{relax}	$\%\Delta$
30	0.05	1	1	39	24	-38.46
30	0.05	1	1.15	36	19	-47.22
30	0.05	10	1	265	159	-40.00
30	0.05	10	1.15	234	114	-51.28
30	0.05	100	1	2487	1483	-40.37
30	0.05	100	1.15	2187	1060	-51.53
30	0.1	1	1	179	146	-18.44
30	0.1	1	1.15	145	126	-13.10
30	0.1	10	1	993	783	-21.15
30	0.1	10	1.15	798	699	-12.41
30	0.1	100	1	9343	7214	-22.79
30	0.1	100	1.15	7368	6404	-13.08

Table 5.3 Comparison between the optimal objective function values of the HAP on benchmark C with (ϕ^*) and without (ϕ^*_{relax}) the connectivity constraint.

Thus, the seven vertices are always divided into exactly $k = 3$ nonempty subgraphs and subset S_2 is always split between two subgraphs.

Figure 5.2(b) depicts the optimal solution $U_1 = \{1,2,3\}$, $U_2 = \{4,5,6\}$, $U_3 = \{7\}$, with $c_{U_1} = q_{\{1\}} + q_{\{2\}} + q_{\{3\}} + q_{S_1} + q_{S_2} = 129$, $c_{U_2} = q_{\{4\}} + q_{\{5\}} + q_{\{6\}} + q_{S_2} + q_{S_3} = 122$ and $c_{U_3} = q_{\{7\}} + q_{S_3} = 38$. The overall cost of this solution is equal to the sum of the costs of the two subsets S_2 and S_3 that are split between two different subgraphs: $\phi^* = q_{S_2} + q_{S_3} = 18$. An equivalent solution can be obtained exchanging vertices 6 and 7 between subgraphs U_2 and U_3. Figure 5.2(c) depicts the suboptimal solution $U_1 = \{1,2,3\}$, $U_2 = \{4\}$, $U_3 = \{5,6,7\}$, with $c_{U_1} = q_{\{1\}} + q_{\{2\}} + q_{\{3\}} + q_{S_1} + q_{S_2} = 129$, $c_{U_2} = q_{\{4\}} + q_{S_2} = 32$ and $c_{U_3} = q_{\{5\}} + q_{\{6\}} + q_{\{7\}} + q_{S_2} + q_3 = 130$. The overall cost of this solution is twice the cost of subset S_2 that is the only subset split among all the three subgraphs: $\phi = 2q_{S_2} = 20$. Figure 5.2(d) depicts the suboptimal solution $U_1 = \{2\}$, $U_2 = \{1,3,4\}$, $U_3 = \{5,6,7\}$, with $c_{U_1} = q_{\{2\}} + q_{S_1} = 31$, $w_{U_3} = q_{\{1\}} + q_{\{3\}} + q_{\{4\}} + q_{S_1} + q_{S_2} = 129$ and $c_{U_3} = q_{\{5\}} + q_{\{6\}} + q_{\{7\}} + q_{S_2} + q_{S_3} = 130$. The overall cost of this solution is equal to the sum of the costs of the two subsets S_1 and S_2 that are split among different subgraphs: $\phi^* = q_{S_1} + q_{S_2} = 19$.

We now try to associate to each pair of vertices u and v an auxiliary cost $c_{uv} = \sum_{S \in \mathscr{S}_u \cap \mathscr{S}_v} \alpha_S q_S$ derived from the cost of the subsets containing both u and v. Since cost c_{uv} is defined by the sum of the costs of the subsets containing at least two vertices, in the following we ignore the singleton subsets contained in $\mathscr{S}$. The aim is to estimate by c_{uv} the cost of assigning the two vertices to different subgraphs. To understand if we can simply reduce the objective function of the HAP to the objective function of the classical GPPs, we need to answer to the following question. Is there an *a priori* definition (computed without knowing the optimal partitioning) of the α_S coefficients, such that an optimal graph partitioning with respect to the edge costs would be optimal also with respect to the HAP objective function?

In the following, we consider three different strategies to define the α_S coefficients. The first one directly moves the cost of each subset $S \in \mathscr{S}$ to the cost of the pairs of vertices that are contained in S: $\alpha_S = 1$ for all $S \in \mathscr{S}$. The second and

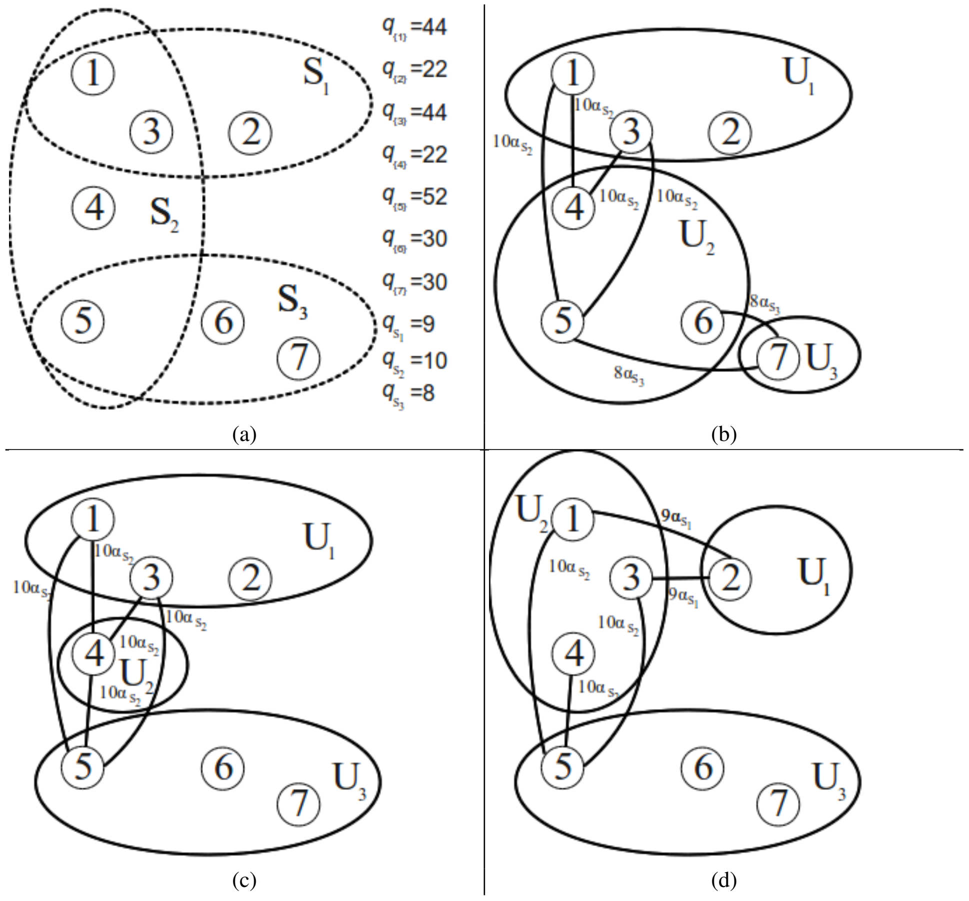


Fig. 5.2 A sample instance that shows how the objective function of the HAP cannot be simply reduced the objective function of a classical GPP.

the third strategies try to reproduce the cost q_S as the sum of the costs of the edges in a cut of V_S, without knowing the specific cut occurring in the solution. The idea is that such a cut always includes a number of edges that is between $|V_S|-1$ and $\lfloor\frac{|V_S|}{2}\rfloor\lceil\frac{|V_S|}{2}\rceil$. Correspondingly, $\alpha_S = 1/(|V_S|-1)$ or $\alpha_S = 1/\lfloor|V_S|/2\rfloor\lceil|V_S|/2\rceil$. Notice that if $|V_S| \leq 2$ all three edge strategies produces costs equivalent to q_S, because the cut includes a single edge of cost q_S. If $|V_S| \leq 3$ the second and the third strategies produce arc costs equivalent to q_S, because the cut includes a single edge of cost q_S or two edges of cost $q_S/2$. Applying the three strategies to the sample instance depicted in Figure 5.2 we obtain the following objective function values:

1. *Uniform costs* ($\alpha_{S_1} = \alpha_{S_2} = \alpha_{S_3} = 1$): The edge-cut cost is $4q_{S_2}+2q_{S_3} = 56$ for solution (b), $5q_{S_2} = 50$ for solution (c), $3q_{S_2}+2q_{S_1} = 48$ for solution (d).

2. *Costs based on the minimum edge cut cardinality* ($\alpha_S = 1/(|V_S| - 1)$): in the present case, $\alpha_{S_1} = \alpha_{S_3} = 1/2$, $\alpha_{S_2} = 1/3$. The edge-cut cost is $4/3q_{S_2} + q_{S_3} = 64/3 = 21.\bar{3}$ for solution (b), $5/3q_{S_2} = 50/3 = 16.\bar{6}$ for solution (c), $q_{S_2} + q_{S_1} = 19$ for solution (d).
3. *Costs based on the maximum edge cut cardinality* ($\alpha_S = 1/\lfloor|V_S|/2\rfloor\lceil|V_S|/2\rceil$): in the present case, $\alpha_{S_1} = \alpha_{S_3} = 1/2$, $\alpha_{S_2} = 1/4$. The edge-cut cost is $q_{S_2} + q_{S_3} = 18$ for solution (b), $5/4q_{S_2} = 50/4 = 12.5$ for solution (c), $3/4q_{S_2} + q_{S_1} = 16.5$ for solution (d).

Solution (b), which is optimal for the HAP, is the worst solution w.r.t. all three edge-cut cost strategies.

Moreover, the optimality of solution (b) w.r.t. other similar strategies would require it to be cheaper than solution (c) ($4\alpha_{S_2}q_{S_2} + 2\alpha_{S_3}q_{S_3} < 5\alpha_{S_2}q_{S_2}$ and therefore $2\alpha_{S_3}q_{S_3} < \alpha_{S_2}q_{S_2}$) and solution (d) ($4\alpha_{S_2}q_{S_2} + 2\alpha_{S_3}q_{S_3} < 3\alpha_{S_2}q_{S_2} + 2\alpha_{S_1}q_{S_1}$, hence $\alpha_{S_2}q_{S_2} + 2\alpha_{S_3}q_{S_3} < 2\alpha_{S_1}q_{S_1}$). These conditions imply $\alpha_{S_1}/\alpha_{S_3} > 2q_{S_3}/q_{S_1}$, which is a relation far from obvious and clearly *nonlocal*, i. e. it does not depend only on the subsets associated with a given pair of vertices. In fact, S_1 and S_3 have the same cardinality and no common activities; the strong difference between their coefficients must be determined by only investigating the whole instance.

To investigate whether the counterexample describes a common or a rare situation, we considered a small subset of instances extracted from benchmark B (we do not test the other benchmarks because even their smallest instances require too much computational resources to be solved considering the GPP-like objective functions) and, using an ILP solver, we optimally solved the problem obtained from HAP replacing its original objective function with the GPP-like objective functions previously introduced. Since we did not change the constraints of the considered problem, the optimal solutions obtained in these experiments are feasible HAP solutions. In Table 5.4 we report the obtained results, columns ϕ_1^*, ϕ_2^* and ϕ_3^* contain the percentage gap between the optimal solution value and the real cost of the solutions obtained using the modified objective functions based, respectively, on coefficients $\alpha_S = 1$, $\alpha_S = 1/(\mid V(S) \mid -1)$ and $\alpha_S = 1/(\lfloor|V_S|/2\rfloor\lceil|V_S|/2\rceil)$. These results show how the three different objective functions can substitute the original objective function only on the smallest instances having $\alpha = 0.05$ and $n = 20$, on the other instances, substituting the original objective function with a GPP-like one, we can easily obtain solutions that are substantially suboptimal.

These remarks on the differences between the constraints and the objective function of the HAP w.r.t. other GPPs have moved us to develop *ad hoc* methods instead of straightforward adaptations of algorithms from the literature.

5.6 Multi-commodity flow formulations

In this section we present two different compact ILP formulations to solve the HAP. Both formulations make use of multi-commodity flows to guarantee the subgraphs connectivity. At the end of this section we discuss how the order of the nodes in

n	α	$\pi_{\max}$	β	$\phi_1^*(\%)$	$\phi_2^*(\%)$	$\phi_3^*(\%)$
20	0.05	10	1.00	0.00	0.00	0.00
20	0.05	10	1.15	0.00	0.00	0.00
20	0.05	100	1.00	0.00	0.00	0.00
20	0.05	100	1.15	0.00	0.00	0.00
20	0.10	10	1.00	10.74	37.56	37.56
20	0.10	10	1.15	19.56	19.56	55.15
20	0.10	100	1.00	7.37	7.37	15.94
20	0.10	100	1.15	34.10	46.02	46.02
25	0.05	10	1.00	0.00	0.00	0.00
25	0.05	10	1.15	28.16	28.16	28.16
25	0.05	100	1.00	1.93	0.00	0.00
25	0.05	100	1.15	0.00	0.00	0.00
25	0.10	10	1.00	17.08	15.09	15.09
25	0.10	10	1.15	10.80	20.71	20.71
25	0.10	100	1.00	7.77	11.74	13.48
25	0.10	100	1.15	11.56	18.65	18.65

Table 5.4 Comparison on benchmark B among the optimal solution of the HAP and the optimal solutions of the problems obtained replacing the HAP objective function with three alternative object functions that mimic the standard GPP function.

a instance can significantly change the performance of the two formulations and how coefficients reduction techniques and valid inequalities can be added to them to reduce the required computational resources.

5.6.1 A commodity for each area

A quite natural ILP formulation for the HAP (in the following denoted by MCA) can be obtained by combining the straightforward ILP formulation for the classical GPPs (see, for example, Ferreira et al. 1996), a multi-commodity flow formulation to guarantee to connectivity of the feasible solutions (Magnanti and Wolsey, 1995) and a set of side constraints to guarantee the cost threshold respect.

We first introduce, for each $v \in V$ and $i = 1,\dots,k$, a binary variable x_v^i that is equal to 1 if node v is contained in the subgraph induced by U_i, or 0 otherwise. To define the cost of the subgraph induced by U_i, we introduce, for each $S \in \mathscr{S}$, the binary variable z_S^i that is equal to 1 if U_i intersect S, or 0 otherwise. Using these variables, we can simply state the cost associated with U_i as $\sum_{S\in\mathscr{S}} q_S z_S^i$. Finally, to guarantee the connectivity of the solutions, we introduce an auxiliary directed graph $G' = (V',A')$ derived from G replacing each edge in E with two opposite arcs, i.e. for each $\{i,j\} \in E$, A' contains both (i,j) and (j,i) and adding to V a super root $s \in V'$ directly connected with all other nodes of the graph, i.e. for each $v \in V$, the set A' contains the arc (s,v). This allows us to enforce the connectivity constraint by representing each subgraph as a rooted arborescence. Each node $v \in V$ can be the unique root of the arborescence associated with subset U_i and we model this choice using the binary variable r_v^i that is equal to 1 if and only if v is the root of the i-th arborescence. Node v can receive the flow labeled with index i directly from super root s if and only if $r_v^i = 1$, and, using the flow balance constraints, we guarantee that all the nodes of each arborescence are directly connected with its root. The quantity of flow associated with the i-th arborescence that passes through arc $a \in A'$ is denoted by the nonnegative variables f_a^i.

Thus, using the variables just introduced, the HAP can be formulated as follows:

$$\text{MCA}: \min \phi = \sum_{i=1}^{k} \sum_{S \in \mathscr{S}} q_S z_S^i - \sum_{S \in \mathscr{S}} q_S \tag{5.2a}$$

$$\sum_{i=1}^{k} x_v^i = 1 \qquad v \in V \tag{5.2b}$$

$$\sum_{S \in \mathscr{S}} q_S z_S^i \leq Q \qquad i = 1, \ldots, k \tag{5.2c}$$

$$x_v^i \leq z_S^i \qquad i = 1, \ldots, k, v \in V, S \in \mathscr{S}_v \tag{5.2d}$$

$$\sum_{v \in V} r_v^i = 1 \qquad i = 1, \ldots, k \tag{5.2e}$$

$$r_v^i \leq x_v^i \qquad i = 1, \ldots, k, v \in V \tag{5.2f}$$

$$\sum_{(s,v) \in A'} f_{sv}^i = \sum_{v \in V} x_v^i \qquad i = 1, \ldots, k \tag{5.2g}$$

$$f_{sv}^i \leq \mid V \mid r_v^i \qquad i = 1, \ldots, k, v \in V \tag{5.2h}$$

$$\sum_{(u,v) \in A'} f_{uv}^i \leq \mid V \mid x_v^i \qquad i = 1, \ldots, k, v \in V \tag{5.2i}$$

$$\sum_{(u,v) \in A'} f_{uv}^i - \sum_{(v,u) \in A'} f_{vu}^i = x_v^i \qquad i = 1, \ldots, k, v \in V \tag{5.2j}$$

$$x_v^i, r_v^i \in \{0, 1\} \qquad i = 1, \ldots, k, v \in V \tag{5.2k}$$

$$z_S^i \geq 0 \qquad i = 1, \ldots, k, S \in \mathscr{S} \tag{5.2l}$$

$$f_{uv}^i \geq 0 \qquad i = 1, \ldots, k, (u,v) \in A' \tag{5.2m}$$

Constraints (5.2b) impose that each node is contained in one and only one arborescence. Constraints (5.2c) impose the respect of the cost threshold for each arborescence. Constraints (5.2d) define which subsets in $\mathscr{S}$ are intersected by which arborescences. Constraints (5.2e) and (5.2f) impose, respectively, the uniqueness of the root of each arborescence and the presence of a node v in the i-th arborescence if v is its root. Constraints (5.2g) impose that the flow labeled with i and exiting from the super root must be equal to the number of vertices in the i-th arborescence. Constraints (5.2h) impose that the flow indexed by i can enter into v directly from s only if v is the root of the i-th arborescence. Constraints (5.2i) impose that the flow associated with the i-th arborescence passes through node v only if it belongs to the arborescence. Constraints (5.2g) and (5.2j) respectively impose the correct generation of flow associated with each arborescence and preserve the flow balance among nodes. Finally constraints (5.2k)-(5.2m) define the domain of the variables. Note that we relax the integrality constraint on variables $\{z_S^i : S \in \mathscr{S}, i = 1, \ldots, k\}$ since constraints (5.2d) combined with the objective function coefficients automatically impose $z_S^a \in \{0, 1\}$ for each $S \in \mathscr{S}, i = 1, \ldots, k$.

Symmetry breaking constraints

The set of feasible solutions of formulation MCA contains two different types of symmetry:

1. Given a feasible solution of formulation (5.2), we can obtain an equivalent solution by permuting the indexes $i = 1, \ldots, k$ of the associated arborescences. Therefore, for each feasible HAP solution, in formulation (5.2) exist $k!$ equivalent solutions.
2. Given the feasible subgraph induced by set of nodes U, each arborescence spanning U is feasible and equivalent w.r.t. formulation (5.2). As a result, given a feasible arborescence we can change its root and its structure preserving its feasibility and its objective function value.

To avoid the exponential growth of the branching tree caused by these symmetries, we developed two families of valid inequalities that make unfeasible most of the equivalent solutions. To handle the first type of symmetry (the one due to the permutation of the arborescence indexes), we introduced the *column inequalities* developed in Kaibel and Pfetsch (2008) to solve a similar problem arising in the natural formulation for the classical GPPs. The idea of the authors of that paper is to define, for each set of equivalent feasible solutions, a representative solution and to make unfeasible all the equivalent solutions different from the representative one. In our case, among all the ordering of the arborescences associated with a feasible solution, we choose the representative one as follows:

- We associate with each set of nodes U the minimum index of its nodes by setting $\mathscr{M}(U) = \min_{v \in U} \{v\}$. Notice that, in order to associate the empty set with the highest indexes, we define $\mathscr{M}(\emptyset) = \infty$.
- The representative solution is the one for which $(\mathscr{M}(U_1), \ldots, \mathscr{M}(U_k))$ defines a nondecreasing sequence.

To make unfeasible all the equivalent solutions that are not the representative one, we introduced the following inequalities:

$$\sum_{v=1}^{i-1} x_v^i = 0 \quad i = 1, \ldots, k \tag{5.3}$$

$$\sum_{i'=i}^{k-1} x_v^{i'} \leq \sum_{u=i-1}^{v-1} x_u^{i-1} \quad i = 2, \ldots, k-1 \text{ and } v = i, \ldots, n \tag{5.4}$$

The first family of constraints (5.3) imposes that all the nodes having and index lower than i cannot be assigned to the i-th arborescence: if we assign node $v < i$ to the i-th arborescence, we cannot define arborescences associated with indexes that are lower than i and such that $(\mathscr{M}(U_1), \ldots, \mathscr{M}(U_k))$ is a nondecreasing sequence .

The second family of constraints (5.4) states that a node v can be contained in the arborescence associated with an index i' greater that i only if exists a node u that can be the root of the $(i-1)$-th arborescence in a representative solution. Note that, to satisfy constraints (5.3), we have $u \geq i-1$ and, since we want to define a representative solution, we have $u \leq v-1$.

To handle the second type of symmetry (the one due to the permutation of the roots) we introduce the following inequalities:

$$x_v^i + \sum_{w=v+1}^{n} r_w^i \leq 1 \quad i = 1, \ldots, k \quad v \in V \tag{5.5}$$

These inequalities, for each $i = 1, \ldots, k$, impose that the node belonging to U_i and associated with the minimum index is the root of its arborescence.

To evaluate the impact of these inequalities on the efficiency of formulation (5.2), we tested them on a small subset of instances belonging to the three different benchmarks described in Section 5.4. We solved formulation (5.2) using CPLEX 12.4 on a PC equipped with an Intel Core2 Quad-core 2.66Ghz and 4GB of RAM, both considering and not considering column inequalities (5.3,5.4) and roots symmetry breaking constraints (5.5). The results we obtained for benchmark `A`, `B` and `C`, are respectively reported in Table 5.5, Table 5.6 and Table 5.7. In these tables, for each combination of inequalities utilization pattern (i.e. none, only column inequalities, only roots symmetry breaking constraints and both families of inequalities) and for each instance, we report the CPU time in seconds required to solve it. For each instance we set an one-hour time limit. If a given instance cannot be solved within time limit (TL) or if the required memory exceeds the available one (MO) we report the upper bound - lower bound percentage gap obtained by the solver before we stopped it (note that in Table 5.7, there is an instance for which the solver has not been able to find a feasible solution in the given amount of time, we use a dash "-" to denote this situation). Finally, to simplify the comparison among the results reported in the different columns, for each instance, we highlight, using a bold font, the content of the column associated with the best performance.

Analyzing these results, it is clear that the tested inequalities are very effective in improving the formulation strength. In fact, a great part of the instances cannot be solved without them and even when we cannot solve the instances within time limit (in particular, if we consider the hard instances in benchmark `C`), by including these inequalities, we can significantly tighten the upper bound - lower bound gaps.

INS	None	(5.3,5.4)	(5.5)	(5.3-5.5)
50_1	1784	1803	**17**	98
50_2	1305	1880	**306**	506
50_3	TL(24.96)	TL(13.91)	**493**	887
50_4	2473	1876	202	**138**
50_5	TL(1.73)	1836	**168**	444
60_1	TL(18.34)	TL(10.94)	MO(14.79)	**687**
60_2	TL(21.19)	TL(28.09)	1110	**20**
60_3	TL(25.80)	TL(18.81)	2429	**95**
60_4	TL(45.50)	TL(32.35)	1011	**43**
60_5	TL(8.50)	TL(15.28)	**306**	366

Table 5.5 Comparison between the time required to optimally solve formulation MCA (5.2) on instances in benchmark `A` with and without the symmetry breaking constraints.

n	α	$\pi_{\max}$	β	None	(5.3,5.4)	(5.5)	(5.3-5.5)
20	0.05	10	1.00	3	**0**	2	**0**
20	0.05	10	1.15	1	**0**	**0**	**0**
20	0.05	100	1.00	653	263	15	**13**
20	0.05	100	1.15	1	**0**	2	**0**
20	0.10	10	1.00	608	127	42	**17**
20	0.10	10	1.15	69	31	7	**6**
20	0.10	100	1.00	TL(13.00)	1086	498	**46**
20	0.10	100	1.15	2164	420	127	**25**
25	0.05	10	1.00	MO(147.99)	3251	496	**22**
25	0.05	10	1.15	TL(106.30)	556	160	**15**
25	0.05	100	1.00	TL(62.33)	2287	135	**24**
25	0.05	100	1.15	TL(11.99)	441	229	**14**
25	0.10	10	1.00	TL(33.57)	TL(16.42)	TL(2.42)	**137**
25	0.10	10	1.15	TL(3.56)	1136	118	**25**
25	0.10	100	1.00	TL(39.18)	TL(27.08)	TL(2.11)	**383**
25	0.10	100	1.15	MO(75.29)	TL(20.45)	3200	**179**

Table 5.6 Comparison between the time required to optimally solve formulation MCA (5.2) on instances in benchmark `B` with and without the symmetry breaking constraints.

n	α	$w_{\max}$	β	None	(5.3,5.4)	(5.5)	(5.3-5.5)
30	0.05	1	1.00	TO(-)	MO(135.12)	TL(95.66)	**MO(82.37)**
30	0.05	1	1.15	MO(211.04)	MO(191.68)	TL(63.35)	**MO(54.37)**
30	0.05	10	1.00	TL(128.42)	TL(151.82)	TL(85.86)	**TL(55.99)**
30	0.05	10	1.15	TL(125.59)	TL(125.95)	TL(93.90)	**TL(70.56)**
30	0.05	100	1.00	TL(110.46)	TL(125.15)	TL(68.39)	**TL(67.40)**
30	0.05	100	1.15	TL(125.90)	TL(103.76)	TL(82.07)	**TL(73.87)**
30	0.10	1	1.00	MO(279.27)	MO(147.78)	MO(135.72)	**MO(67.98)**
30	0.10	1	1.15	MO(111.01)	MO(61.27)	MO(46.56)	**1053**
30	0.10	10	1.00	MO(179.88)	MO(126.85)	MO(141.25)	**MO(54.66)**
30	0.10	10	1.15	MO(151.36)	MO(90.65)	TL(10.91)	**940**
30	0.10	100	1.00	MO(293.32)	MO(152.17)	MO(72.17)	**MO(32.37)**
30	0.10	100	1.15	MO(87.50)	MO(71.52)	1602	**1062**

Table 5.7 Comparison between the time required to optimally solve formulation MCA (5.2) on instances in benchmark `C` with and without the symmetry breaking constraints.

5.6.2 A commodity for each node

A different way to overcome the symmetry problems afflicting the natural formulation MCA (5.2) can be obtained applying the procedure used in (Campêlo et al., 2008) to solve the graph coloring problem. The main idea behind this new formulation (in the following denoted by MCN) is to associate a single commodity with each vertex $\ell \in V$: by correctly balancing the flow associated with commodity ℓ, we impose the connectivity of the (eventually empty) arborescence rooted in ℓ and spanning only nodes associated with an index u greater than ℓ.

To formally describe this formulation, as before, we start introducing the auxiliary directed graph $G' = (V, A)$ derived from G, replacing each edge in E with two opposite arcs. However, differently from what described for formulation MCA (5.2), in this case G' does not contain a super root node. In order to guarantee that ℓ is the vertex having the minimum index in the arborescence rooted in it, we impose that the arborescence rooted in ℓ is defined on the reduced subgraph $G^\ell = \left(V^\ell, A^\ell\right)$, where $V^\ell = \{v \in V : v \geq \ell\}$ and $A^\ell = \left\{(u,v) \in A' : u, v \in V^\ell\right\}$. In Figure 5.3 we show some of the subgraphs G^ℓ extracted from the sample instance reported in Figure 5.1. Notice that while the number of nodes in V^ℓ decreases linearly when ℓ increases, the number of nodes in the connected component containing ℓ (and, consequently, the maximum size of an arborescence rooted in ℓ) shows a nonlinear decreasing trend. In order to define formulation MCN, we introduce variables x_v^ℓ that is equal to 1 if the arborescence rooted in ℓ includes $v \in V^\ell$ (0 otherwise), z_S^ℓ that is equal to 1 if the arborescence rooted in ℓ intersects $S \in \mathscr{S}$ (0 otherwise), and f_{uv}^ℓ that denotes the flow, generated by node $\ell \in V$, passing through $(u,v) \in A^\ell$.

Given these definitions and variables, we can now state the following formulation:

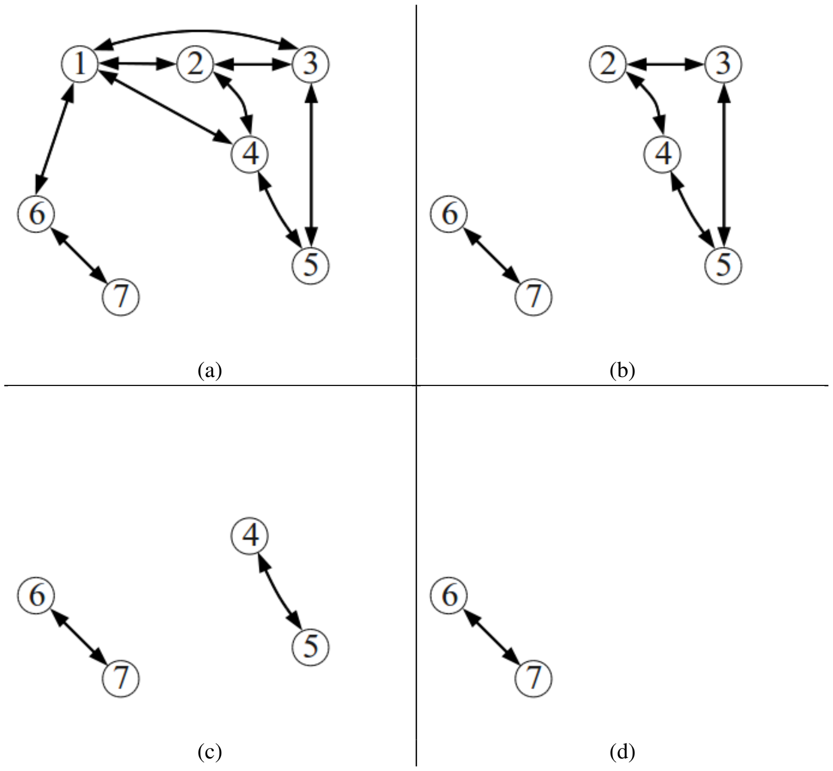


Fig. 5.3 Some G^ℓ subgraphs extracted from the sample instance in Figure 5.1 a) G^1, the connected component containing the vertex $\ell = 1$ has 7 nodes b) G^2, the connected component containing the vertex $\ell = 2$ has 6 nodes c) G^4, the connected component containing the vertex $\ell = 4$ has 2 nodes d) G^6, the connected component containing the vertex $\ell = 6$ has 2 nodes.

$$\text{MCN}: \min \phi = \sum_{S\in\mathscr{S}} \sum_{\ell\in V} q_S z_S^\ell - \sum_{S\in\mathscr{S}} q_S \tag{5.6a}$$

$$\sum_{\ell\in V: v\in V^\ell} x_v^\ell = 1 \qquad v \in V \tag{5.6b}$$

$$\sum_{\ell\in V} x_\ell^\ell \le k \tag{5.6c}$$

$$\sum_{S\in\mathscr{S}} q_S z_S^\ell \le Q x_\ell^\ell \qquad \ell \in V \tag{5.6d}$$

$$x_v^\ell \le z_S^\ell \qquad \ell \in V, v \in V^\ell, S \in \mathscr{S}_v \tag{5.6e}$$

$$x_v^\ell \le x_\ell^\ell \qquad \ell \in V, v \in V^\ell \tag{5.6f}$$

$$\sum_{(\ell,v)\in A^\ell} f_{\ell v}^\ell = \sum_{v\in V^\ell\setminus\{\ell\}} x_v^\ell \qquad \ell \in V \tag{5.6g}$$

$$\sum_{(u,v)\in A^\ell} f_{uv}^\ell \le \left(\left|V^\ell\right| - 1\right) x_v^\ell \qquad \ell \in V, v \in V^\ell \setminus \{\ell\} \tag{5.6h}$$

$$\sum_{(u,v)\in A^\ell} f_{uv}^\ell - \sum_{(v,w)\in A^\ell} f_{vw}^\ell = x_v^\ell \qquad \ell \in V, v \in V^\ell \setminus \{\ell\} \tag{5.6i}$$

$$x_v^\ell \in \{0,1\} \qquad \ell \in V, v \in V^\ell \tag{5.6j}$$

$$z_S^\ell \ge 0 \qquad S \in \mathscr{S}, \ell \in V \tag{5.6k}$$

$$f_{uv}^\ell \ge 0 \qquad \ell \in V, (u,v) \in A^\ell \tag{5.6l}$$

Constraints (5.6b) state that each vertex belongs to exactly one arborescence. Constraint (5.6c), since $x_\ell^\ell = 1$ if and only if solution contains an arborescence rooted in ℓ, imposes the correct number of arborescences. Constraints (5.6d) state that if the arborescence rooted in ℓ is not empty, its cost must not exceed Q. Constraints (5.6e) state that, if a node v is assigned to an arborescence, all subsets $S \in \mathscr{S}$ which contain v contribute to the cost of the arborescence. Constraints (5.6f) state that if a node belongs to arborescence ℓ, node ℓ is the root of the arborescence. Constraints (5.6g) state that, if an arborescence includes other nodes besides the root, then the root must generate the right amount of flow (one less than the number of nodes contained in the arborescence). Constraints (5.6h) state that, if a node receives flow ℓ, it must belong to arborescence ℓ. Coefficient $\left|V^\ell\right|$ can be replaced by any upper bound on the number of the nodes in arborescence ℓ, since each node absorbs exactly one unit of flow. As discussed in the following, tighter values should be preferred because they strengthen the LP-Relaxation of the model. Constraints (5.6i) guarantee the conservation of flow, while the following ones impose integrality or nonnegativity on the decision variables. Notice that, since variables $\left\{x_v^\ell : \ell \in V, v \in V^\ell\right\}$ are binary, constraints (5.6e) and the objective function trivially guarantee that also variables $\left\{z_S^\ell : \ell \in V, S \in \mathscr{S}\right\}$are binary.

Ordering of nodes in the MCN formulation

The order in which the vertices are considered in formulation MCN (5.6) has a great impact on the time required to solve it. As we discussed in the previous section, each feasible arborescence rooted in $\ell \in V$ is contained in the strongly connected component of G^ℓ to which ℓ belongs. Thus, the number of feasible arborescences rooted in ℓ depends directly on the size of this component and its size depends on the order in which the vertices are defined. As example, consider the sample instance in Figure 5.1 and the G^ℓ graphs derived from it and depicted in Figure 5.3: the size of the strongly connected component containing ℓ follows the sequence $(7,4,3,2,1,2,1)$ when ℓ increases from 1 to 7. Now, what happens if we reorder the vertices in V by applying to G the graph isomorphism that exchanges the label of vertex v with label $(n - v + 1)$ (i.e. we reverse the order of the vertices)? In Figure 5.4 we report some of the subgraphs G^ℓ obtained from the reordered graph. When we consider it, the size of the connected component containing ℓ follows the monotonically decreasing sequence $(7,6,5,4,3,2,1)$: for each $\ell \in V$ the size of the connected component containing ℓ hits the upper bound $(n - \ell + 1)$.

Intuitively, if we reorder the vertices in V in such a way that the number of vertices in each strongly connected component is minimized, we decrease the number of feasible arborescences in each subgraph G^ℓ and we both increase the number of valid *variable fixing constraints* (5.12) (see the following Section 5.6.3) and improve the LP-Relaxation of formulation MCN (5.6). To heuristically obtain this result we reorder the vertices, first by decreasing node degree, then, if the degrees of two or more vertices are equal, we order them by decreasing total cost, where the total cost of a vertex v is defined as $\sum_{S\in\mathscr{S}_v} q_S$. We computationally validate this heuristic reordering strategy in Section 5.6.5

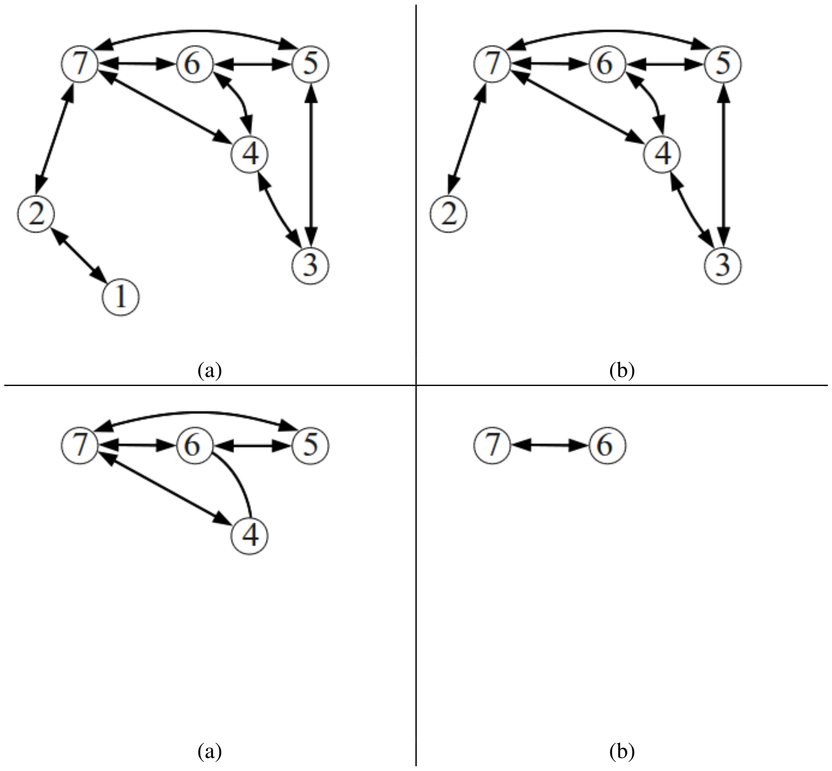


Fig. 5.4 Some G^ℓ subgraphs extracted from the instance obtained reversing the order of nodes of the sample instance depicted in Figure 5.1 a) G^1, the connected component containing the vertex $\ell = 1$ has $(n-\ell+1) = 7$ nodes b) G^2, the connected component containing the vertex $\ell = 2$ has $(n-\ell+1) = 6$ nodes c) G^4, the connected component containing the vertex $\ell = 4$ has $(n-\ell+1) = 4$ nodes d) G^6, the connected component containing the vertex $\ell = 6$ has $(n-\ell+1) = 2$ nodes.

5.6.3 *Valid inequalities and big-M reductions*

Fixing the root of a subgraph, either by using variables $\left\{r_v^i : v \in V, i = 1,\ldots,k\right\}$ and the symmetry breaking constraints (5.3-5.5) as in MCA (5.2) or by associating with each node an arborescence as in MCN (5.6), allows us to improve the proposed formulations both by introducing new valid inequalities and by tightening the big-M coefficients contained in the formulations.

Valid inequalities

The basic idea behind the following valid logic inequalities is to define two disjoint subsets, denoted by V_{in} and V_{out}, and to focus our attention on feasible HAP solutions which contain a connected subgraph including all the vertices in V_{in} and no one of the vertices in V_{out}. Then, we compute a lower bound on the cost of such a subgraph. If the obtained lower bound exceeds the cost threshold, all the solutions containing all the vertices in V_{in} and no one of the vertices in V_{out} are unfeasible. As a consequence, a logic inequality can be introduced to forbid them. Since the considered subgraph must be connected, a lower bound on its cost can be obtained by computing, on a suitable weighted graph, the minimum-weight path connecting any pair of vertices in V_{in} without using vertices in V_{out} and by selecting the computed path associated with the maximum weight.

Proposition 1. *Let $V_{\text{in}}, V_{\text{out}} \subseteq V$ such that $V_{\text{in}} \cap V_{\text{out}} = \emptyset$, ℓ_{in} be the vertex of minimum index in V_{in}, $\tilde{V} = \{v \in V \setminus V_{\text{out}} : v \geq \ell_{\text{in}}\}$ and $\tilde{G} = (\tilde{V}, \tilde{E})$ be the subgraph induced by $\tilde{V}$ on G, i. e. $\tilde{E} = \{(u,v) \in E : u, v \in \tilde{V}\}$. Finally, let*

$$k^{(V_{\text{in}})} = \sum_{S \in \mathscr{S}_{V_{\text{in}}}} q_S$$

and $p^{(V_{\text{in}},V_{\text{out}})}$ be an auxiliary weight function defined on $\tilde{V}$ as

$$p_t^{(V_{\text{in}},V_{\text{out}})} = \begin{cases} \sum\limits_{S \in \mathscr{S}_t \setminus \mathscr{S}_{V_{\text{in}}}} \dfrac{q_S}{|V_S \cap \tilde{V}|} & \text{for } t \in \tilde{V} \setminus V_{\text{in}} \\ 0 & \text{for } t \in V_{\text{in}} \end{cases}$$

Given any pair of vertices $s, t \in V_{\text{in}}$, if the sum of $k^{(V_{\text{in}})}$ plus the minimum weight with respect to $p^{(V_{\text{in}},V_{\text{out}})}$ of a path between s and t on graph $\tilde{G}$ exceeds the cost threshold Q, then:

1. *All feasible solutions of formulation MCA satisfy the following inequalities*

$$\sum_{v \in V_{\text{in}}} x_v^i \leq \sum_{v \in V_{\text{out}}} x_v^i + |V_{\text{in}}| - r_\ell^i \quad \text{for each } i = 1, \ldots, k \tag{5.7}$$

2. *All feasible solutions of formulation MCN satisfy the following inequality:*

$$\sum_{v \in V_{\text{in}}} x_v^\ell \leq \sum_{v \in V_{\text{out}}} x_v^\ell + |V_{\text{in}}| - 1 \tag{5.8}$$

Proof. Consider the following auxiliary combinatorial optimization problem on $\tilde{G}$:

$$\min_{U \subseteq \tilde{V}} w_U = \sum_{S \in \mathscr{S}_U} q_S \tag{5.9a}$$

$$U \supseteq V_{\text{in}} \tag{5.9b}$$

$$U \text{ induces a connected subgraph on } \tilde{G} \tag{5.9c}$$

Its feasible solutions are the connected subsets including V_{in}, with ℓ_{in} as minimum index vertex and not intersecting V_{out}. If the optimum of this auxiliary problem exceeds Q, none of the considered subsets can occur in a feasible solution of the given HAP instance. First we split the expression of the cost c_U in two parts:

$$c_U = \sum_{S \in \mathscr{S}_{V_{\text{in}}}} q_S + \sum_{S \in \mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}}} q_S$$

then, using the definition of $k^{(V_{\text{in}})}$

$$c_U = k^{(V_{\text{in}})} + \sum_{S \in \mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}}} q_S$$

we divide the cost q_S of subset S among the vertices of graph $\tilde{G}$ involved in it. Then we approximate q_S from below by considering only vertices in U.

$$\sum_{S \in \mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}}} q_S = \sum_{S \in \mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}}} \sum_{t \in V_S \cap \tilde{V}} \frac{q_S}{|V_S \cap \tilde{V}|} \geq \sum_{S \in \mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}}} \sum_{t \in U \cap V_S \cap \tilde{V}} \frac{q_S}{|V_S \cap \tilde{V}|} \tag{5.10}$$

The aim of this last step is to linearize the cost function, approximating it from below: when U includes vertices from V_S, we gradually sum fractions of q_S to the subgraph cost, instead of summing at once the whole cost q_S. In doing that, we only consider the vertices which can be included in U, i. e. those of $\tilde{V}$.

Notice that the double sum in the last term of (5.10) extends over all pairs (S,t) such that $S \in \mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}}$, $t \in U \cap \tilde{V} = U$ and $t \in V_S$. The last condition is equivalent to $S \in \mathscr{S}_t$. Therefore, the sum can be rewritten exchanging the two indices

$$\sum_{S \in \mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}}} \sum_{t \in U \cap V_S \cap \tilde{V}} \frac{q_S}{|V_S \cap \tilde{V}|} = \sum_{t \in U} \sum_{S \in \mathscr{S}_t \cap (\mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}})} \frac{q_S}{|V_S \cap \tilde{V}|}$$

since $\mathscr{S}_t \cap (\mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}}) = (\mathscr{S}_t \cap \mathscr{S}_U) \setminus \mathscr{S}_{V_{\text{in}}} = \mathscr{S}_t \setminus \mathscr{S}_{V_{\text{in}}}$

$$\sum_{S \in \mathscr{S}_U \setminus \mathscr{S}_{V_{\text{in}}}} q_S \geq \sum_{t \in U} \sum_{S \in \mathscr{S}_t \setminus \mathscr{S}_{V_{\text{in}}}} \frac{q_S}{|V_S \cap \tilde{V}|} \geq \sum_{t \in U \setminus V_{\text{in}}} \sum_{S \in \mathscr{S}_t \setminus \mathscr{S}_{V_{\text{in}}}} \frac{q_S}{|V_S \cap \tilde{V}|}$$

Hence

$$c_U \geq k^{(V_{\text{in}})} + \sum_{t \in U \setminus V_{\text{in}}} \left(\sum_{S \in \mathscr{S}_t \setminus S_{V_{\text{in}}}} \frac{q_S}{|V_S \cap \tilde{V}|} \right)$$

Using the last inequality and the definition of $p_t^{(V_{\text{in}},V_{\text{out}})}$, we can obtain the following lower bound on c_U:

$$c_U \geq k^{(V_{\text{in}})} + \sum_{t \in U \setminus V_{\text{in}}} p_t^{(V_{\text{in}},V_{\text{out}})} = k^{(V_{\text{in}})} + \sum_{t \in U} p_t^{(V_{\text{in}},V_{\text{out}})}$$

The former term represents the contribution of the vertices in V_{in} to the cost of any of the considered subsets. The latter represents an approximation from below of the contribution of each other town t to that cost. Computing the minimum of this expression is in general hard. A relaxation, however, can be obtained selecting two vertices in V_{in} and connecting them at minimum cost. The optimal solution of this relaxation is the minimum weight of a path between the two chosen vertices, and it can be computed by a simple vertex-weighted variation of Dijkstra's algorithm.

If the result exceeds Q, then any subgraph in $\tilde{G}$ including V_{in} is unfeasible and should be forbidden.

A simple way to express this condition in formulation MCA is:

$$\sum_{v \in V_{\text{in}}} x_v^i + \sum_{v \in V_{\text{out}}} \left(1 - x_v^i\right) \leq |V_{\text{in}}| + |V_{\text{out}}| - r_\ell^i$$

While in formulation MCN we can express the same condition as:

$$\sum_{v \in V_{\text{in}}} x_v^\ell + \sum_{v \in V_{\text{out}}} \left(1 - x_v^\ell\right) \leq |V_{\text{in}}| + |V_{\text{out}}| - 1$$

These two constraints are respectively equivalent to (5.7) and (5.8). □

The previous proposition allows us to generate a potentially exponential family of logic constraints that depend on a pair of vertex subsets. In practice, we only consider the following three cases:

1. *variable fixings*: $V_{\text{in}} = \{\ell, u\}$ and $V_{\text{out}} = \emptyset$. If the computed lower bound exceeds Q, applying (5.7), for formulation MCA (5.2), the inequality $x_\ell^i + x_u^i \leq 2 - r_\ell^i$ is valid for each $i = 1, \dots, k$, and applying (5.8), for formulation MCN (5.6), the inequality $x_\ell^\ell + x_u^\ell \leq 2 - 1$ is valid. Since, in formulation MCA, constraints (5.2f) impose $r_\ell^i \leq x_\ell^i$ and, in formulation MCN, constraints (5.6f) impose $x_\ell^\ell = 1$, if $x_u^\ell = 1$, we obtain the following valid inequalities, respectively, for MCA and MCN:
$$x_u^i \leq (1 - r_\ell^i) \text{ for each } i = 1, \dots, k \tag{5.11}$$
$$x_u^\ell = 0 \tag{5.12}$$
2. *binding constraints*: $V_{\text{in}} = \{\ell, u\}$ and $V_{\text{out}} = \{v\}$; if the computed lower bound exceeds Q, the inequality $x_\ell^i + x_u^i \leq x_v^i + 2 - r_\ell^i$ is valid for formulation MCA, for each $i = 1, \dots, k$, and the inequality $x_\ell^\ell + x_u^\ell \leq x_v^\ell + 2 - 1$ is valid for MCN. As before, since in formulation MCA we have $r_\ell^i \leq x_\ell^i$ and in formulation MCN constraint $x_u^\ell = 1$ imposes $x_\ell^\ell = 1$, the following inequalities are valid, respectively, for MCA and MCN:
$$x_u^i \leq x_v^i + (1 - r_\ell^i) \text{ for each } i = 1, \dots, k \tag{5.13}$$
$$x_u^\ell \leq x_v^\ell \tag{5.14}$$
3. *incompatibility constraints*: $V_{\text{in}} = \{\ell, u, v\}$ and $V_{\text{out}} = \emptyset$; if the lower bound computed exceeds the cost threshold, the inequality $x_\ell^i + x_u^i + x_v^i \leq 3 - r_v^i$ is valid for

each $i = 1, \dots, k$ in formulation MCA, and the inequality $x^\ell_\ell + x^\ell_u + x^\ell_v \le 3 - 1$ is valid for MCN. As before, using inequalities (5.2f) and (5.6f) we can obtain the following valid inequalities, respectively, for MCA and MCN:

$$x^i_u + x^i_v \le 2 - r^i_\ell \text{ for each } i = 1, \dots, k \tag{5.15}$$

$$x^\ell_u + x^\ell_v \le 1 \tag{5.16}$$

Coefficients reductions

The big-M coefficients in constraints (5.6h) can be reduced from $\left|V^\ell\right| - 1$ to a tighter, but still feasible, value $M^\ell - 1$ by computing for each $\ell \in V$ an upper bound on the number of vertices which can be feasibly assigned to the arborescence rooted in ℓ and contained in G^ℓ.

Proposition 2. *The optimum of the following formulation* (5.17) *provides an upper bound on the number of vertices which can be feasibly assigned to an arborescence rooted in ℓ and contained in G^ℓ*

$$\begin{aligned}
&\max M^\ell = \sum_{v \in V^\ell} x_v && && (5.17\text{a})\\
&\sum_{S \in \mathscr{S}} q_S z_S \le Q && && (5.17\text{b})\\
&x_\ell = 1 && && (5.17\text{c})\\
&x_v \le z_S && v \in V^\ell, S \in \mathscr{S}_v && (5.17\text{d})\\
&x_v \in \{0,1\} && v \in V^\ell && (5.17\text{e})\\
&z_S \in \{0,1\} && S \in \mathscr{S} && (5.17\text{f})
\end{aligned}$$

Proof. The problem can be obtained by relaxing all constraints concerning the flow variables in formulation MCN (5.6), that is by neglecting the connectivity requirement, and independently maximizing for each $\ell \in V$ the number of vertices which can be assigned to the subgraph rooted in ℓ.

Of course, the bound improves if one includes in formulation (5.17) the fixing, binding and incompatibility constraints previously described.

A similar formulation can be used to derive a lower bound on the number of vertices which can be feasibly assigned to each subgraph. This requires a lower bound on the overall workload in any feasible solution, e.g.

$$LB_Q = \sum_{S \in \mathscr{S}} q_S$$

. Consequently, the cost of any feasible subgraph has to be at least equal to:

$$Q_m = LB_Q - (k-1)\,Q$$

.

Proposition 3. *The optimum of the following formulation provides a lower bound on the number of vertices which can be feasibly assigned to an arborescence rooted in ℓ and contained in G^ℓ.*

$$\min m^\ell = \sum_{v \in V^\ell} x_v \tag{5.18a}$$

$$\sum_{S \in \mathscr{S}} q_S z_S \geq Q_m \tag{5.18b}$$

$$x_l^\ell = 1 \tag{5.18c}$$

$$x_v \leq z_S \qquad v \in V^\ell, S \in \mathscr{S}_v \tag{5.18d}$$

$$x_v \in \{0,1\} \qquad v \in V \tag{5.18e}$$

$$z_a \in \{0,1\} \qquad a \in A \tag{5.18f}$$

We omit the proof since it follows the same arguments used in Proposition 2. If the obtained values are greater or equal to 0 we can add the following constraints to formulation (5.6):

$$\sum_{v \in V} x_v^\ell \geq m^\ell x_\ell^\ell \qquad \ell \in V \tag{5.19}$$

We have described all the steps required to tighten the big-M coefficients considering formulation MCN. However, if we include symmetry breaking constraints (5.3-5.5) in formulation MCA, we can use the coefficients M^ℓ to tighten constraints (5.2h) as follows:

$$f_{sv}^i \leq M^i r_v^i \quad i = 1, \ldots, k, v \in V$$

The same reasonings can be applied to reuse the m^ℓ coefficient in MCA. We can impose a minimum size to the arborescences in formulation MCA, using the following constraints

$$\sum_{v \in V} x_v^i \geq \sum_{\ell \in V} m^\ell r_\ell^i \qquad i = 1, \ldots, k \tag{5.20}$$

5.6.4 Computational comparison of the two compact formulations

To compare how the valid inequalities and the coefficient reductions proposed in Section 5.6.3 impact on the two compact formulations, we solved them by means of CPLEX 12.4 on a PC equipped with an Intel Core2 Quad-core 2.66Ghz and 4GB of RAM, with and without formulations strengthenings.

In Table 5.8, Table 5.9 and Table 5.10, we report the results obtained, respectively, on benchmark `A`, `B` and `C`. In these tables, for each instance, we report the results obtained by the different formulations, using the same style used in the tables presented at the end of Section 5.6.1. We report the results obtained both including the proposed formulation strengthenings ("red." column) and not including them ("no red." column). Moreover, to evaluate the effectiveness of the big-M reductions and of the proposed valid inequalities, for each instance, we provide the following values: $\%\Delta M$ is the percentage decrement of the M^{ℓ} coefficient (computed as $(M^{\ell} - \mid V \mid)/\mid V \mid)$ averaged on all vertices $\ell \in V$, columns %Fix., %Bind. and %Inc., respectively, report the percentage of generated variable fixings, binding constraints and incompatibility constraints (see Section 5.6.3) w.r.t. the maximum number of constraints that can be generated (for example, the maximum number of fixing constraints that can be generated is $\frac{n(n-1)}{2}$ since we need to choose u and ℓ such that $u > \ell$) . We do not provide the CPU time required to compute the M^{ℓ} coefficients and to generate the valid inequalities because it is negligible (few seconds at most), and we do not report the impact of the m^{ℓ} coefficients since they are, with very rare exceptions, trivially equal to 0: the introduction of inequalities (5.19) in MCA formulation and of inequalities (5.19) in MCN is aimless.

Observing these results we can conclude that formulation MCN is the more robust one. On benchmark `C` (which is the benchmark containing the most difficult instances among the tested ones), using MCN we are not able to solve, within the time limit, only 3 instances, while using MCA we are not able to solve 9 of the 12 tested instances and 3 of these unsolved instances consumed all the available memory before reaching the time limit. When tested on the two other benchmarks (`A` and `B`), formulation MCN does not clearly dominate formulation MCA: on that instances the two formulations achieve similar performances.

When comparing the results obtained by the two formulations with and without valid inequalities and coefficient reductions, we come to different conclusions, depending on the considered formulation. For formulation MCN, the proposed strengthenings are very effective, on average, they increase the number of solved instances, decrease the required computational time and, for unsolved instances, they tighten the final upper bound-lower bound gap achieved by CPLEX. However, when applied to formulation MCA the same strengthenings have a less strong impact: on many tested instances, MCA obtains the best results without the valid inequalities and the coefficient reductions proposed in Section 5.6.3.

The average decrement achieved by tightening M_{ℓ} coefficients is always near 80%, in benchmark `B` we can see a decrement if we move from considering instances having $n = 20$ (on average, $\%\Delta M$ is equal to -77.41%) to considering instances having $n = 25$ (on average, $\%\Delta M$ is equal to -80.88%), the same pattern can be noticed in benchmark `A` if we move from $n = 50$ (on average, $\%\Delta M$ is equal to -82.91%) to $n = 60$ (on average, $\%\Delta M$ is equal to -85.57%). The number of generated valid inequalities strongly depends on the instance parameter α that describes the number of subsets in $\mathscr{S}$ associated with each vertex $v \in V$ in both benchmark `B` and benchmark C: in benchmark `B` we generated on average 484.17 inequalities for instances having $\alpha = 0.1$ (i.e. a higher average $\mathscr{S}_v$ cardinality) and 810.50 inequalities for instances having $\alpha = 0.05$ (i.e. a lower average $\mathscr{S}_v$ cardinality), in benchmark `C` we generated on average 323.23 inequalities for instances having $\alpha = 0.1$ and 495.88 inequalities for instances having $\alpha = 0.05$. The procedures that generate valid inequalities is very effective for what concerns variable fixings, for each instance we generate at least 20% of the fixings that theoretically can be generated, for some instances (especially in benchmark A) this percentage is greater than 50%. For the other two kinds of valid inequalities we do not obtain these percentages but, in this case, the number of inequalities that theoretically can be generated is higher w.r.t. the variable fixings and, as a consequence, to lower percentages reported in the tables, correspond higher absolute numbers of generated inequalities.

					MCA		MCN	
INS	%ΔM	%Fix.	%Bind.	%Inc.	(no red.)	(red.)	(no red.)	(red.)
50_1	-83.24	46.69	2.85	10.02	20	**33**	82	134
50_2	-81.40	35.84	4.43	6.52	95	**73**	75	97
50_3	-82.40	41.31	4.08	12.18	444	1196	**387**	480
50_4	-82.76	51.35	1.92	12.66	98	**59**	80	117
50_5	-84.76	58.86	1.26	17.43	**43**	51	85	81
60_1	-85.31	54.63	1.62	11.18	**366**	1159	3236	1595
60_2	-84.36	45.93	2.71	10.23	687	336	413	**332**
60_3	-85.31	50.17	2.20	13.90	**887**	2624	1990	1543
60_4	-86.42	59.89	1.34	17.40	**138**	958	407	282
60_5	-86.47	52.26	4.03	8.95	506	1997	**459**	517

Table 5.8 Comparison between the results obtained using formulation MCA (5.2) and the ones obtained using formulation MCN (5.6) on benchmark A.

								MCA		MCN	
n	α	$\pi_{\max}$	β	%ΔM	%Fix.	%Bind.	%Inc.	(no red.)	(red.)	(no red.)	(red.)
20	0.05	10	1.00	-78.00	38.42	3.77	10.88	**0**	41	1	**0**
20	0.05	10	1.15	-76.00	33.68	4.56	8.51	**0**	36	1	**0**
20	0.05	100	1.00	-78.75	37.89	3.86	11.27	13	8	3	**1**
20	0.05	100	1.15	-77.00	33.68	4.21	7.81	**0**	7	**0**	**0**
20	0.10	10	1.00	-76.75	25.26	6.32	2.50	17	**1**	16	11
20	0.10	10	1.15	-74.00	24.74	6.84	1.62	6	184	8	**5**
20	0.10	100	1.00	-80.50	28.95	5.44	7.32	46	**15**	43	24
20	0.10	100	1.15	-78.25	26.84	5.70	3.42	25	**6**	29	14
25	0.05	10	1.00	-82.08	41.33	4.35	9.41	22	25	10	**2**
25	0.05	10	1.15	-80.32	37.33	5.52	6.91	15	**1**	3	**1**
25	0.05	100	1.00	-83.20	43.00	4.30	11.72	24	47	9	**1**
25	0.05	100	1.15	-81.60	38.00	5.22	9.02	14	**0**	6	1
25	0.10	10	1.00	-79.20	29.33	5.74	2.87	137	**21**	114	67
25	0.10	10	1.15	-76.96	27.67	7.35	1.74	**25**	205	30	26
25	0.10	100	1.00	-83.04	33.00	6.09	9.30	383	**7**	143	33
25	0.10	100	1.15	-80.64	30.33	6.48	3.87	179	109	148	**60**

Table 5.9 Comparison between the results obtained using formulation MCA (5.2) and the ones obtained using formulation MCN (5.6) on benchmark B.

								MCA		MCN	
n	α	$w_{\max}$	β	$\%\Delta M$	%Fix.	%Bind.	%Inc.	(no red.)	(red.)	(no red.)	(red.)
30	0.05	1	1.00	-75.33	24.83	2.17	2.00	MO(82.37)	TL(72.22)	TL(5.35)	**571**
30	0.05	1	1.15	-78.89	21.84	5.74	5.58	MO(54.37)	MO(95.66)	TL(18.04)	**2208**
30	0.05	10	1.00	-75.00	23.45	2.71	1.75	TL(55.99)	TL(66.73)	3025	**868**
30	0.05	10	1.15	-82.00	25.75	5.94	8.92	TL(70.56)	TL(64.04)	TL(24.47)	**1525**
30	0.05	100	1.00	-78.78	24.83	2.17	2.00	TL(67.40)	TL(58.19)	TL(15.73)	**747**
30	0.05	100	1.15	-81.67	25.75	6.26	9.54	TL(73.87)	TL(83.02)	TL(3.45)	**1808**
30	0.10	1	1.00	-80.22	23.68	5.37	4.72	MO(67.98)	TL(27.71)	TL(21.20)	**TL(4.22)**
30	0.10	1	1.15	-78.78	24.83	2.17	1.92	**1053**	1307	2314	1117
30	0.10	10	1.00	-78.89	23.45	2.71	1.75	MO(54.66)	MO(30.00)	TL(34.20)	**TL(25.18)**
30	0.10	10	1.15	-81.78	25.52	6.43	9.70	**940**	3011	1374	1246
30	0.10	100	1.00	-78.89	22.07	5.64	5.57	MO(32.37)	MO(57.01)	**TL(11.39)**	TL(32.00)
30	0.10	100	1.15	-75.44	24.83	2.17	1.92	**1062**	1616	2902	1466

Table 5.10 Comparison between the results obtained using formulation MCA (5.2) and the ones obtained using formulation MCN (5.6) on benchmark `C`.

In the final experiment described in this section, we want to investigate the impact of the different ordering of the vertices in determining the performance obtained by formulation MCN. All the results reported so far have been obtained testing instances in which the vertices have been ordered adopting the strategy proposed in Section 5.6.2. To evaluate the impact of this strategy, we generate a different order of vertices by uniformly extracting a random permutation of the vertices and using it to generate new instances. In Table 5.11 we report, for each tested instance, the time required by CPLEX to solve formulation MCN both with a random order of the vertices (column "Rand.") and with the order of the vertices described in Section 5.6.2 (column "Ord."). Analyzing these results, it is immediately clear the importance of a correct order of the vertices, if we use a random order we cannot solve 9 instances, while using the order of vertices previously proposed we cannot solve only 3 instances. Moreover, only for one of the instances in benchmark `B` and one of the instances in benchmark `C` the time required to solve the ordered instance is greater than the time required to solve the corresponding randomized instance, for the other instances the time required to solve the ordered instances is significantly less than the time required to solve the corresponding randomized instance.

5.6.5 Scalability of compact formulations

In the previous section we show that formulation MCN (5.6) is more robust and efficient than formulation MCA (5.2). To complete our analysis of compact formulations for the HAP, we tested, on the same PC used in the previous sections, formulation MCN on bigger instances using the valid inequalities and the coefficient reductions proposed in Section 5.6.3. We impose the same one-hour time limit imposed in the previous experiments, and we tested MCN on all the three benchmarks, gradually incrementing the number n of vertices of the considered instances.

Benchmark A

INS	Rand.	Ord.
50_1	185	**134**
50_2	498	**97**
50_3	2408	**480**
50_4	932	**117**
50_5	3237	**81**
60_1	TL(45.92)	**1595**
60_2	TL(20.68)	**332**
60_3	2377	**1543**
60_4	TL(49.62)	**282**
60_5	TL(-)	**517**

Benchmark B

n	α	$\pi_{\max}$	β	Rand.	Ord.
20	0.05	10	1	1	**0**
20	0.05	10	1.15	**0**	**0**
20	0.05	100	1	4	**1**
20	0.05	100	1.15	**0**	**0**
20	0.1	10	1	13	**11**
20	0.1	10	1.15	**2**	5
20	0.1	100	1	43	**24**
20	0.1	100	1.15	22	**14**
25	0.05	10	1	37	**2**
25	0.05	10	1.15	5	**1**
25	0.05	100	1	33	**1**
25	0.05	100	1.15	20	**1**
25	0.1	10	1	269	**67**
25	0.1	10	1.15	102	**26**
25	0.1	100	1	523	**33**
25	0.1	100	1.15	457	**60**

Benchmark C

n	α	$w_{\max}$	β	Rand.	Ord.
30	0.05	1	1	TL(23.85)	**571**
30	0.05	1	1.15	TL(27.65)	**2208**
30	0.05	10	1	TL(7.16)	**868**
30	0.05	10	1.15	3208	**1525**
30	0.05	100	1	2041	**747**
30	0.05	100	1.15	3287	**1808**
30	0.1	1	1	TL(21.11)	**TL(4.22)**
30	0.1	1	1.15	1554	**1117**
30	0.1	10	1	**TL(10.56)**	TL(25.18)
30	0.1	10	1.15	1371	**1246**
30	0.1	100	1	**TL(15.80)**	TL(32.00)
30	0.1	100	1.15	**1373**	1466

Table 5.11 Comparison between the results obtained with formulation MCN (5.6) on the three benchmarks using the instances with ordered vertices (Ord.) and the instances where the order of vertices has been randomized (Rand.).

As soon as we are not able to solve any instance for a given benchmark and for a given number of vertices, we start testing the next benchmark. The results we obtained during this experiment are synthetically reported in Table 5.12. The instances in benchmark C, as previously observed, seem to be the hardest ones: in one hour of CPU time, we are able to solve only some instances having at most $n = 30$ nodes. It is interesting that in both benchmarks B and C, the sparsest instances (i.e. the ones generated setting $\alpha = 0.05$) require less computing resources w.r.t. the densest instances (i.e. the ones generated setting $\alpha = 0.10$). The correlation between the density of an instance and its difficulty is confirmed by the results obtained on benchmark A that is characterized by a low density degree: in this case we can easily solve all the instances up to $n = 60$ and some instances having $n = 70$ and $n = 80$. The gap increases steeply when passing from 30 to $35-40$ vertices for benchmarks B and C and from 70 to 80 vertices for benchmark A. In these cases, the branching tree exploration requires to analyze hundreds of thousands of branching nodes, with a non negligible memory consumption.

Since the previous results showed how limited is the size of the HAP instances that we can solve using the proposed compact formulations, in the following section, we describe a different approach based on a new extended ILP formulation for the HAP that can be effectively solved using the Column Generation method.

Benchmark A

INS	T
50_1	134
50_2	97
50_3	480
50_4	117
50_5	81
60_1	1595
60_2	332
60_3	1543
60_4	282
60_5	517
70_1	TL(18.05)
70_2	411
70_3	3315
70_4	TL(6.3)
70_5	TL(-)
80_1	TL(21.46)
80_2	1228
80_3	TL(29.72)
80_4	TL(10.46)
80_5	TL(-)

Benchmark B

n	α	π	β	T
20	0.05	10	1.00	0
20	0.05	10	1.15	0
20	0.05	100	1.00	1
20	0.05	100	1.15	0
20	0.10	10	1.00	11
20	0.10	10	1.15	5
20	0.10	100	1.00	24
20	0.10	100	1.15	14
25	0.05	10	1.00	2
25	0.05	10	1.15	1
25	0.05	100	1.00	1
25	0.05	100	1.15	1
25	0.10	10	1.00	67
25	0.10	10	1.15	26
25	0.10	100	1.00	33
25	0.10	100	1.15	60
30	0.05	10	1.00	181
30	0.05	10	1.15	312
30	0.05	100	1.00	328
30	0.05	100	1.15	186
30	0.10	10	1.00	3139
30	0.10	10	1.15	TL(13.2)
30	0.10	100	1.00	TL(13.0
30	0.10	100	1.15	TL(14.0)
35	0.05	10	1.00	TL(9.7)
35	0.05	10	1.15	TL(26.9)
35	0.05	100	1.00	2621
35	0.05	100	1.15	TL(18.7)
35	0.10	10	1.00	TL(64.6)
35	0.10	10	1.15	TL(63.3)
35	0.10	100	1.00	TL(32.7)
35	0.10	100	1.15	TL(56.0)

Benchmark C

n	α	π	β	T
30	0.05	1	1.00	571
30	0.05	1	1.15	2208
30	0.05	10	1.00	868
30	0.05	10	1.15	1525
30	0.05	100	1.00	747
30	0.05	100	1.15	1808
30	0.10	1	1.00	TL(4.22)
30	0.10	1	1.15	1117
30	0.10	10	1.00	TL(25.18)
30	0.10	10	1.15	1246
30	0.10	100	1.00	TL(32.00)
30	0.10	100	1.15	1466

Table 5.12 Scalability analysis of formulation MCN on all the three considered benchmarks.

5.7 A Column Generation approach

In this section, applying the Dantizg-Wolfe decomposition (see Chapter 2) to the compact ILP formulation MCA (5.2), we derive a new extended formulation. Using the Column Generation method we can obtain its LP-Relaxation without enumerating all its variables. As discussed in Section 3.1, the associated Pricing Problem is similar to the KPCSTP. In both problems we want to identify a connected subgraph that minimizes the difference between its cost and the prize associated with the spanned nodes. However, while in the KPCSTP the cost of a subgraph is simply defined as the cost of the used arcs, in the Pricing Problem the cost of a subgraph is defined by the cost of the subsets in $\mathscr{S}$ intersected by it. The latter cost is used also in the definition of the cost threshold constraint that can be seen as a non linear knapsack constraint since the weight of a subgraph does not depend directly on the set of the spanned nodes. Nonetheless, the two exact methods proposed in the following to solve the Pricing Problem have similarities with the ones proposed for the KPCSTP in Chapter 3. In particular, both the methods derive from the single-commodity flow formulation described in Section 3.4.1.

Similarly to what done for the MRWADC problem (see Section 4.5.3), in order to speed up the Column Generation method, we developed a Tabu Search heuristic for the PP and we describe it in Section 5.7.6. The way in which the exact and heuristic methods cooperate in order to found the optimal solution of formulation MCE_{LP} (5.24) is deeply described in Section 5.7.7.

5.7.1 Dantzig-Wolfe decomposition and extended formulation

Consider the following polytope Ω that describes the LP-Relaxation of the set containing all the feasible arborescences defined on G. This set is obtained by considering constraints (5.2c - 5.2m), i.e. all the constraints of formulation MCA (5.2) with the exception of the partitioning constraints (5.2b), and by arbitrary fixing the subgraph index $i \in \{1, \dots, k\}$:

$$\begin{aligned}\Omega = \{\ (\underline{z}, \underline{x}) : & \sum_{S \in \mathscr{S}} q_S z_S \leq Q; \quad x_v \leq z_S \text{ for each } v \in V, s \in \mathscr{S}_v; \quad \sum_{v \in V} r_v = 1; \\ & r_v \leq x_v \text{ for each } v \in V; \quad \sum_{(s,v) \in A'} f_{sv} = \sum_{v \in V} x_v; \quad f_{sv} \leq \mid V \mid r_v \text{ for each } v \in V; \\ & \sum_{(u,v) \in A'} f_{uv} \leq \mid V \mid x_v \text{ for each } v \in V; \\ & \sum_{(u,v) \in A'} f_{uv} - \sum_{(v,u) \in A'} f_{vu} = x_v \text{ for each } v \in V \\ & 0 \leq x_v \leq 1; \quad 0 \leq r_v \leq 1; \quad 0 \leq z_S \leq 1; \quad f_{uv} \geq 0 \}\end{aligned}$$

Now the LP-Relaxation of formulation MCA (5.2) can be compactly rewritten as follows:

$$\text{MCA}_{\text{LP}} : \min \phi = \sum_{i=1}^{k} \sum_{S \in \mathscr{S}} q_S z_S^i - \sum_{S \in \mathscr{S}} q_S \tag{5.21a}$$

$$\sum_{i=1}^{k} x_v^i = 1 \qquad v \in V \tag{5.21b}$$

$$(\underline{x}^i, \underline{z}^i) \in \Omega \qquad i = 1, \dots, k \tag{5.21c}$$

Since, for each $i = 1, \dots, k$, the HAP requires the integrality of variables $\underline{x}^i$ and $\underline{z}^i$, to improve the lower bound obtained by solving MCA_{LP} we can substitute, in constraints (5.21c), the polytope Ω with the convex hull of its integral points, *conv*(Ω_{IP}), obtaining the following extended formulation:

$$\text{MCE}_{\text{LP}} : \min \phi = \sum_{i=1}^{k} \sum_{S \in \mathscr{S}} q_S z_S^i - \sum_{S \in \mathscr{S}} q_S \tag{5.22a}$$

$$\sum_{i=1}^{k} x_v^i = 1 \qquad v \in V \tag{5.22b}$$

$$(\underline{x}^i, \underline{z}^i) \in conv(\Omega_{IP}) \quad i = 1, \ldots, k \tag{5.22c}$$

If we properly choose graph G (i.e. if we consider a complete graph) and subsets $\mathscr{S}$ (i.e. if we introduce a singleton subset for each vertex), we can reduce polytope Ω to the polytope describing the LP-Relaxation of a binary knapsack problem. Since it is wide known (see, for example, Martello and Toth 1990) that this polytope does not have the integrality property, also polytope Ω does not have this property. Hence, the extreme points of Ω can be fractional. When at least one of the extreme points of Ω is fractional, Ω is a proper subset of *conv*(Ω_{IP}) and the lower bound obtained solving MCE_{LP} (5.22) is stronger than the one obtained solving MCA_{LP} (5.21).

Since *conv*(Ω_{IP}) is a polytope, it does not contain any extreme ray. Thus, if we denote its extreme points as follows:

$$\mathscr{V} = \left\{ (\underline{\bar{x}}^{(1)}, \underline{\bar{z}}^{(1)}), (\underline{\bar{x}}^{(2)}, \underline{\bar{z}}^{(2)}), \ldots, (\underline{\bar{x}}^{(j)}, \underline{\bar{z}}^{(j)}), \ldots, (\underline{\bar{x}}^{(L)}, \underline{\bar{z}}^{(L)}) \right\}$$

We can express, in a compact form, all the points contained in *conv*(Ω_{IP}):

$$conv(\Omega_{IP}) = \left\{ (\underline{x}, \underline{z}) : (\underline{x}, \underline{z}) = \sum_{j=1}^{L} \lambda_j (\underline{\bar{x}}^{(j)}, \underline{\bar{z}}^{(j)}),\ \sum_{j=1}^{L} \lambda_j = 1,\ \lambda_j \geq 0 \right\}$$

Introducing this new definition of *conv*(Ω_{IP}) we can rewrite formulation (5.22) using coefficients λ as variables, obtaining:

$$\text{MCE}_{\text{LP}} : \min \phi = \sum_{i=1}^{k} \sum_{j=1}^{L} \left(\sum_{S \in \mathscr{S}} q_S z_S^{(j)} \right) \lambda_j^i - \sum_{S \in \mathscr{S}} q_S \tag{5.23a}$$

$$\sum_{i=1}^{k} \sum_{j=1}^{L} \lambda_j^i \bar{x}_v^{(j)} = 1 \qquad v \in V \tag{5.23b}$$

$$\sum_{j=1}^{L} \lambda_j^i = 1 \qquad i = 1, \ldots, k \tag{5.23c}$$

$$\lambda_j^i \geq 0 \qquad i = 1, \ldots, k,\ j = 1, \ldots, L \tag{5.23d}$$

If we denote the cost of the subgraph associated with point $(\underline{\bar{x}}^{(j)}, \underline{\bar{z}}^{(j)}) \in \Omega_{IP}$ as $\phi_j = \sum_{S \in \mathscr{S}} q_S z_S^{(j)}$ and if we define the aggregate variable $\lambda_j = \sum_{i=1}^{K} \lambda_j^i$, we can simplify, without loss of generality (see Chapter 2), the previous formulation and obtaining the final form of the the extended formulation MCE_{LP}:

$$\text{MCE}_{\text{LP}} : \min \phi = \sum_{j=1}^{L} \phi_j \lambda_j - \sum_{S \in \mathscr{S}} q_S \tag{5.24a}$$

$$\sum_{j=1}^{L} \lambda_j \bar{x}_v^{(j)} = 1 \qquad v \in V \qquad\qquad (\eta_v \text{ free }) \tag{5.24b}$$

$$\sum_{j=1}^{L} \lambda_j \leq k \qquad\qquad (\mu \geq 0) \tag{5.24c}$$

$$\lambda_j \geq 0 \qquad j = 1, \ldots, L \tag{5.24d}$$

Within parenthesis we report the dual variables associated respectively with the partitioning constraints (5.24b) and with the cardinality constraints (5.24c). Notice that MCE_{LP} does not suffer from the symmetry drawbacks previously described for MCA (see Section 5.6.1).

5.7.2 Computing the LP-Relaxation of the extended formulation

Formulation MCE_{LP} (5.24) has an exponential number of variables, it contains a variable for each extreme point of $conv(\Omega_{IP})$. Therefore, to avoid the complete enumeration of these variables, we applied a Column Generation method to solve it. In particular, formulation MCE_{LP} (5.24) plays the role of the Master Problem (MP) and we obtain the Reduced Master Problem (RMP) by initially considering only a small subset of the extreme points in $\mathscr{V}$. Then we solve the RMP and we use the optimal values of its dual variables either to generate non basic variables with negative reduced costs or to prove that the current basic solution is optimal also for the MP. This last step requires the resolution of the following Pricing Problem:

$$\text{PP} : z_{\text{PP}} = \min_{(\bar{x}, \bar{z}) \in \mathscr{V}} \sum_{S \in \mathscr{S}} q_S z_S - \sum_{v \in V} \eta_v - \mu$$

If we consider how the set $\mathscr{V}$ is defined, we can see that PP requires to identify a connected subgraph of G that respects the cost threshold Q and minimizes the reduced cost $\sum_{S \in \mathscr{S}} q_S z_S - \sum_{v \in V} \eta_v - \mu$. As shown by the following proposition, this problem is $\mathscr{N}\mathscr{P}$-hard.

Proposition 4. *The PP problem is $\mathscr{N}\mathscr{P}$-hard, even if $\eta_v \geq 0$ for each $v \in V$ and $Q = +\infty$.*

Proof. The proof is based on a reduction from the *Maximum Weight Connected Subgraph* (MWCS) problem, which is $\mathscr{N}\mathscr{P}$-hard (Ideker et al., 2002) and is defined as follows. Given a graph $\tilde{G} = (\tilde{V}, \tilde{E})$, and a weight function defined on the vertices, $w : \tilde{V} \to \mathbb{R}$, the problem requires to find a connected subgraph $\tilde{G}' = (\tilde{V}', \tilde{E}')$ of $\tilde{G}$ associated with the maximum total weight $w_{\tilde{G}'} = \sum_{v \in \tilde{V}'} w_v$.

We now show that starting from any given instance of the MWCS problem, it is possible to derive an instance of the PP, such that their optimal solutions correspond one-to-one. Graph $G = (V,E)$ coincides with $\tilde{G}$. We set $Q = +\infty$ and $\mu = 0$. For each vertex $v \in \tilde{V}$ such that $w_v < 0$, we define a singleton subset $S = \{v\} \in \mathscr{S}$ with cost $q_S = -w_v$ and we set $\eta_v = 0$. For each vertex $v \in \tilde{V}$ such that $w_v \geq 0$, we set $\eta_v = w_v$.

Each PP feasible solution $G' = (V',E')$ is a connected subgraph, and therefore is also feasible for the MWCS problem. Its cost is equal to $\sum_{v\in V':w_v<0}(-w_v) - \sum_{v\in V':w_v\geq 0} w_v = -\sum_{v\in V'} w_v$, which is the opposite of the objective function of the MWCS problem. Thus, minimizing the objective of the PP corresponds to maximizing the objective of the MWCS problem.

5.7.3 A root-independent Pricing Problem formulation

A natural ILP formulation for the PP can be easily obtained adding to polytope Ω (see Section 5.7.1) the integrality constraint on the x_v, r_v and z_S variables and introducing an objective function that minimizes the reduced cost of the selected subgraph. The obtained formulation can be stated as follows:

$$\text{PPI} : \min z_{\text{PPI}} = \sum_{S\in\mathscr{S}} q_S z_S - \sum_{v\in V} \eta_v x_v - \mu \tag{5.25a}$$

$$\sum_{S\in\mathscr{S}} q_S z_S \leq Q \tag{5.25b}$$

$$x_v \leq z_S \qquad v \in V, S \in \mathscr{S}_v \tag{5.25c}$$

$$\sum_{v\in V} r_v = 1 \tag{5.25d}$$

$$r_v \leq x_v \qquad v \in V \tag{5.25e}$$

$$\sum_{(s,v)\in A'} f_{sv} = \sum_{v\in V} x_v \tag{5.25f}$$

$$f_{sv} \leq |V| r_v \qquad v \in V \tag{5.25g}$$

$$\sum_{(u,v)\in A'} f_{uv} \leq |V| x_v \qquad v \in V \tag{5.25h}$$

$$\sum_{(u,v)\in A'} f_{uv} - \sum_{(v,u)\in A'} f_{vw} = x_v \qquad v \in V \tag{5.25i}$$

$$x_v, r_v \in \{0,1\} \qquad v \in V \tag{5.25j}$$

$$z_S \geq 0 \qquad S \in \mathscr{S} \tag{5.25k}$$

$$f_{uv} \geq 0 \qquad (u,v) \in A' \tag{5.25l}$$

The objective function is given by equation (5.25a) and it corresponds to the reduced cost of variable λ_j in formulation MCE$_{\text{LP}}$ (5.24) that is associated with the subgraph defined by variables $(\underline{x},\underline{z})$ and with the dual variables values $(\underline{\eta},\mu)$.

Constraint (5.25b) limits the cost of the arborescence. Constraints (5.25c) state that, if a node v is assigned to the arborescence, all the subsets in $\mathscr{S}$ which contain v contribute to its cost. Constraint (5.25d) guarantees that the associated arborescence has exactly one root. Constraints (5.25e) guarantee that the root node belongs to the arborescence. Constraints (5.25f - 5.25i) impose the connectivity of the arborescence. Notice that, similarly to what we have done for formulations MCA (5.2) and MCN (5.6), by introducing the objective function (5.25a) that minimizes the value of z_S, we can relax the integrality constraint on z variables

This formulation suffers from the same symmetry drawbacks on the root selection suffered by MCA: given a feasible subgraph induced by the set of nodes U, we can choose any nodes $v \in U$ to be the root of the arborescence without modifying neither feasibility nor optimality. To avoid this drawback we can use the following constraints that are similar to the ones (5.5) proposed for formulation MCA (5.2) and impose to select as root of the arborescence the node with the minimum index among the ones associated with $x_v = 1$:

$$x_v + \sum_{w=v+1}^{n} r_w \leq 1 \quad v \in V \tag{5.26}$$

5.7.4 Solving the Pricing Problem by iterative fixing the root

A different approach to solve the PP is based on the resolution, for each $\ell \in V$, of the subproblem, denoted by PP$^\ell$, that requires to find, among all the feasible arborescences rooted in ℓ that only contain nodes having index $v \geq \ell$, the one having the minimum reduced cost. If we denote by z_{PP^ℓ} the optimal objective function value of PP$^\ell$ we can obtain the optimal objective function of the PP by simply setting $z_{\text{PP}} = \min_{\ell \in V} z_{\text{PP}^\ell}$. Moreover, each arborescence that is optimal for a subproblem PP$^\ell$ such that $z_{\text{PP}} = z_{\text{PP}^\ell}$, is also optimal for the PP. Notice that using this decomposition approach we avoid the symmetry problems associated with the root selection in the PP formulation described in the previous section.

The Pricing Problem PP$^\ell$ is defined on subgraph $G^\ell = (V^\ell, A^\ell)$ introduced in Section 5.6.2 for formulation MCN (5.6.2) and it can be formulated using a set of decision variables similar to the ones used for formulation PPI (5.25). The only two differences are that in formulation PP$^\ell$ we do not have variables $\{r_v : v \in V\}$ and variables $\left\{f_{uv} : (u,v) \in A^\ell\right\}$ and $\left\{x_v : v \in V^\ell\right\}$ are now defined on G^ℓ and not on G.

$$\mathrm{PP}^\ell : \min z_{\mathrm{PP}^\ell} = \sum_{S\in\mathscr{S}} q_S z_S - \sum_{v\in V^\ell} \eta_v x_v - \mu \tag{5.27a}$$

$$\sum_{S\in\mathscr{S}} q_S z_S \le Q \tag{5.27b}$$

$$x_v \le z_S \qquad v\in V^\ell, S\in\mathscr{S}_v \tag{5.27c}$$

$$x_\ell = 1 \tag{5.27d}$$

$$\sum_{(\ell,v)\in A^\ell} f_{\ell v} = \sum_{v\in V^\ell\setminus\{\ell\}} x_v \tag{5.27e}$$

$$\sum_{(u,v)\in A^\ell} f_{uv} \le \left(|V^\ell|-1\right) x_v \qquad v\in V^\ell\setminus\{\ell\} \tag{5.27f}$$

$$\sum_{(u,v)\in A^\ell} f_{uv} - \sum_{(v,u)\in A^\ell} f_{vu} = x_v \qquad v\in V^\ell\setminus\{\ell\} \tag{5.27g}$$

$$x_v \in \{0,1\} \qquad v\in V^\ell \tag{5.27h}$$

$$z_S \ge 0 \qquad S\in\mathscr{S} \tag{5.27i}$$

$$f_{uv} \ge 0 \qquad (u,v)\in E^\ell \tag{5.27j}$$

The objective function is given by equation (5.27a) and it corresponds to the reduced cost of the variable λ_j in formulation MCE$_{\mathrm{LP}}$ (5.24) associated with the arborescence rooted in ℓ defined by variables $(\underline{x},\underline{z})$ and with the dual variables values $(\underline{\eta},\mu)$. Constraint (5.27b) limits the cost of the arborescence. Constraints (5.27c) state that, if a node v is assigned to the arborescence, all subsets $S\in\mathscr{S}$ which contain v contribute to its cost. Constraint (5.27d) guarantees that the root node belongs to the arborescence. Constraint (5.27e) sets the amount of flow generated by the root of the arborescence to be equal to the number of the other nodes belonging to it. Constraints (5.27f) state that, if a node receives one unit of flow, it must belong to the arborescence. Constraints (5.27g) guarantee the conservation of flow, while the following ones impose integrality or nonnegativity on the decision variables.

5.7.5 Valid inequalities and coefficient reductions

Both formulation PPI (5.25) and formulation PP$^\ell$ (5.27) can be strengthened by adapting the valid inequalities and the coefficient reductions procedures introduced in Section 5.6.3 for the compact formulations MCA (5.2) and MCN (5.6).

The valid inequalities (5.11-5.16) can be adapted to the two PP formulations as follows:

- *variable fixings*: given two sets $V_{\mathrm{in}} = \{\ell,u\}$ and $V_{\mathrm{out}} = \emptyset$, for which the lower bound on cost computed as described in Section 5.6.3 exceeds Q, the following inequalities are respectively valid for PPI and PP$^\ell$:

$$x_u \le (1-r_\ell) \tag{5.28}$$

$$x_u = 0 \tag{5.29}$$

- *binding constraints*: given two sets $V_{\text{in}} = \{\ell, u\}$ and $V_{\text{out}} = \{v\}$, for which the lower bound on cost computed as described in Section 5.6.3 exceeds Q, the following inequalities are respectively valid for PPI and PP$^\ell$:

$$x_u \leq x_v + (1 - r_\ell) \tag{5.30}$$

$$x_u \leq x_v \tag{5.31}$$

- *incompatibility constraints*: given two sets $V_{\text{in}} = \{\ell, u, v\}$ and $V_{\text{out}} = \emptyset$, for which the lower bound on cost computed as described in Section 5.6.3 exceeds Q, the following inequalities are respectively valid for PPI and PP$^\ell$:

$$x_u + x_v \leq 2 - r_\ell \tag{5.32}$$

$$x_u + x_v \leq 1 \tag{5.33}$$

In a similar way, we can reuse the coefficient M^ℓ and m^ℓ defined in Section 5.6.3 to tighten both PP formulations:

- In formulation PPI we can tighten constraints (5.25g) as follows:

$$f_{sv} \leq M^v r_v \qquad v \in V \tag{5.34}$$

- In formulation PP$^\ell$ we can tighten constraints (5.27f) as follows:

$$\sum_{(u,v) \in A^\ell} f^\ell_{\ell v} \leq (M^\ell - 1) x_v \tag{5.35}$$

5.7.6 A Tabu Search heuristic for the Pricing Problem

Since the Pricing Problem is $\mathscr{NP}$-hard and since, as we discuss in Section 5.7.7, the systematic application of a commercial ILP solver proved rather inefficient to solve it, we developed a Tabu Search heuristic, *PPTS*, to quickly identify negative reduced cost columns. Hence, we can limit the use of the exact solver to the cases in which *PPTS* fails to provide any column with a negative reduced cost.

Tabu Search is a well-known local search metaheuristic approach which allows the visit of nonimproving solutions. It is controlled by memory mechanisms to avoid the insurgence of cyclic behaviors. It was introduced by Glover (1986) and the interested reader can find in Glover and Laguna (1997) a detailed treatment of its applications and variants. In the following, we mainly focus our attention on the specific aspects of our implementation.

First of all, *PPTS* does not impose a fixed root node, it implicitly solves the overall Pricing Problem PP instead of the single subproblems PP^{ℓ} in which a root $\ell \in V$ is fixed. *PPTS* starts from a given feasible solution $G' = (U, E')$, which corresponds to a subgraph of G, possibly empty. We defined two simple moves: the addition (removal) of a node to (from) the current set of nodes U. At each iteration, the heuristic evaluates all nodes $v \in V$, one at a time: if $v \in U$, it computes the value of the reduced cost of the subgraph induced by $U \setminus \{v\}$; if $v \in V \setminus U$, it computes the value of the reduced cost of the subgraph induced by $U \cup \{v\}$. In both cases, the move is forbidden if the resulting subset of nodes does not induce a connected subgraph or if its cost exceeds the threshold Q. Therefore, the neighborhood of each solution, therefore, is the set of all the solutions which can be obtained by applying one of the two kinds of move, and it contains at most n members.

A nonstandard tabu mechanism

As for any Tabu Search method, *PPTS* classifies the solutions in the current neighborhood either as tabu or non tabu. The presence of a tabu mechanism has the purpose to avoid the visit of previously obtained solutions by forbidding the reversal too recently performed moves. The common way to implement this mechanism is to maintain, explicitly or implicitly, a list of attributes of performed moves and to forbid the execution of moves whose attributes are in the list. The list has a limited length, say tt, usually called *tabu tenure*, and it is managed as a FIFO list. This implies that, after lasting tt iterations in the list, an attribute is removed from it and all the moves which have that attribute can now be performed.

In our algorithm, for each node $v \in V$, we save in I_v the last iteration in which v changed its status, either entering or leaving the solution. In technical terms, the *attribute* of a move is the index v of the node which is removed or added. As a consequence, the solution obtained adding (removing) a node v to (from) the current set of nodes U is tabu if the value of the current iteration counter is smaller than $I_v + tt$, meaning that v was moved for the last time less than tt iterations ago.

In the Tabu Search literature there are two mainstreams, respectively adopting a fixed and a variable tabu tenure. In the latter case, the tenure is usually updated depending either on the quality of the last performed move or on the cardinality of the neighborhood. More specifically, it is common to decrease the value of the tenure when the last performed move is improving and to increase it in the opposite case; as well, it is common to decrease the value of the tenure when the neighborhood becomes smaller and to increase it when it becomes larger (Glover and Laguna, 1997). The purpose of these adaptive mechanisms is to favor the exploration of more promising regions of the solution space and to drive the search away from less promising ones.

Since we visit only feasible solutions, which correspond to connected subgraphs of G, and since in general graph G is not complete, the size of the neighborhood defined above can vary significantly from iteration to iteration. For this reason, using a fixed tabu tenure proved very ineffective, and even the standard adaptive mechanisms, based on the quality of the last move or on the current size of the neighborhood, failed. In fact, we frequently observed that the value of the tabu tenure could not keep pace with the current state of the search. For example, the moves whose attributes were saved in the tabu list were quite often nearly all unfeasible, and therefore unnecessarily tabu. On the other hand, the insurgence of a cyclic behavior triggered the anti-cycling mechanism (described in the following) that increase the tabu tenure, until nearly all feasible moves became tabu. This worsened the quality of the available solutions. The result was that the search moved alternatively between cycles and bad solutions.

In order to solve this problem, we decided to get rid of the tabu tenure, while preserving the basic idea of Tabu Search. At each iteration, we compute the number k of feasible moves and we consider tabu the $\lfloor \varepsilon k \rfloor$ moves with the most recent attribute I_v. Parameter $\varepsilon \in (0;1)$ is defined by the user. Please notice that, due to the above rounding and since $\varepsilon < 1$, at least one move is always non tabu.

In general, the move selected at each step is the non tabu one which produces the solution with the minimum reduced cost in the neighborhood, but we also apply the standard *aspiration criterion*: if a tabu move brings us to a solution whose reduce cost is the smallest one found so far, we override its tabu status.

We also apply the following *anti-cycling mechanism*: if for a given number of consecutive iterations K_{acm} the same sequence of moves generates the same sequence of objective function values, we assume this as a hint that a cyclic behavior is occurring, and consequently increase ε to $\varepsilon' \in (\varepsilon;1)$ for the following K_{acm} iterations, in an attempt to break the cycle.

Finally, we adopt a *frequency-based diversification strategy*. We save the number n_v of visited solutions which contain node $v \in V$ and the number n_S of visited solutions which contain subset $S \in \mathscr{S}$. If the objective function does not improve for K_{ni} consecutive iterations, we start a diversification phase, which lasts for K_{div} consecutive iterations. During this phase, we replace the objective function with the following one:

$$\tilde{\phi} = \sum_{S \in \mathscr{S}} \frac{n_S}{\max\limits_{S \in \mathscr{S}} n_S} q_S z_S - \sum_{v \in V} \left(1 - \frac{n_v}{\max\limits_{v \in V} n_v}\right) \pi_v x_v$$

The aim of this change is to decrease the cost of the subsets and to increase the prize of the nodes which have occurred less frequently in the visited solutions.

Initialization

Heuristic *PPTS* requires a starting solution. We could start *PPTS* from an empty solution, however to deeply exploit the information obtained by the RMP instances we already solved, we adopt a *warm start* strategy: we restart *PPTS* from the subgraphs associated with all the k' basic variables λ_j which have a strictly positive value in the current optimal RMP solution. These solutions are promising starting points because, by definition, they have a zero reduced cost. Our preliminary experiments show that, using this *warm start* strategy, the overall Column Generation method requires less computing time w.r.t. the strategy that initializes *PPTS* with an empty solution.

Solution pool

To improve the Column Generation convergence rate, instead of the best solution, we save all the negative reduced cost columns found by *PPTS*. When the heuristic terminates, we add all the saved columns to the RMP.

Stopping criteria

PPTS has three stopping criteria. First of all, it stops as soon as it has found $C_{\max}$ columns. In fact, adding several columns in each iteration decreases the number of iterations required to obtain the optimal solution, but also increases the time required to solve the RMP. So, we need to find a trade-off between these two effects. Second, for each one of the k' starting solutions, *PPTS* performs at least $I_{\min}/k'$ iterations. If during this search it finds at least one negative reduced cost column, it moves to the next starting solution. Otherwise, it proceeds until either it finds a negative reduced cost column or it performs $I_{\max}/k'$ iterations, and moves to the next starting solution.

5.7.7 *Lower bound comparison*

Anti-cycling			Diversification		Stopping criteria		
K_{acm}	ε	ε'	K_{ni}	K_{div}	$C_{\max}$	$I_{\min}$	$I_{\max}$
3	0.7	0.9	25	30	1000	2000	20000

Table 5.13 Values of the parameters for *PPTS*.

This section compares the lower bound obtained by solving with CPLEX 12.4, on a PC equipped with and Intel Core2 Quad-core 2.66Ghz and 4GB of RAM, the compact formulation (5.6) with the lower bound achieved by solving the extended formulation MCE_{LP} (5.24) with our Column Generation method. formulation MCN (5.6) has been strengthened using the valid inequalities and the coefficient reductions described in Section 5.6.3. In the Column Generation method, to exactly solve the Pricing Problem we adopt the decomposition approach described in Section 5.7.4 that in our preliminary tests (not reported here for the sake of brevity) outperforms the one-step approach based on formulation PP (5.25).

We adopt the following strategy to speed up the Column Generation process. At first, we apply *PPTS* with the parameter setting reported in Table 5.13. As long as *PPTS* finds negative reduced cost columns, we add them to the RMP, and reoptimize it. After that, for each $\ell \in V$ in turn, we invoke CPLEX to solve problem PP^{ℓ}. As soon as CPLEX finds a negative reduced cost column (no matter if it is an optimal solution or not), we add it to the RMP and reoptimize. When CPLEX proves that PP^{ℓ} admits no such column, we select the next root ℓ and proceed with the associated Pricing Problem. When CPLEX fails to identify a negative reduced cost column for all $\ell \in V$, the optimal solution of the RMP is optimal also for the MP and provides a lower bound for the extended formulation. The whole process benefits by considering the roots ℓ in an increasing order, because in this way we solve first the larger subproblems PP^{ℓ}, which are more likely to provide negative reduced cost columns. Notice that in our first experiments we directly applied CPLEX to solve the Pricing Problems, because we had not yet implemented the *PPTS* heuristic. With that configuration, we could not compute the LP-Relaxation of the extended formulation in a reasonable amount of time even for small instances. Only after developing *PPTS*, it became possible to solve the MP. Moreover, our analysis of the computational experiments shows that the time required to solve the smallest Pricing Problems is negligible w.r.t. the time required to solve the biggest ones. As a consequence, we did not enumerate the smallest solutions, and we did not develop any *ad hoc* exhaustive search procedure, but we decided to simply apply the commercial *ILP* solver.

Tables 5.14, 5.15 and 5.16 report the results obtained, respectively, on benchmarks `A`, `B` and `C`. The first columns of the three tables identify each tested instance. For both lower bounding methods, the one based on the compact formulation under label CF (5.6), and the one based on the extended formulation under label MCE (5.24), column $\%\Delta$ reports the percentage gap between the best known value and the corresponding lower bound (computed as $(BK - LB)/LB$), and column CPU reports the required time in seconds. A time limit of one hour has been imposed on the computation: if an instance could be solved to optimality by CPLEX, the column $\%\Delta$ reports a “-” label and CPU column reports a value lower than 3600; if it could not be solved within the time limit, CPU column reports the label “TL” and the column $\%\Delta$ reports the residual gap.

Notice that in these tables we report only the results obtained on the smallest instances in the considered benchmarks. The larger instances, in fact, could not be solved using formulation MCN (5.6) and the best lower bounds found by CPLEX on these instances at the end of its execution exhibited very large gaps w.r.t. the best known upper bounds.

The results for formulation MCN (5.6) confirm its scalability issues already noted in Section 5.6.5: for the two random benchmarks `B` and `C`, considered in tables 5.15 and 5.16, some instances with 30 vertices and a single instance with 35 vertices can be solved to optimality within the time limit. If we consider benchmark `A` CPLEX is able to solve, in one hour of computation time, all the instances up to $n = 60$ and some instances with 70 and 80 nodes. The final gaps for the unsolved instances steeply increase when we pass from instances having 30 vertices to instances having

	CF (5.6)		MCE (5.24)	
Instance	%Δ	CPU	%Δ	CPU
50-1	-	134	14.73	76
50-2	-	97	0.35	73
50-3	-	480	3.73	53
50-4	-	117	14.37	122
50-5	-	81	4.70	68
60-1	-	1595	9.76	71
60-2	-	332	9.73	515
60-3	-	1543	10.54	95
60-4	-	282	-	395
60-5	-	517	4.68	220
70-1	2.55	TL	2.94	66
70-2	-	411	4.50	445
70-3	-	3315	9.06	368
70-4	3.80	TL	12.95	284
70-5	7.54	TL	6.43	352
80-1	20.30	TL	16.62	114
80-2	-	1228	3.67	524
80-3	15.90	TL	8.25	334
80-4	10.46	TL	4.92	141
80-5	16.39	TL	8.97	255

Table 5.14 Comparison between the lower bound achieved by compact formulation MCN (5.6) and by the Column Generation method applied to formulation $\mathrm{MCE_{LP}}$ (5.24) on benchmark A.

				MCN (5.6)		MCE (5.24)	
n	α	$q_{\max}$	β	%Δ	CPU	%Δ	CPU
30	0.05	10	1.00	-	181	3.02	3
30	0.05	10	1.15	-	312	3.64	3
30	0.05	100	1.00	-	328	3.01	4
30	0.05	100	1.15	-	186	2.78	2
30	0.10	10	1.00	-	3139	2.02	7
30	0.10	10	1.15	13.24	TL	14.85	7
30	0.10	100	1.00	13.05	TL	1.91	5
30	0.10	100	1.15	13.96	TL	4.50	7
35	0.05	10	1.00	9.72	TL	5.66	4
35	0.05	10	1.15	26.89	TL	4.92	8
35	0.05	100	1.00	-	2621	3.59	4
35	0.05	100	1.15	18.72	TL	5.97	4
35	0.10	10	1.00	64.58	TL	6.79	13
35	0.10	10	1.15	63.25	TL	10.54	16
35	0.10	100	1.00	32.73	TL	3.18	10
35	0.10	100	1.15	56.02	TL	5.77	14

Table 5.15 Comparison between the lower bound achieved by compact formulation MCN (5.6) and the Column Generation method applied to formulation $\mathrm{MCE_{LP}}$ (5.24) on benchmark B.

				MCN (5.6)		MCE (5.24)	
n	α	$w_{\max}$	β	$\%\Delta$	CPU	$\%\Delta$	CPU
30	0.05	1	1.00	-	571	6.68	3
30	0.05	1	1.15	-	2208	12.79	10
30	0.05	10	1.00	-	868	10.78	5
30	0.05	10	1.15	-	1525	17.28	6
30	0.05	100	1.00	-	747	11.13	3
30	0.05	100	1.15	-	1808	17.08	6
30	0.10	1	1.00	10.05	TL	8.01	12
30	0.10	1	1.15	-	1117	21.44	15
30	0.10	10	1.00	22.36	TL	10.23	10
30	0.10	10	1.15	-	1246	24.42	24
30	0.10	100	1.00	23.21	TL	12.36	12
30	0.10	100	1.15	-	1466	22.22	13

Table 5.16 Comparison between the lower bound achieved by compact formulation MCN (5.6) and the Column Generation method applied to formulation $\mathrm{MCE_{LP}}$ (5.24) on benchmark C.

$35-40$ vertices in benchmarks B and C and when we pass from instances having 60 vertices to instances having $70-80$ vertices in benchmark A.

The computation of the lower bounds obtained by CPLEX using formulation MCN for the unsolved instances requires to analyze thousands of branching nodes, with a non negligible memory. On the contrary, The Column Generation method does not require any branching operation and its results are obtained in a matter of few seconds (few minutes for the instances of benchmark A). The gap is quite stable with respect to the size of the considered instance and it mainly depends on the value of other instance parameters: the hardest instances for benchmark B are those having $\alpha = 0.10$, $q_{\max} = 10$ and $\beta = 1.15$; the hardest ones for benchmark C have $\alpha = 0.10$ and $\beta = 1.15$.

5.8 Local search based heuristics for the HAP

This section presents two local search based heuristics to determine feasible solutions of good quality for the HAP. The former is a Tabu Search algorithm based on the exchange of single vertices between adjacent subgraphs. Since the connectivity constraint is enforced, this neighborhood typically includes a very small number of solutions. To cope with this limitation, the algorithm relaxes cost threshold constraint, with an adaptive mechanism to penalize its violations. The latter algorithm strictly respects both the connectivity and the cost threshold constraint, but enlarges the search space through a Very Large Scale Neighborhood approach: instead of moving a single vertex from subgraph to subgraph, it allows whole chains of transfers, determining their cost through a shortest path computation on a suitable auxiliary graph.

Notice that both algorithms are based on the same elementary mechanisms, i. e. the addition or removal of a single vertex with respect to a subgraph, followed by the evaluation of the resulting cost and connectivity. Moreover, the same mechanisms can be used to build a starting feasible solution from scratch, through a sequence of incomplete solutions in which the subset of unassigned vertices is progressively reduced. The only difference between improving a complete solution and augmenting an incomplete one is that in the latter case the vertex added to a subgraph comes from the subset of unassigned vertices, instead of another subgraph. For this reason, both algorithms can be straightforwardly extended to deal with incomplete solutions, by simply representing the subset of unassigned towns as a fictitious subgraph.

In the description of both algorithms, we assign to each subgraph an integer index $l \in \mathbb{N}_k^0 = \{0,\dots,k\}$, where 0 denotes the fictitious subgraph, i. e. the subset of unassigned vertices, while $\mathbb{N}_k = \{1,\dots,k\}$ denotes the real subgraphs. This coincides with the indexing strategy used in formulation (5.2) and is in contrast with formulation (5.6), where each subgraph is denoted by the index of its first vertex $\ell \in V$.

5.8.1 A Tabu Search algorithm

For a general introduction to the Tabu Search see the introductory paragraph in Section 5.7.6, in the following, we mainly focus our attention on the specific aspects of our implementation for the HAP.

The algorithm starts by assigning all vertices to the fictitious subgraph $l = 0$. At each iteration, the procedure takes into account all vertices $v \in V$ and tries to move them from the current subgraph $l \in \mathbb{N}_k^0$ to another subgraph $l' \in \mathbb{N}_k \setminus \{l\}$. Notice that no move assigns a vertex to the fictitious subgraph 0, since it is the main objective of the algorithm to empty this subgraph.

In order to always keep connected subgraphs, moving v from l to l' is evaluated only if both the following conditions hold:

1. Either $l = 0$ or subgraph $l \in \mathbb{N}_k$ is not disconnected by the move.
2. Either v is adjacent to subgraph l' or subgraph l' is empty.

The tabu mechanism has the purpose to avoid reversing recently performed moves. If vertex v moves from subgraph l to subgraph l', it is forbidden to move it back to the original subgraph for a suitable number of iterations. This is obtained by saving in I_v^l the last iteration in which vertex v resided in subgraph l and declaring tabu any move which puts vertex v into subgraph l before iteration $I_v^l + L + 1$. In other words, the attribute of the move of v from l to l' is the pair (v,l). The number of forbidden iterations L is known as *tabu tenure*.

The cost threshold constraint can be freely violated by the moves, but these violations are suitably penalized in the objective function. This is done to guarantee a certain degree of flexibility to the approach, which might be otherwise too tightly limited by the small number of feasible neighbor solutions. For each move, the algorithm evaluates the following three objective function components:

- The new cost of subgraph 0.
- The resulting value of the objective function ϕ (5.1).
- The resulting total violation of the cost threshold constraint, which is defined as

$$\xi = \sum_{l=1}^{k} \max\left(c_{U_l} - Q, 0\right)$$

Then, the algorithm determines the moves which produce the minimum cost for subgraph 0. If different moves produce the same cost, the algorithm computes for these moves a linear combination of the variations of ϕ and ξ with respect to the current solution:

$$(\phi_{\text{new}} - \phi_{\text{curr}}) + \psi\,(\xi_{\text{new}} - \xi_{\text{curr}})$$

where $\psi > 0$ is a suitable coefficient. The best non tabu move with respect to this combination identifies the incumbent solution, which will replace the current one. There is, however, also an *aspiration criterion* which states that the best tabu move is preferred if it yields a solution which is both better and not more unfeasible than the best known one.

The penalization factor ψ is adaptively tuned with a strategy inspired by the so called *strategic oscillation*, which is a general technique suggesting to visit alternatively feasible and unfeasible solutions, when looking for optimal solutions to problems with tight or complex constraints Glover and Hao (2011). Specifically, given a coefficient $\rho > 1$, the value of ψ increases at each iteration in which the current solution is unfeasible ($\psi \leftarrow \psi \cdot \rho$), whereas it decreases when the current solution is feasible ($\psi \leftarrow \psi/\rho$). This guarantees that the search will abandon sooner or later the unfeasible region; on the other side, if the search dwells for long in the feasible region, the penalization decreases making it easier to evaluate unfeasible solutions, if profitable. Coefficient ρ determines how fast this process should be and how ample the resulting oscillations are.

The *tabu tenure* L also varies adaptively according to the recent results of the search. Specifically, L increases after a worsening move, respecting a maximum predefined value, $L \leftarrow \min(L+1, L_{\max})$, and it decreases after an improving move, respecting a minimum predefined value, $L \leftarrow \max(L-1, L_{\min})$.

5.8.2 A Very Large Scale Neighborhood algorithm

The Very Large Scale Neighborhood algorithm, denoted in the following as *VLSN*, is a classical local search algorithm, in which the current solution is iteratively replaced by the best one drawn from a suitable neighborhood, as long as this replacement improves the objective function. The neighborhood adopted has a cardinality that is exponential in the size of the considered HAP instance, but the selection of its best element is performed without explicitly evaluating them one by one, thanks to a customized optimization technique (Ahuja et al., 2002). Our application of this general framework to the HAP is inspired by the *cyclic exchanges* proposed in (Thompson and Psaraftis, 1993) for the Vehicle Routing Problem and the *ejection chains* proposed in (Glover, 1996). The idea is to generate compound sequences of elementary moves, which move towns from their current subgraph to another one, producing the "ejection" of other towns.

Definition of the neighborhood

We define the neighborhood of a given solution as the set of all ordered sequences $(v_1,\dots,v_r,U_l)$, where r is a positive integer, v_i is a vertex for $i=1,\dots,r$ and U_l is one of the subgraph in the current solution. Notice that, as in the Tabu Search algorithm (see Section 5.8.1) and contrary to formulation (5.6), the subgraph is identified by an integer number $l \in \mathbb{N}_k$. Each sequence $(v_1,\dots,v_r,U_l)$ identifies a neighbor solution, which is obtained from the current one by moving vertex v_i to the subgraph of vertex v_{i+1} for $i=1,\dots,r-1$ and by moving vertex v_r to subgraph U_l. For example, the left side of Figure 5.5 represents an instance of the HAP with eleven vertices and a solution with the following four subgraphs: $U_1=\{1,4,6\}$, $U_2=\{2,3,5\}$, $U_3=\{7,9,11\}$, $U_4=\{8,10\}$. The sequence $(4,3,11,U_4)$ described by the arrows corresponds to moving vertex $v_1=4$ from the first subgraph to replace vertex $v_2=3$ in the second subgraph; this replaces vertex $v_3=11$ in the third, and finally vertex $v_3=11$ moves into subgraph $l=4$. The right side of Figure 5.5 represents the resulting solution, in which $U_1=\{1,6\}$, $U_2=\{2,4,5\}$, $U_3=\{7,9,3\}$ and $U_4=\{8,10,11\}$. In general, the subgraph of the starting vertex in the sequence loses one vertex, the final subgraph gains one vertex, while the subgraphs of the intermediate towns lose and gain exactly one vertex.

Auxiliary graph representation

The cardinality of this neighborhood is exponential in the size of the considered HAP instance, and it is impractical to evaluate each neighbor solution explicitly. However, a move can be represented by a path on a suitable auxiliary graph, G_{VLSN}, and applying some further limitations it is possible to compute the best move with a shortest path computation. Figure 5.6 provides the auxiliary graph for the HAP instance of Figure 5.5. The node set of graph G_{VLSN} includes a source node s, a sink node t, a subset N_V containing a *vertex node* n_v for each vertex $v \in V$, and a subset N_L containing a *subgraph node* n'_l for each subgraph $l \in \mathbb{N}_k$. The arc set consists of four subsets: the arcs in $A_{sV}=\{s\}\times N_V$ connect the source node s to all the nodes associated with vertices in V, the arcs in $A_{VV}=N_V\times N_V$ connect all pairs of nodes associated with vertices in V, the arcs in $A_{VL}=N_V\times N_L$ connect nodes associated with vertices in V to subgraph nodes, the arcs in $A_{Lt}=N_L\times\{t\}$ connect all subgraph nodes to the sink t. An element of the neighborhood corresponds to

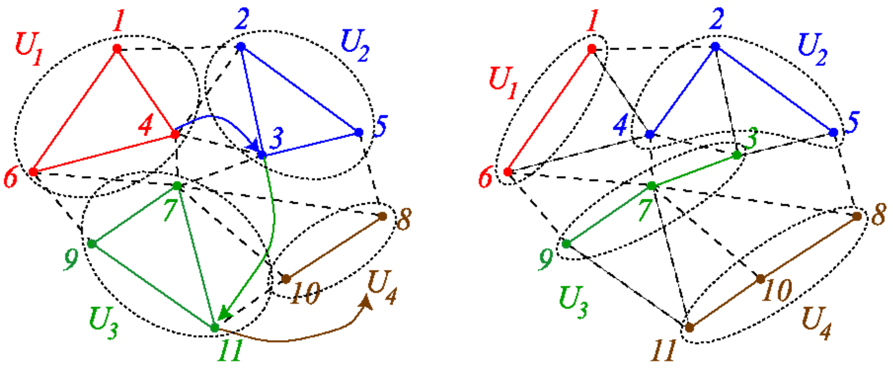


Fig. 5.5 An instance of the HAP with a starting solution (on the left), a *VLSN* move $(4,3,11,U_4)$ represented by arrows and the resulting solution (on the right): vertex 4 moves to replace vertex 3, which replaces vertex 11, which in the end moves to subgraph U_4; all the vertices involved belong to subgraphs different from each other and from the final one.

an ordered sequence $(v_1,\ldots,v_r,U_l)$, i. e. to a path from s to t on graph G_{VLSN}. Figure 5.6 shows the path corresponding to the move which turns the solution on the left side of Figure 5.5 into the solution on the right side.

Each arc of this path represents an elementary component of the overall move: each arc $(s,n_u) \in A_{sV}$ models the removal of vertex u from the subgraph it currently belongs to (in our example, arc (s,n_4)); each arc $(n_u,n_v) \in A_{VV}$ models the removal of subgraph v from its current subgraph and the insertion of vertex u in that subgraph (i. e. arcs (n_4,n_3) and (n_3,n_{11})); each arc $(n_v,n'_l) \in A_{VL}$ models the insertion of subgraph v into subgraph U_l (i. e. arc (n_{11},n'_4)). Finally, the arcs in A_{Lt} have the purpose to impose a common destination to all relevant paths.

We assign to each arc $a \in A_{sV} \cup A_{VV} \cup A_{VL} \cup A_{Lt}$ a cost c_a, equal to the variation of objective function ϕ produced by the elementary removal, replacement or insertion modeled by the arc; the arcs in A_{Lt} have zero cost, because they do not represent any modification. Provided that each vertex v_i belongs to a different subgraph and that none of them belongs to subgraph l, the overall variation of the objective produced by the move $(v_1,\ldots,v_r,U_l)$ is equal to the cost of the corresponding $s-t$ path in G_{VLSN}, i.e. to the sum of the costs of the path arcs. The additional restriction of visiting at most one (vertex or subgraph) node for each subgraph is necessary; otherwise, the costs should take into account the interactions of multiple modifications on the same subgraph. The additional constraint also allows to neglect negative cost cycles, since a node cannot be visited twice. On the other side, the resulting constrained shortest path problem is $\mathscr{N}\mathscr{P}$-hard, and it is solved as described in the following.

Moreover, not all moves produce feasible solutions: some yield disconnected subgraphs or cost exceeding the threshold. In order to guarantee feasibility, the cost of an arc a is set to $c_a = +\infty$ when: (a) the elementary removal, replacement or insertion associated to the arc produces a disconnected subgraph, (b) the elementary

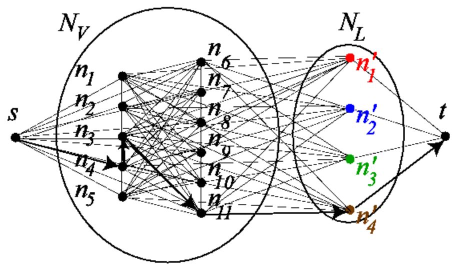


Fig. 5.6 The auxiliary graph G_{VLSN} for the HAP instance of Figure 5.5; the path $(s, n_4, n_3, n_{11}, n'_4)$ corresponds to the *VLSN* move $(4, 3, 11, U_4)$, which turns the solution on the left side of Figure 5.5 into the solution on the right side: in detail, vertex 4 moves from the first subgraph to the second one, from which vertex 3 moves to replace in the third subgraph vertex 11, which in the end enters subgraph U_4.

modification produces a subgraphs whose cost is greater than Q. By $+\infty$ we denote a value sufficiently large to forbid the use of the arc in any optimal solution.

Extension to incomplete solutions

The definition of neighborhood can be easily extended to incomplete solutions, in which some vertices have not yet been assigned to any subgraph. In order to do that, it suffices to augment $\mathbb{N}_k$ to $\mathbb{N}_k^0 = \{0, \dots, k\}$, introducing a fictitious subgraph $l = 0$, which includes all unassigned vertices. In order to favor the generation of feasible solutions by transferring vertices from subgraph 0 to the other ones, we multiply the nonpositive costs of the arcs $(s, n_u) \in A_{sV}$ for all unassigned vertices u by a positive constant factor C, large enough to make them more profitable than any path avoiding those arcs. In order to forbid all transfers of vertices from actual subgraphs to the fictitious one, the cost of the arcs $(n_u, n_v) \in A_{VV}$ is set to $+\infty$ when v belongs to the fictitious subgraph 0 and subset N_L includes no node for subgraph 0. This extension allows to apply Algorithm *VLSN* both as a constructive and as an improvement procedure, given that the algorithmic steps required are the same in the two cases.

Efficient identification of the optimal move

A pseudocode of Algorithm *VLSN* is provided in Figure 5.7. First of all, procedure *BuildGraph* constructs the topological structure of the auxiliary graph G_{VLSN}. This is fixed once for all, whereas the arc costs change from iteration to iteration, depending on the current solution. Then, the algorithm assigns all vertices to the fictitious subgraph $l = 0$.

At each local search iteration, procedure *ComputeArcCost* computes the arc costs referring to the current solution and procedure *ComputeConstrainedShortestPath* solves the constrained shortest path problem mentioned above, in order to find the path sp which represents the optimal move, whose cost is denoted as $c(sp)$. If the move is improving ($c(sp) < 0$), procedure *ApplyMove* performs it and another local search iteration starts; otherwise, the algorithm terminates. Notice that, since no move introduces vertices into the fictitious subgraph, once the algorithm obtains a complete solution, it only visits other complete solutions, as a standard local search algorithm. Moreover, due to the definition of the cost function, all moves which increase the number of assigned vertices dominate the other ones. Therefore, the algorithm typically starts with a constructive phase (of at most n iterations), followed by an improvement phase. However, the two phases can overlap, when a vertex, which first fails to find a feasible subgraph, finds it after some local search iterations.

For the sake of simplicity and computational efficiency, the auxiliary constrained shortest path problem is solved heuristically with a vertex-weighted version of Dijkstra's algorithm. With respect to the standard implementation, this algorithm removes from the graph all the nodes belonging to the same subgraph of the last node marked permanently at each step. The path thus obtained necessarily satisfies the requirement to visit at most one node for each subgraph, though it might not be the optimal path. The choice of a heuristic procedure in this situation is common in cyclic exchange or ejection chain heuristics (Thompson and Psaraftis, 1993; Glover, 1996), and its aim is to limit the computational time of the single iteration, in order to perform a larger number of iterations in the available time.

```
**Algorithm** VLSN($G, \mathscr{S}, q, Q, k$);
$G_{\text{VLSN}}$ := *BuildGraph*($G$, $k$);
$U_0 := V$; **foreach** $l = 1, \dots, k$ **do** $U_l := \emptyset$;
**repeat**
    **foreach** $(i, j) \in A$ **do** $c_{ij}$ := *ComputeArcCost*($G, U_0, U_1, \dots, U_k, \mathscr{S}, q, Q$);
    $sp$ := *ComputeConstrainedShortestPath*($G_{\text{VLSN}}, U_0, U_1, \dots, U_k, c$);
    **if** $c(sp) < 0$ **then** *ApplyMove*($sp, U_0, U_1, \dots, U_k$);
**until** $c(sp) \geq 0$;
**return** $(U_0, U_1, \dots, U_k)$;
```

Fig. 5.7 Pseudocode of the *VLSN* algorithm.

5.8.3 *Computational experiments on the local search based heuristics*

In this section we report the results we obtained when we compared the two different local search based heuristics described in the two previous sections.

Tuning of the Tabu Search parameters

A first phase of experiments was dedicated to tuning the parameters of the Tabu Search algorithm for the HAP. In the following, we denote this algorithm by *HAP-TS*. The behavior of *HAP-TS* is controlled by three parameters, namely the range of the tabu tenure $[L_{\min};L_{\max}]$ and the coefficient ρ used to adapt the penalization factor ψ associated with the cost threshold violation. To achieve results that do not depend too much on the first initial solution visited by *HAP-TS*, we developed a multi start version of *HAP-TS*, denoted in the following by *MS-TS*. A single execution of *MS-TS* consists of R executions of *HAP-TS*, where each one of these executions is initialized by selecting with a random uniform distribution a single vertex to be inserted in each real subgraph, and the not selected vertices define the fictitious subgraph $l = 0$. In these first experiments, we set $R = 10$ and we limit the total number of iterations in each restart to $T = 10000$ in order to achieve sufficiently stable results while keeping limited the amount of computational resources required by it. Some previous computational experiments, described in details in (Ceselli et al., 2013), indicate that a variability in the range of tabu tenure equal to $(L_{\max} - L_{\min}) = 5$ is sufficient to effectively adapt the behavior of *MS-TS* according to the quality of the visited solutions. Thus, we concentrate our attention on five different ranges of tabu tenure:

$$[L_{\min};L_{\max}] \in \{[10;15], [15;20], [20;25], [25;30]\}$$

Similarly, we focus our attention on the following values of parameter ρ that, in preliminary tests, seemed to be the most promising ones:

$$\rho \in \{1.1, 1.15, 1.2, 1.25, 1.3, 1.35, 1.4\}$$

In Figure 5.8 we report four different plots, one for each considered tabu tenure range, that compare the average performances of *MS-TS* varying the value of parameter ρ. On the y-axis, we report the average value of the percentage gaps between the results obtained with each parameter setting and the overall best known results. On the x-axis, we report the seven ρ values considered during this campaign. Analyzing these results we can easily conclude that the two most robust ranges for the tabu tenure are $[15;20]$ and $[20;25]$. The minimum average gap (equal to 0.17%) is obtained using either range $[15;20]$ and $\rho = 1.35$ or range $[20;25]$ and $\rho = 1.3$. The performance of *MS-TS* rapidly deteriorates if we either increase or decrease the considered tabu tenure values. For the two best tabu tenure ranges choosing ρ in the interval $[1.25, 1.35]$ produces always good results. If we consider the range $[15;20]$, we can obtain good results also by setting $\rho = 1.15$. Moreover, using range $[15;20]$ combined with 4 of the 7 tested values of parameter ρ we achieve the best average results for the fixed ρ value. Thus, in the following experiments we set $[L_{\min}, L_{\max}] = [15;20]$ and $\rho = 1.35$.

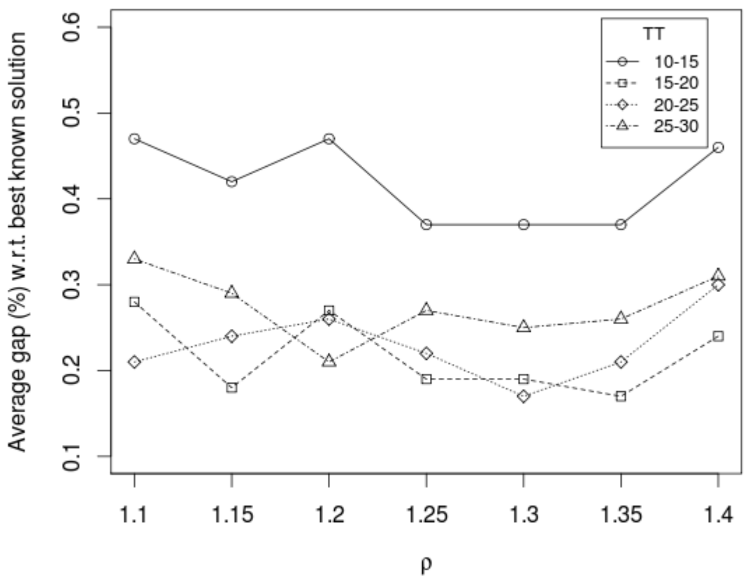


Fig. 5.8 Comparison of the results obtained by the *MS-TS* varying the tabu tenure range and the ρ value.

Very Large Neighborhood Search

We then consider the results that can be obtained using the Very Large Neighborhood Search heuristic described in Section 5.8.2. Since a single run of this algorithm takes a very short computational time, but yields solutions of very heterogeneous quality, we have implemented a multi-start version of it, denoted as *MS-VLNS*, which generates the starting solution assigning a random town to each one of the actual areas, as in *MS-TS*. We compared *MS-VLNS* with *MS-TS* with the best parameter setting among the ones tested in the previous section: $\rho = 1.35$, $[L_{\min}, L_{\max}] = [15, 20]$ and $T = 10000$. For the sake of fairness, we did not limit the number of *MS-VLNS* restarts but, for each tested instance, we executed *MS-VLNS* with a time limit equal to the time required by *MS-TS* to terminate its execution. The PC used to execute both heuristics is equipped with an Intel Xeon Processor E5-1620 Quad Core 3.60GHz and 16 GB of RAM. In Table 5.17 we report, for each benchmark, the average gaps between the solutions obtained both by *MS-TS* and by *MS-VLNS* and the best solutions found during the entire computational campaign. Analyzing these results we can observe that, on all the three considered benchmarks, *MS-TS* outperforms *MS-VLNS*.

	MS-VLNS%	*MS-TS%*
A	1.86	0.49
B	1.11	0.11
C	2.54	0.12

Table 5.17 Average percentage gaps between the best known solutions of the instances in the three different benchmarks and the solutions found by *VLNS* and *MS-TS*.

Combining Tabu Search and Very Large Scale Neighborhood Search

In order to improve the diversification phase of *MS-TS*, we consider a modified version of algorithm *MS-VLNS*, in which the solution computed at each restart is improved by Tabu Search. This modified version can be seen equivalently as a modified version of *MS-TS* which improves the starting random solution with *VLNS* before applying Tabu Search. We call this hybrid version *VLNS-TS*.

We compare the two restart strategies for the *HAP-TS* by using the best parameters among the one tested in Section 5.8.3, i.e. $\rho = 1.35$ and $[L_{\min}, L_{\max}] = [15, 20]$. However, here we set the maximum number T of *HAP-TS* iterations for each restart in a different way. Namely, we consider four different cases, $T \in \{2500, 5000, 10\,000, 25\,000\}$. We execute all the tests using the same time limit strategy and the same machine used in the previous section.

In figures 5.9,5.10 and 5.11 we report the average gaps between the results obtained by *VLNS-TS* and *MS-TS* and the best solution found during the whole computational campaign, varying T, respectively for benchmark A,B and C. In Figure 5.12 we report the same results but there the average gaps are computed on all the three considered benchmarks.

Analyzing these results, we can immediately note that the relative performance of *VLNS-TS* and *MS-TS* strongly depends on the considered benchmark: on benchmark A and benchmark B, we obtain the best results with *VLNS-TS* and on benchmark C we obtain the best results with *MS-TS*. We can note a similar dependence on the tested benchmark, if we consider the performance of the two heuristics varying the number of iterations T for each restart: *VLNS-TS* obtains the best results on benchmarks A and C when $T = 5\,000$, but on benchmark B the best results are obtained by setting $T = 25\,000$. This ranking changes if we restrict our attention on the results obtained by *MS-TS*: using this algorithm, the best results are obtained by setting $T = 2\,500$, $T = 25\,000$ and $T = 5\,000$, respectively for benchmark A, benchmark B and benchmark C. The results averaged on all the three benchmarks clearly show that *VLNS-TS* has a better restart strategy w.r.t. *MS-TS* since it outperforms its competitor when we set $T \in \{5\,000, 10\,000, 25\,000\}$. However, if we consider $T = 2\,500$ *MS-TS* obtains slightly better results w.r.t. *VLNS-TS* (0.044% versus 0.062%).

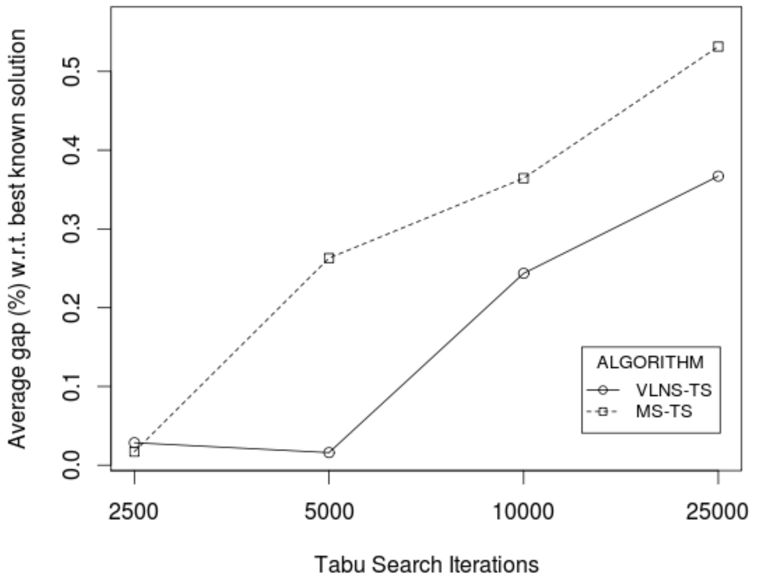


Fig. 5.9 Comparison between the average gaps between the solutions obtained by *VLNS-TS* and *MS-TS* and the best solutions found during the entire computational campaign on instances in benchmark A.

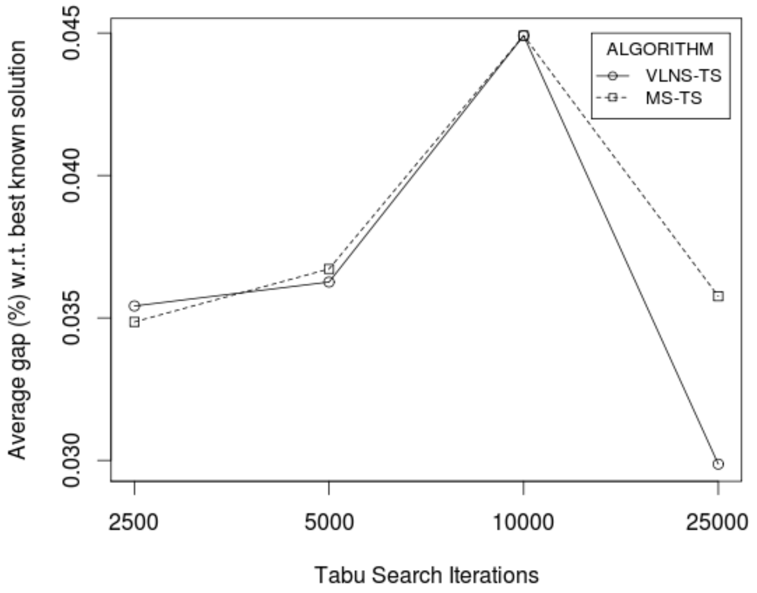


Fig. 5.10 Comparison between the average gaps between the solutions obtained by *VLNS-TS* and *MS-TS* and the best solutions found during the entire computational campaign on instances in benchmark B.

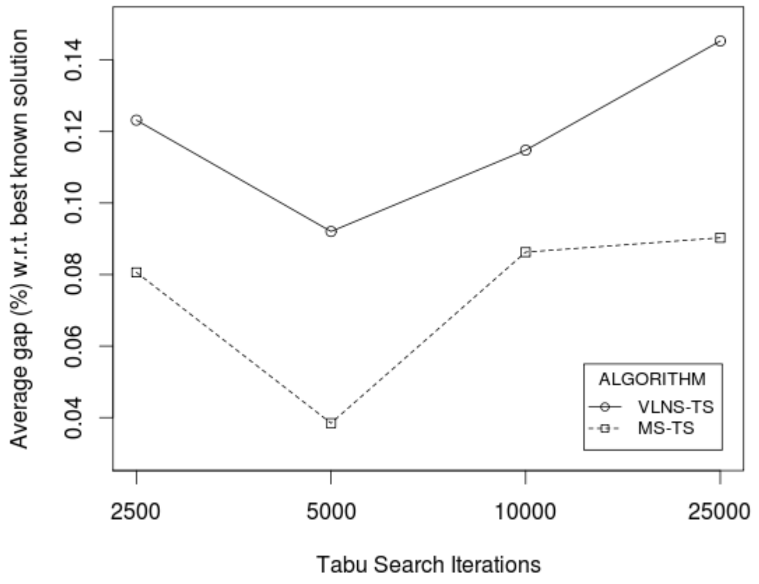


Fig. 5.11 Comparison between the average gaps between the solutions obtained by *VLNS-TS* and *MS-TS* and the best solutions found during the entire computational campaign on instances in benchmark `C`.

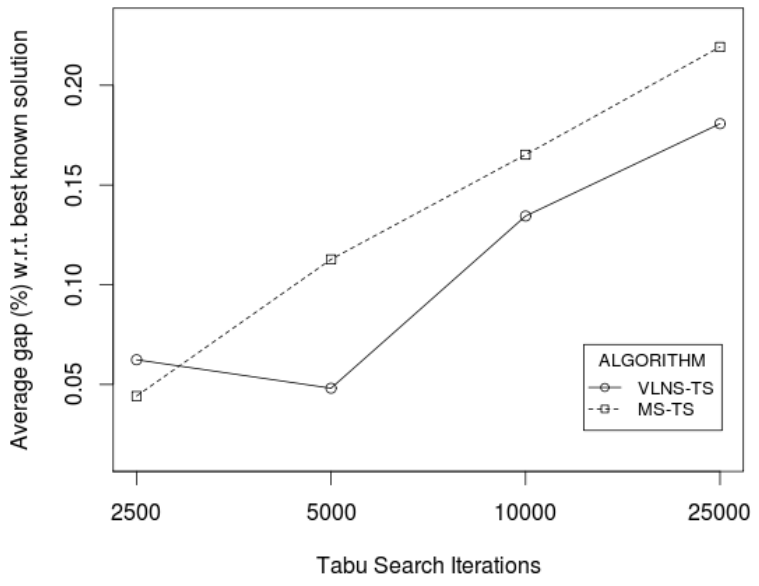


Fig. 5.12 Comparison between the average gaps between the solutions obtained by *VLNS-TS* and *MS-TS* and the best solutions found during the entire computational campaign, computed on all the instances belonging to the three considered benchmarks.

5.9 Column Generation based heuristics for the HAP

Besides a tight lower bound, during its execution, the Column Generation method provides also valuable information that we can exploit to construct near optimal solutions. In this section we investigate three different strategies to embed the Column Generation method described in full details in Section 5.7 in a hybrid heuristic in order to efficiently solve the HAP.

The first one of these strategies (described in detail in Section 5.9.1) has been first proposed in Joncour et al. (2010), the idea behind this strategy is to heuristically explore a branching tree based on formulation MCE_{LP} (5.24). At each branching node, the branching variable is chosen according to the information extracted from the optimal solution of the last solved RMP. To balance intensification and diversification, this strategy executes repeated diving phases guided by a simple backtracking mechanism.

The second strategy that we consider (described in detail in Section 5.9.2) is based on a fast exploration of a branching tree similar to the one explored by the previous strategy. However, differently from what done by the first strategy, in this case the order in which the branching nodes are processed is dynamically defined by a simple Tabu Search mechanism proposed in Cacchiani et al. (2012).

Finally, the last strategy we considered (described in detail in Section 5.9.3), is a Large Neighborhood Search (LNS) algorithm where we use the Column Generation method to define an effective repair operator. The hybridization between Column Generation and LNS has been first proposed in Prescott-Gagnon et al. (2009) to solve the Vehicle Routing Problem with Time Windows.

5.9.1 Limited Discrepancy Search

In the following, we describe the *CG-LDS* heuristic for the HAP. This algorithm heuristically explores a branching tree based on formulation MCE_{LP} (5.24). At each branching node, using the Column Generation method, it solves a LP formulation obtained by adding some branching constraints to MCE_{LP}. Differently from what generally done in a standard *Branch & price* algorithm (Vanderbeck, 2000), it branches directly on MCE_{LP} variables.

Since PP is $\mathcal{NP}$-hard (see Section 5.7.2), at each branching node, we truncate the Column Generation method as soon *PPTS* fails to find variables with negative reduced costs and we do not execute any exact solver for the PP. Hence, the final value of the last solved RMP is not guaranteed to be a lower bound of the original problem. However, despite its suboptimality, the heuristic solution of MCE_{LP} obtained in this way might provide useful information to identify good feasible integer solutions. The number of iterations of *PPTS* executed at each node can be controlled by the user by setting the *PPTS* parameters $I_{\max}$ and $I_{\min}$ as described in Section 5.7.6.

CG-LDS maintains a list L of tabu variables, which is initially empty. At each branching node, *CG-LDS* selects the variable that does not belong to L, associated with the largest value in the solution of the current RMP. Then, it fixes to 1 the selected variable, and updates the RMP accordingly: the right-hand side of constraints (5.24b) turns from 1 to 0 for the nodes belonging to the subgraphs associated with the fixed variable and the right-hand side of constraint (5.24c) decreases by one. After this fixing, *CG-LDS* iteratively reoptimizes the RMP with the Column Generation method, always applying only the heuristic pricing procedure. In practice, it is very easy to take into account the fixed variables when we heuristically solve the Pricing Problem: we just need to remove from graph G all the nodes associated with the fixed variables, and solve the pricing subproblem only on the remaining subgraph. The process of fixing variables and reoptimizing the RMP is called *diving*. It terminates when the solution of the current RMP is either integer or unfeasible.

At the end of each diving phase, *CG-LDS* starts a *backtracking* phase. This phase is controlled by two parameters: the maximum backtracking depth $D_{\max}$ and the maximum length $L_{\max}$ of the tabu list. In detail, the backtracking stops when the current depth becomes $< D_{\max}$ or when the length of the current tabu list becomes $< L_{\max}$. When it is no longer possible to backtrack, *CG-LDS* terminates. Otherwise, it creates a new child node, whose tabu list includes all the variables which were tabu in the parent node plus those that have been fixed in the previously generated sibling nodes.

An example of the branching nodes explored by *CG-LDS* is illustrated in Figure 5.13, for $D_{\max} = 2$, $L_{\max} = 2$. The labels of the branching nodes indicate the order in which they are visited. The label of each arc reports in parenthesis the tabu list L which constrains the choice of the next fixed variable and the variable which has been fixed into the previous branching node. Let the candidate branching variables at the root node be y_a, y_b and y_c, in nonincreasing order of value. After fixing variable y_a and reoptimizing the obtained RMP, let the candidate branching variables at node 1 be y_d, y_e and y_f. Since list L is empty, algorithm *CG-LDS* chooses y_d and reoptimizes the RMP. Then, it *dives*, i. e. it keeps fixing other variables until the current RMP has an integer solution or becomes unfeasible. It backtracks up to the first level whose depth is $< D_{\max} = 2$, i. e. up to node 1, inserts variable y_d into list L and creates a new child node 3, fixing variable y_e. From there, the algorithm dives again, and backtracks once more up to 1. Now, it inserts variable y_e into list L and creates a new child node 4, fixing variable y_f. From node 4, *CG-LDS* first dives and then backtracks up to node 0; node 1, in fact, cannot generate other children, because the length of list L has grown to $L_{\max} = 2$. Back at the root node, the algorithm puts variable y_a into L and fixes variable y_b. Then, it proceeds as reported in Figure 5.13. In particular, notice how parameter $L_{\max}$ limits the number of children at node 5 and directly imposes to dive at node 8.

The best feasible integer solution found during the different diving phases is saved. Since the HAP is a partitioning problem and imposes an upper limit k on the number of subgraphs, the algorithm does not guarantee to always obtain a feasible solution. At the end of the branching process, however, the columns generated in all

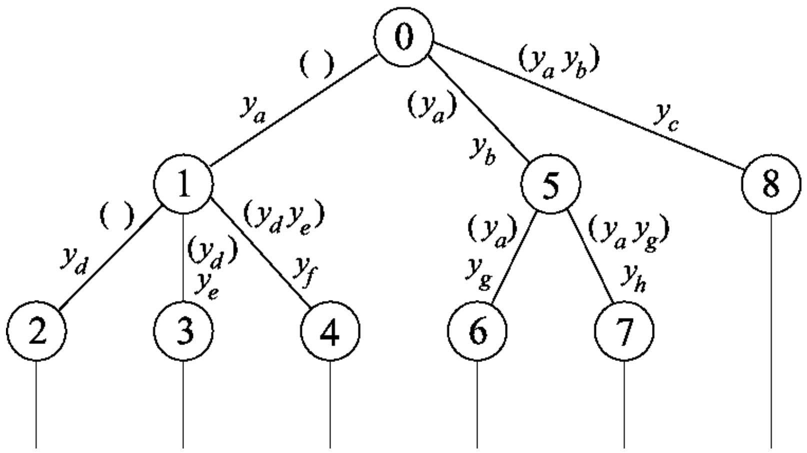


Fig. 5.13 An example of branching tree for heuristic *CG-LDS*.

the processed branching nodes form an ILP problem, which is a reduced instance of formulation (5.24). We solve it by means of a general-purpose solver, possibly obtaining a feasible solution, which cannot be worse than the best one found (if any) during the diving phases.

5.9.2 An alternative way to use a tabu memory

The authors of Cacchiani et al. (2012) proposed a different way to use a tabu memory to diversify the set of solutions visited by a Column Generation based heuristic. We followed their idea and we developed the *CG-TS* heuristic described in the following.

CG-TS partially explores the same branching tree explored by *CG-LDS* and described in the previous section. Each branching node is processed using a Column Generation method truncated as soon as *PPTS* is not able to find any negative reduced cost arborescence in the given number of iterations. Then, *CG-TS* introduces the integrality constraint in the last generated RMP and solve the obtained problem with an ILP solver. If the feasible solution found during this last step is better than the incumbent solution, we substitute the latter solution with the former. Note that, in order to limit the computational resources required by this last step, we set a 100 seconds timeout in the ILP solver.

Differently from *CG-LDS*, *CG-TS* does not visit branching nodes following a parent-child relationship. When *CG-TS* finishes to process the current node it generates the next node to be visited by applying one of the two following procedures to the just processed node :

- *Variable fixing*: a new branching node is generated by fixing to one a variable among the ones belonging to the optimal solution of the last solved RMP.
- *Variable releasing*: a new branching node is generated by releasing a fixed variable.

By fixing variables, *CG-TS* can quickly find integer solutions, however, since these fixings rapidly shrink the feasible region, the obtained solutions may be of poor quality and, in the worst cases, the shrunk region can become empty. Thus, the releasing phase is crucial for a successful exploration of the solution space. We release the fixed variable that has been fixed for the longest time. This choice allows *CG-TS* to diversify the search. A different strategy based on the release of a recently fixed variable could intensify the search around the last feasible solution found. However, as deeply discussed in Cacchiani et al. (2012), the interleaving between the generation of new columns and the fixing of new variables may already intensify the search around the fixed variables.

The information gained through the truncated execution of the Column Generation method on the current branching node is exploited by two different parts of *CG-TS*:

1. The optimal objective function value of the last solved RMP, denoted by $\phi(\tilde{X}_{LB})$, is used to choose which node to visit next: if we denote the incumbent solution by X^*, *CG-TS* fixes a new variable if $\phi(\tilde{X}_{LB}) < \phi(X^*)$, otherwise it releases one of the fixed variables.
2. To choose the variable to fix in each fixing step, *CG-TS* considers the values of the variables defining $\tilde{X}_{LB}$ and it fixes the variable associated with the highest value.

To avoid cycles consisting of fixing and releasing of the same set of variables, *CG-TS* drives the search with a tabu list having a fixed length TT. While a variable belongs to this list, it cannot be fixed and, after the fixing of a non-tabu variable we insert it in the tabu list. After a preliminary tuning phase, we set $TT = 25$.

In Figure 5.14 we provide a pseudocode for *CG-TS*. At the beginning, it generates the feasible solution $\tilde{X}$ and its continuous relaxation $\tilde{X}_{LB}$ by using the Column Generation method and by solving the final RMP with integrality constraint (both steps are summarized by the *CGProcess* function). At each iteration, it first updates the set of fixed variables and the tabu list, then it applies *CGProcess* on the modified $FIXED$ set, generating a new pair of solutions $(\tilde{X}_{LB}, \tilde{X})$.

5.9.3 Large Neighborhood Search

The last Column Generation based heuristic that we propose for the HAP, denoted in the following by *CG-LNS*, is based on the Large Neighborhood Search (LNS, first introduced in Shaw 1998, see Pisinger and Røpke 2010 for a complete survey). LNS denotes a particular set of algorithms belonging to the Very Large Scale Neighborhood Search (VLSN, see Section 5.8.2) class. Differently from the most local search

```
Algorithm CG-TS($G, \mathscr{S}, q, Q, k$);
$L := \emptyset$
$FIXED := \emptyset$
$(\tilde{X}_{LB}, \tilde{X})$ :=CGProcess($G, \mathscr{S}, q, Q, k, FIXED$);
$X^* = \tilde{X}$
repeat
    if $\phi(\tilde{X}_{LB}) < \phi(X^*)$ then
        $T$ := arborescence with the highest value in $\tilde{X}_{LB}$;
        $FIXED := FIXED \cup \{T\}$;
        if $| L |= TT$ then remove the oldest entry in $L$;
        $L := L \cup \{T\}$
    else
        remove the oldest entry in $FIXED$;
    if $\phi(\tilde{X}) < \phi(X^*)$ then $X^* := \tilde{X}$
    $(\tilde{X}_{LB}, \tilde{X})$ :=CGProcess($G, \mathscr{S}, q, Q, k, FIXED$);
until termination criterion is met
return $X^*$;
```

Fig. 5.14 Pseudocode of the *CG-TS* algorithm.

based heuristics, LNS does not define the solution neighborhood in an explicit way. In a LNS heuristic the neighborhood is implicitly defined by a destroy and a repair procedure. The former procedure partially destroys the current solution, the latter procedure repairs the partial solution generated by the destroy procedure. Thus, the neighborhood explored at each LNS iteration contains all the solutions that can be obtained by consequently applying these two procedures to the incumbent solution. The size of this neighborhood mainly depends on the size of the part of solution that is destroyed in each LNS iteration. A LNS extension called Adaptive Large Neighborhood Search (ALNS) has been proposed in Røpke and Pisinger (2006) in order to diversify the search by introducing several destroy and repair procedures and by choosing which methods to execute at each iteration using an adaptive random mechanism.

In our implementation of the ALNS metaheuristic for the HAP we consider a single repair procedure that is based on the Column Generation method. This procedure is similar to the one proposed in Prescott-Gagnon et al. (2009), it takes in input a partial HAP solution denoted by $\mathscr{A} = \{A_1, \ldots, A_k\}$. Each $A_i = \{v_1, \ldots, v_{N_i}\} \in \mathscr{A}$ is a vertex subset that induces a subgraph in G that must satisfy the cost threshold constraint but may be not connected and the set of vertices belonging to the subsets in $\mathscr{A}$ does not contain all the G vertices (i.e. $\bigcup_{i=1}^{k} A_i \subset V$). At each iteration of the Column Generation method we use a modified version of *PPTS* that takes into account the $\mathscr{A}$ structure: it can add (resp. drop) a single vertex to (resp. from) the current solution only if it belongs to $V \setminus \bigcup_{i=1}^{k} A_i$ and all the vertices contained in a subset $A_i \in \mathscr{A}$ can be added or dropped only at once. As soon as the modified version of *PPTS* is not able to find a negative reduced cost arborescence, we stop the Column Generation method and we add the integrality constraint to the last generated RMP. We solve the obtained ILP problem using an ILP solver in order to complete the repair procedure. In order to limit the computational resources required by this last step we impose a timeout of 100 seconds to its execution. If the solution obtained by the repair procedure is better than the incumbent solution we substitute the latter solution with the former one.

To diversify the solutions generated at each *CG-LNS* iteration we define three different destroy procedures that are described in the following. All these procedures try to remove $K_{\max}$ vertices from the incumbent solution and the value $K_{\max}$ is automatically tuned to intensify or diversify the search around the incumbent solution:

- *High cost subsets intersection destroy procedure*: this procedure begins by extracting a seed vertex $v \in V$ at random, using a uniform distribution. The extracted vertex is added to the set of the removed vertices R that initially is empty. After this first step, always using a random uniform distribution, the procedure extracts a vertex $i \in R$ and computes, for each vertex $j \in V \setminus R$, the proximity measure $\rho(i,j)$. This value is computed by summing up all the costs associated with subsets contained in $\mathscr{S}_i \cap \mathscr{S}_j$. After this computation, the procedure selects the next vertex v to be removed in $V \setminus R$, adopting a randomized greedy strategy (Feo and Resende, 1995): it selects v among the $\lceil \delta \mid V \setminus R \mid \rceil$ vertices associated with the highest $\rho(i,j)$ values, using a uniform random distribution. By gradually changing $\delta \in (0,1)$, we can modify the selection criterion, starting from a greedy criterion and arriving to a complete random one. After some preliminary tests, we concluded that $\delta = 0.1$ provides a right trade-off between greediness and diversification. All the previous steps, with the exception of the first initialization step, are repeated until $\mid R \mid < K_{\max}$.
- *Connectivity destroy procedure*: while the previous destroy procedure tries to remove from the incumbent solution a set of vertices having in common high cost subsets, the aim of this second procedure is to remove a set of vertices that are connected. The procedure begins by extracting a seed vertex $v \in V$, using a random uniform distribution, and inserting it in the set of removed vertices R. Then the procedure executes a breadth-first visit of G starting from v and inserts all the visited vertices in R. The breadth-first visit is interrupted as soon as $\mid R \mid = K_{\max}$.
- *Subgraph portions destroy procedure*: differently from the two previous destroy procedures, this one takes more deeply into account the structure of the incumbent solution. Initially the pool of removed vertices R is empty and the procedure initializes the set of removable vertices by setting $\tilde{V} = V$. The procedure iteratively chooses a vertex $v \in \tilde{V}$ using a uniform random distribution and it identifies the subgraph G_v in the incumbent solution that contains v. Then, it inserts in R all the vertices that can be reached from v in $\tilde{d}$ hops using only edges that belong to G_v and it removes from $\tilde{V}$ all the vertices belonging to G_v. In this way, at each iteration, the procedure considers a different subgraph of the incumbent solution. The procedure stops its execution when either $\mid R \mid = K_{\max}$ or $\tilde{V} = \emptyset$. The distance $\tilde{d}$ is dynamically updated in order to remove the correct number $K_{\max}$ of vertices at each execution of the procedure. Initially, we set $\tilde{d} = 1$ and, at the end of each execution of the procedure, we update $\tilde{d}$ by multiplying it with $\frac{K_{\max}}{|R|}$. Thus, if us-

ing the current value of $\tilde{d}$ the procedure is not able to remove $K_{\max}$ vertices, we increase the value of $\tilde{d}$ that will be used in the next execution of the procedure.

To choose which destroy procedure to apply in each *CG-LNS* iteration, we use a *Roulette-Wheel Procedure*. We denote the set of destroy procedures by Ω and we associate with each procedure $i \in \Omega$ a dynamically tuned coefficient π_i. Initially, we set $\pi_i = 1$ for each $i \in \Omega$ and, at the end of each *CG-LNS* iteration in which procedure $t \in \Omega$ is used, we increment π_t by one only if the incumbent solution has been updated. At each *CG-LNS* iteration, we execute the destroy procedure $t \in \Omega$ with a probability equal to $\frac{\pi_t}{\sum_{i \in \Omega} \pi_i}$.

The value of $K_{\max}$ that defines the number of vertices to be removed in each *CG-LNS* iteration, is dynamically updated using a self-adapting mechanism that resembles the strategy adopted in the Variable Search Neighborhood algorithms (Mladenović and Hansen, 1997). We initially set $K_{\max} = 5$ and, at the end of each *CG-LNS* iteration, if the incumbent solution has been improved we increment $K_{\max}$ by one, otherwise we reset $K_{\max}$ to its initial value. We reset $K_{\max} = 5$ also when it reaches its limit value $\mid V \mid$.

In Figure 5.15 we provide a pseudocode for the *CG-LNS* heuristic. It takes in input a HAP instance. It initializes the values of $\{\pi_i\}_{i \in \Omega}$ and $K_{\max}$. The first incumbent solution is generated by executing the Column Generation method until *PPTS* fails and by solving the final generated RMP with the integrality constraint (both steps are summarized by the *InitializeCG* function). Then, iteratively, it executes the randomly chosen destroy procedure t and the repair procedure. After these two procedures it, eventually, updates the incumbent solution and the values $\{\pi_i\}_{i \in \Omega}$ and $K_{\max}$.

```
Algorithm CG-LNS$(G, \mathscr{S}, q, Q, k)$;
for each $i \in \Omega$ do $\pi_i := 1$;
$X^* := InitializeCG(G, \mathscr{S}, q, Q, k)$
$K_{\max} := 5$;
repeat
    Choose destroy operator $t$ with probability $\frac{\pi_t}{\sum_{i \in \Omega} \pi_i}$;
    $\mathscr{A} := Destroy\ (G, \mathscr{S}, q, Q, k, t, K_{\max}, X^*)$;
    $\tilde{X} := Repair\ (G, \mathscr{S}, q, Q, k, \mathscr{A})$;
    if $\phi(\tilde{X}) < \phi(X^*)$ then
        $X^* := \tilde{X}$;
        $K_{\max} := 5$;
        $\pi_k := \pi_k + 1$;
    else
        $K_{\max} := K_{\max} + 1$;
        if $K_{\max} > \mid V \mid$ then $K_{\max} := 5$;
until termination criterion is met
return $X^*$;
```

Fig. 5.15 Pseudocode of the *CG-LNS* algorithm.

5.9.4 Computational experiments on Column Generation based heuristics

In order to compare the three different methods proposed in the previous sections to turn our Column Generation method in an effective heuristic, a first aspect to consider is the different termination criteria adopted by them. On the one hand, *CG-LDS* terminates as soon as it completes the partial exploration of the branching tree, whose structure and size is induced by its parameters $L_{\max}$ and $D_{\max}$, and it computes the optimal solution of the final ILP problem. On the other hand, both *CG-TS* and *CG-LNS* are iterative algorithms thus we can limit their execution by imposing a time limit or a maximum number of iterations. Thus, for the sake of fairness, we first executed *CG-LDS* with two different set of parameters, $L_{\max} = 3$, $D_{\max} = 2$ and $L_{\max} = 4$, $D_{\max} = 2$ which, in our preliminary tests, allow us to reach a good trade-off between the quality of the obtained solutions and the required computational resources. Then, we executed both *CG-TS* and *CG-LNS* setting a time limit according to the time required by *CG-LDS* to solve the considered instance. All the considered experiments have been executed using CPLEX 12.5 to solve the ILP subproblems on a PC equipped with an Intel Xeon Processor E5-1620 Quad Core 3.60GHz and 16 GB of RAM. The Pricing Problem heuristic *PPTS* embedded in all the three proposed hybrid heuristics has been configured to use the same parameters reported in Table 5.13 except for the stopping criteria, which have been modified setting $I_{\min} = I_{\max} = 10000$. In this case, in fact, we are mainly interested in finding a large number of columns to increase the chance that the intermediate and final ILP problems contain good feasible solutions.

In Table 5.18 we report, for the three different tested benchmarks, the average CPU time in seconds required by *CG-LDS*, varying the size of the considered instances and the discrepancy level $L_{\max} \in \{3,4\}$. Analyzing these results, we can see, that, as expected, the average CPU time increases when we consider both bigger instances and higher discrepancy levels. However, this trend is not monotonic for all the considered cases: for example, if we consider instances having $n = 40$ and $L_{\max} = 4$ the average CPU time is equal to 954.33 seconds while if we increase the instances size to $n = 50$ the average CPU time decreases to 314.17 seconds. This somewhat contradictory behavior can be explained by considering that the complexity of the final ILP problem may depend not only on the considered instances size but also on the structure of the ILP problems generated at the end of the *CG-LDS* exploration phase.

A first experiment on Column Generation based heuristics regards the improvements that *CG-LDS* obtains by optimizing the final ILP problem w.r.t. to the best solutions found during the exploration phase. In Table 5.19 we report, for each tested benchmark and for each n, the average percentage improvement associated with the considered instances class. If we denote with x^*_{EXP} the objective function value of the best solution found at the end of the exploration phase and with x^*_{FIN} the objective function value of the best solutions found during the complete *CG-LDS* execution, the reported value has been computed averaging $100(x^*_{EXP} - x^*_{FIN})/x^*_{FIN}$

Benchmark A

n	$L_{\max} = 3$	$L_{\max} = 4$
50	103.40	138.80
60	267.40	458.20
70	871.20	526.80
80	929.40	1738.20
90	1689.60	2668.60

Benchmark B

n	$L_{\max} = 3$	$L_{\max} = 4$
20	4.50	6.13
25	6.25	8.75
30	11.38	16.38
35	23.13	29.63
40	34.00	49.00
45	39.00	51.00
50	56.50	78.38
55	82.00	116.75
60	159.00	153.00
65	115.00	334.13
70	124.88	178.38

Benchmark C

n	$L_{\max} = 3$	$L_{\max} = 4$
30	13.17	15.58
40	74.83	954.33
50	187.67	314.17
60	977.67	1016.33
70	1015.67	1166.75

Table 5.18 Average CPU time in seconds required by *CG-LDS* to solve the instances of a given size in benchmarks A,B and C, using two different values for the discrepancy level $L_{\max}$ and setting $D_{\max} = 2$.

on all the considered instances. Note that for some instances *CG-LDS* is not able to identify a feasible solution during the exploration phase. Thus, the average percentage improvement is computed only on the instances for which *CG-LDS* can find a feasible solution during the exploration phase.The number of instances for which a feasible solution cannot be identified during the exploration phase is reported within parenthesis near the corresponding instance class.

Benchmark A

n	$L_{\max} = 3$	$L_{\max} = 4$
50	0.02	0.01
60	1.70	1.70
70	10.25(3)	4.93(1)
80	1.34(1)	1.50(1)
90	6.41(1)	3.47

Benchmark B

n	$L_{\max} = 3$	$L_{\max} = 4$
20	5.50	5.50
25	1.03	0.91
30	1.42	1.64
35	5.21	2.44
40	2.04	1.70
45	1.58	0.75
50	0.86(1)	0.62(1)
55	1.12	1.04
60	1.25	1.25
65	1.76(1)	1.38(1)
70	1.24	1.09

Benchmark C

n	$L_{\max} = 3$	$L_{\max} = 4$
30	1.53	0.67
40	3.52	2.47
50	3.94(1)	2.29(1)
60	1.91	1.65
70	1.27	1.25

Table 5.19 Average percentage improvement obtained by *CG-LDS* between the solutions found by solving the final ILP problem w.r.t. the solutions obtained at the end of the exploration phase.

Analyzing these results, we can see that the contribution of the final ILP optimization is not negligible, the average percentage increment reach a maximum of 10.25% on benchmark A when $L_{\max} = 3$ and $n = 50$. Moreover, on 8 instances when $L_{\max} = 3$ and on 5 instances when $L_{\max} = 4$, *CG-LDS* is not able to identify a feasible solution during the exploration phase. Finally, passing from $L_{\max} = 3$ to $L_{\max} = 4$

the number of unsolved instances during the exploration phase decreases as well and the average percentage improvements obtained by the last ILP optimization, in general, decrease. This behavior can be explained considering that, by increasing the discrepancy level, the number of processed nodes increases and the probability to find an integer solutions during the exploration phase grows accordingly.

To study the efficiency of the diversification strategies adopted by *CG-TS* and *LNS-TS*, we report in Table 5.20 (*CG-TS*) and in Table 5.21 (*LNS-TS*), for each tested benchmark and for each instances class, the total number of iterations that can be executed in the given amount of time and the number of the iteration in which the considered algorithm found the best solution.

Benchmark A

	$L_{\max} = 3$		$L_{\max} = 4$	
n	IT	BIT	IT	BIT
50	15.80	4.00	21.00	8.00
60	24.00	14.00	37.00	14.00
70	36.20	6.80	27.20	6.80
80	53.80	20.60	127.60	20.60
90	98.20	47.80	104.60	47.40

Benchmark B

	$L_{\max} = 3$		$L_{\max} = 4$	
n	IT	BIT	IT	BIT
20	12.50	2.13	16.13	2.13
25	13.00	1.44	17.33	1.44
30	17.89	7.11	24.33	7.67
35	27.22	7.89	32.67	7.89
40	33.11	8.00	44.33	8.00
45	38.00	3.00	48.44	6.11
50	39.56	7.89	51.56	12.89
55	51.78	15.78	69.89	24.89
60	78.22	27.78	82.56	32.56
65	59.33	21.00	139.00	32.00
70	60.00	20.11	83.78	28.00

Benchmark C

	$L_{\max} = 3$		$L_{\max} = 4$	
n	IT	BIT	IT	BIT
30	14.25	3.42	17.92	3.42
40	39.83	15.25	835.17	19.00
50	79.67	29.25	129.75	41.75
60	242.67	123.00	266.08	123.00
70	228.08	129.50	305.83	170.17

Table 5.20 Average total number of iterations (Column IT) and average number of the iteration in which *CG-TS* found the best solution (Column BIT), for each considered benchmark and each tested discrepancy level.

The first observation that we can make considering these results is that on both benchmarks B and C, to execute a single iteration, on average, *CG-TS* requires less time w.r.t. *CG-LNS*. On the contrary, if we consider benchmark A, *CG-LNS* is faster than *CG-TS*. These differences can be explained considering the different structures of the tested benchmarks (A contains realistic instances while B and C contain random instances) and the different ways in which the two considered algorithms insert constraints in the original MCE_{LP} (5.24) formulation.

For what concerns the ratio between the number of the iteration in which the two heuristics found the best solution and the total number of executed iterations, the two considered heuristics obtain similar results: the efficiency of the two diversification mechanisms is similar.

In the final experiment, in addition to the three Column Generation based heuristics, we consider also the local search heuristic *VLNS-TS* described in Section 5.8, setting to $T = 5000$ the number of iterations for each restart. Similarly to what done in the previous experiments, *CG-TS*, *CG-LNS* and *VLNS-TS* have been executed

Benchmark A

	$L_{\max} = 3$		$L_{\max} = 4$	
n	IT	BIT	IT	BIT
50	28.00	7.60	32.80	7.60
60	40.60	23.20	53.80	23.20
70	96.00	34.40	76.00	34.40
80	87.00	43.60	144.00	43.60
90	146.20	116.00	226.00	171.60

Benchmark B

	$L_{\max} = 3$		$L_{\max} = 4$	
n	IT	BIT	IT	BIT
20	5.50	0.25	8.75	2.00
25	6.50	0.00	10.25	1.13
30	9.75	1.88	15.63	3.75
35	17.00	4.50	22.63	5.75
40	19.38	0.88	31.13	8.25
45	20.75	1.38	29.13	4.38
50	24.13	1.38	34.00	4.75
55	30.13	11.25	46.50	15.00
60	41.63	17.88	49.88	21.25
65	36.75	13.88	95.75	18.38
70	37.25	19.75	58.38	32.75

Benchmark C

	$L_{\max} = 3$		$L_{\max} = 4$	
n	IT	BIT	IT	BIT
30	16.08	3.67	16.08	3.67
40	46.17	11.83	46.17	11.83
50	68.83	8.17	68.83	8.17
60	128.00	64.42	128.00	64.42
70	132.83	63.00	132.83	63.00

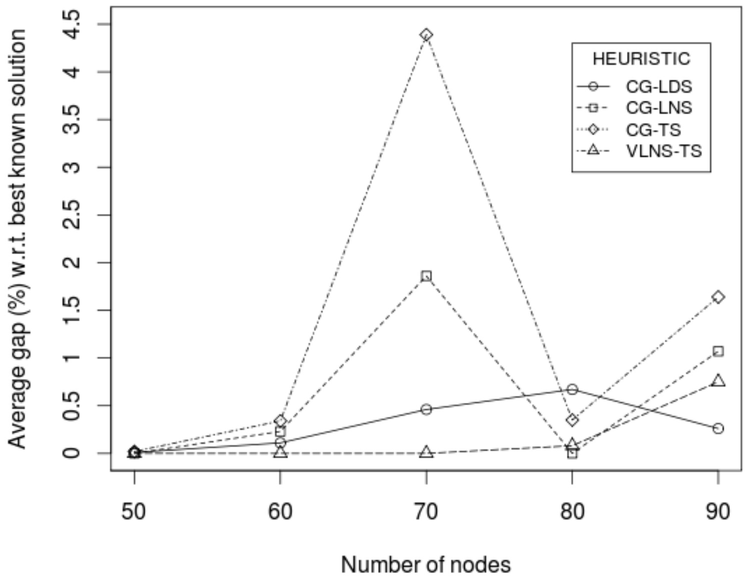
Table 5.21 Average total number of iterations (Column IT) and average number of the iteration in which *LNS-TS* found the best solution (Column BIT), for each considered benchmark and each tested discrepancy level.

setting a time limit equal to the time required by *CG-LDS* on the corresponding instance.

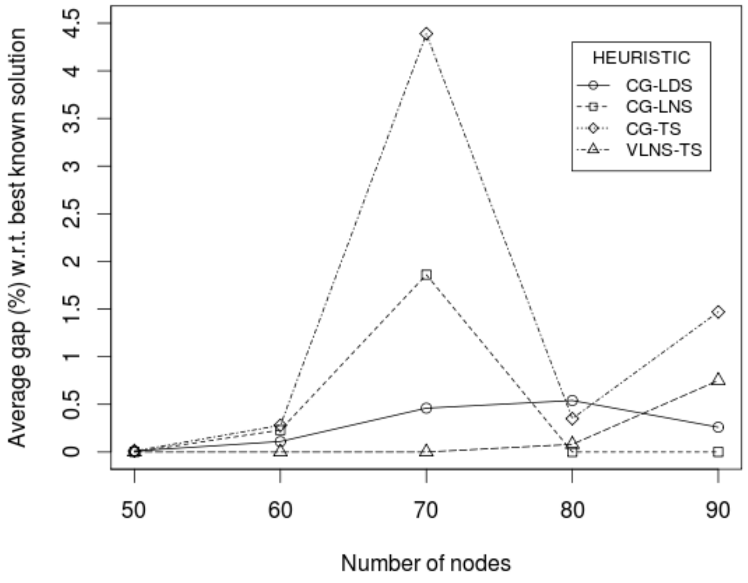
Figure 5.16, Figure 5.17 and Figure 5.18, for both the considered parameter settings of *CG-LDS* (i.e. $L_{\max} = 3$ and $L_{\max} = 4$), contain plots comparing the average gap between the best known solutions and the best solutions found by the four different tested heuristics, varying the size of the considered instances.

Analyzing these results, it is evident that the heuristics that achieve the best results are *CG-LDS* and *VLNS-TS*. In particular the former has some difficulties to identify good solutions for small instances, while if we increase the instance size *CG-LDS* is able to find solutions of better quality w.r.t. the ones found by *VLNS-TS*. This behavior is particularly noticeable if we consider benchmark B (see Figure 5.17) that contains instances with size starting from $n = 20$.

The two other heuristics that we considered (i.e. *CG-TS* and *CG-LNS*) are often outperformed by *VLNS-TS* and *CG-LDS*. However, *CG-LNS* seems to be the heuristic that employs better the larger time associated with *CG-LDS* parameters $L_{\max} = 4$, $D_{\max} = 2$ w.r.t. the time associated with *CG-LDS* parameters $L_{\max} = 3$, $D_{\max} = 2$: on average, when we increase the time limit, the average gaps obtained by *CG-LNS* decrease by 20.00% on benchmark A, 25.90% on benchmark B and 17.48% on benchmark C, while the average gaps obtained by *CG-TS* decrease only by 13.62% on benchmark A, 10.51% on benchmark B and 7.69% on benchmark C. The increment of *CG-LNS* performance obtained increasing the time limit, allows to *CG-LNS*, on some instances, to be competitive with *CG-LDS* and *VLNS-TS*: for example, on benchmark A, *CG-LNS* obtains the lowest percentage gap on average for instances having $n \geq 80$.

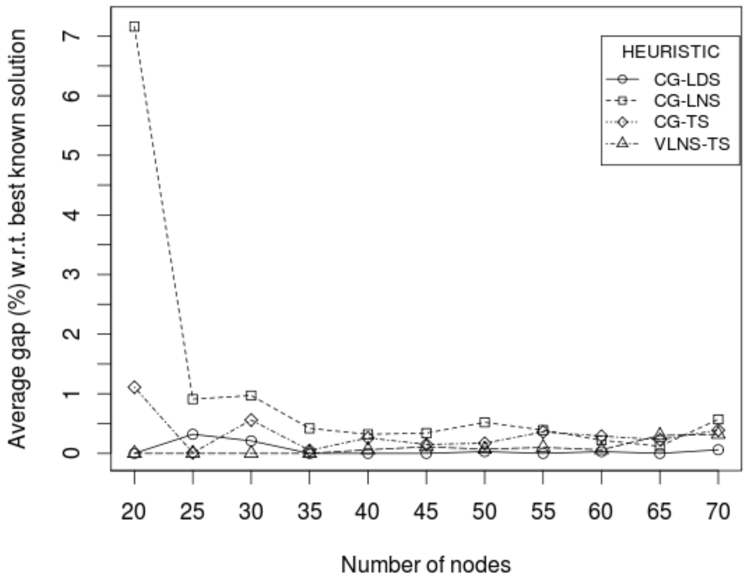


$L_{\max} = 3$, $D_{\max} = 2$

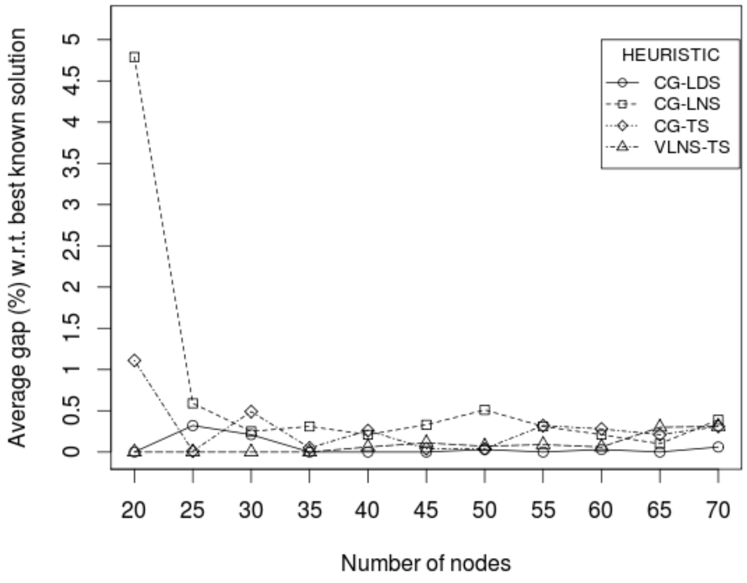


$L_{\max} = 4$, $D_{\max} = 2$

Fig. 5.16 Average percentage gap between the best known solutions and the solutions obtained by the four different considered heuristics on benchmark `A` in a time equal to the one used by *CG-LDS*, with parameters, respectively, equal to $L_{\max} = 3$, $D_{\max} = 2$ and $L_{\max} = 4$, $D_{\max} = 2$.

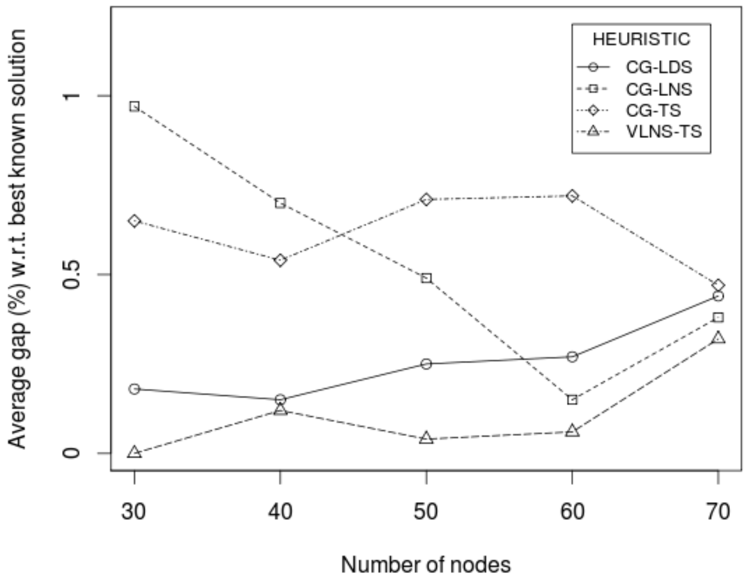


$L_{\max} = 3$, $D_{\max} = 2$

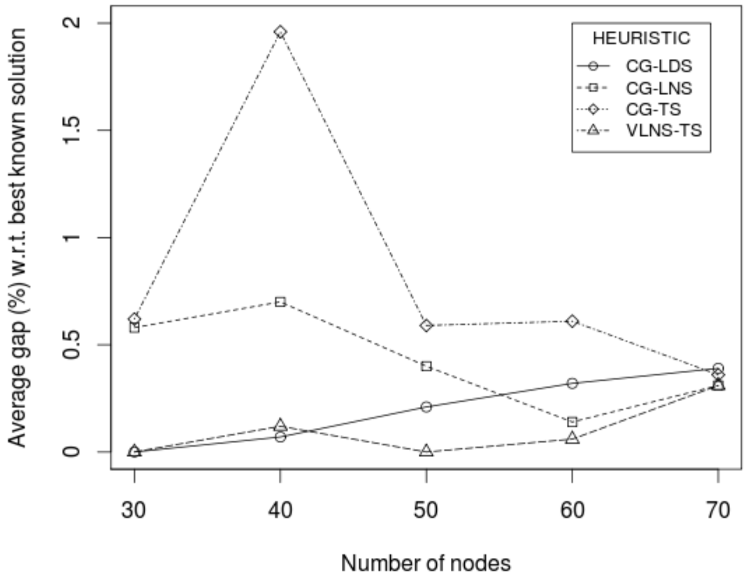


$L_{\max} = 4$, $D_{\max} = 2$

Fig. 5.17 Average percentage gap between the best known solutions and the solutions obtained by the four different considered heuristics on benchmark B in a time equal to the one used by *CG-LDS*, with parameters, respectively, equal to $L_{\max} = 3$, $D_{\max} = 2$ and $L_{\max} = 4$, $D_{\max} = 2$.



$L_{\max} = 3$, $D_{\max} = 2$



$L_{\max} = 4$, $D_{\max} = 2$

Fig. 5.18 Average percentage gap between the best known solutions and the solutions obtained by the four different considered heuristics on benchmark C in a time equal to the one used by *CG-LDS*, with parameters, respectively, equal to $L_{\max} = 3$, $D_{\max} = 2$ and $L_{\max} = 4$, $D_{\max} = 2$.

5.10 Computational experiments on real world instances

We start to investigate on the HAP because we were required to effectively partition two real provinces: the province of Monza and Brianza having 55 towns and 426 activities (thus the corresponding HAP instance is such that $\mid V \mid = 55$ and $\mid \mathscr{S} \mid = 481$) and the bigger instance of Milan having 134 towns and 774 activities (thus the corresponding HAP instance is such that $\mid V \mid = 134$ and $\mid \mathscr{S} \mid = 908$). In this section we provide a description of what can be obtained executing the methods proposed in the previous part of this chapter on these two real world instances. All the experiments reported in this section have been executed on a PC equipped with an Intel Xeon Processor E5-1620 Quad Core 3.60GHz and 16 GB of RAM and using CPLEX 12.5 as ILP solver . Thanks to its small size, the Monza instance can be exactly partitioned by both the exact methods proposed in Section 5.6: using CPLEX with formulation MCA (5.2) and all the valid inequalities described in Section 5.6.1 we can solve the Monza instance in 3 seconds and 83 branching nodes; similarly, using formulation MCN (5.6) with all the valid inequalities described in Section 5.6.3 we can solve the problem in 3 seconds and 63 branching nodes. If we execute the Column Generation method described in Section 5.7.1, we can solve the LP-Relaxation of the extended formulation $\mathrm{MCE}_{\mathrm{LP}}$ (5.22) in 81 seconds obtaining a lower bound equal to 382.148 and, by solving the final Reduced Master Problem with integrality constraint, we can obtain the optimal solution equal to 419.21 in 5 seconds. Note that the performance of CPLEX using the compact ILP formulations cannot be directly compared with the performance of our Column Generation method since the *PPTS* heuristic (that consumes a large amount of the computational resources required by the Column Generation method) is not multi-threaded, while CPLEX can efficiently use all the 4 cores available on our PC. Nonetheless, the obtained results show that, for small real instances like the Monza one, the compact ILP formulations are the right tool to achieve the optimal solution in a small computation time.

The bigger instance of Milan cannot be solved exactly in a reasonable amount of time: in one hour of computation time, using formulation MCA (5.2) CPLEX is able to process only 79 branching nodes without finding any feasible solution and obtaining a best lower bound equal to 3943.79. As already discussed in Section 5.6.4, formulation MCN (5.6) is tighter than formulation MCA (5.2). Thus, is not surprising that, within one hour, using it, CPLEX is able to process 2245 branching nodes, obtaining a best lower bound equal to 8048.32. However, also using this tighter formulation, CPLEX is not able to find a feasible integer solution.

With such a big instance, we can obtain a better result using the Column Generation method: it provides a lower bound equal to 9668.84 in 2229 seconds, and, to solve the final RMP with integrality constraint, CPLEX required 17 seconds of CPU time, providing a feasible solution having the objective function value equal to 10019.9.

To test the heuristics described in Section 5.8 and in Section 5.9 on Milan instance, we started by executing *CG-LDS* obtaining a best solution with an objective function value equal to 9785.58 considering both discrepancy levels $L_{\max} = 3$ and $L_{\max} = 4$. However, if we set $L_{\max} = 3$, *CG-LDS* requires 719 seconds to complete, while, if we set $L_{\max} = 4$, the required time is equal to 1098 seconds. Then, using these two different time limits, we tested the other three heuristics we developed. All the tested heuristics are not able to exploit the larger time limit associated with $L_{\max} = 4$ and the best solution they found does not improve if consider a time limit equal to 1098 seconds instead of 719 seconds. The best solution found by *CG-TS*, *CG-LNS* and *VLNS-TS* have an objective function value, respectively, equal to 9787.28, 9968.19 and 9860.83. Thus, while the experiments reported in Section 5.9.4 show that *VLNS-TS* is competitive with *CG-LDS*, on the Milan instance, the latter algorithm outperforms the former one.

5.11 Conclusion

In this chapter we considered a Graph Partitioning Problem which models the partition of an organization into administrative areas, named the HAP. We started our investigation considering two compact formulations for the HAP that can be directly solved using a commercial ILP solver. The first compact formulation derived from a natural formulation of the classical Graph Partitioning Problem but it presents several symmetry-related issues. To overcame these issues we developed valid inequalities for the considered formulation but we also proposed a different compact formulation that, with a slight increment in the number of variables and constraints, solve some of these issues.

Nonetheless, both the compact formulations cannot scale to realistic size instances, thus we developed a new extended formulation and we propose a Column Generation method that allows us to obtain tight lower bounds also for realistic instances (from 70 to 90 vertices). To achieve these results we needed to combine exact and heuristic methods to solve the $\mathcal{NP}$-hard Pricing Problem associated with the extended formulation. Since the Pricing Problem is similar to the KPC-STP investigated in Chapter 3 the two exact methods we introduce to solve it have been directly derived from the single-commodity flow formulation described in Section 3.4.1.

Finally, following what we described in Section 2.3, we propose three different ways to embed our Column Generation method in an effective heuristic. We compared the obtain heuristics with a finely tuned local search based heuristic and we found that the best one among the hybrid heuristics is competitive with the local search one, especially when the number of vertices in the considered instance increases.

5.12 References

R. K. Ahuja, Ö. Ergun, J. B. Orlin, and A. P. Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75 –

102, 2002.

S. Arora, S. Rao, and U. Vazirani. Geometry, flows, and graph-partitioning algorithms. *Commun. ACM*, 51(10):96–105, 2008.

V. Cacchiani, V. C. Hemmelmayr, and F. Tricoire. A set-covering based heuristic algorithm for the periodic vehicle routing problem. *Discrete Applied Mathematics*, 2012. doi: 10.1016/j.dam.2012.08.032.

M. Campêlo, V. A. Campos, and R. C. Corrêa. On the asymmetric representatives formulation for the vertex coloring problem. *Discrete Applied Mathematics*, 156 (7):1097 – 1111, 2008.

A. Ceselli, F. Colombo, R. Cordone, and M. Trubian. Employee workload balancing by graph partitioning. *Discrete Applied Mathematics*, 2013. doi: 10.1016/j.dam.2013.02.014.

F. Colombo, R. Cordone, and M. Trubian. On the partition of an administrative region in homogenous districts. In *Atti del Convegno AIRO 2011*, Brescia, Italy, 2011.

F. Colombo, R. Cordone, and M. Trubian. Upper and lower bounds for the homogenous areas problem. In *Proceedings of the 11th Cologne-Twente Workshop on Graphs and Combinatorial Optimization*, München, Germany, 2012.

F. Colombo, R. Cordone, and M. Trubian. Column-generation based bound for the homogeneous areas problem, 2013. Submitted to European Journal of Operational Research, under second review round.

W. E. Donath and A. J. Hoffman. Lower bounds for the partitioning of graphs. *IBM Journal of Research and Development*, 17(5):420–425, 1973.

N. Fan and P. Pardalos. Linear and quadratic programming approaches for the general graph partitioning problem. *Journal of Global Optimization*, 48:57–71, 2010.

T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133, 1995.

C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. Formulations and valid inequalities for the node capacitated graph partitioning problem. *Mathematical Programming*, 74(3):247–266, 1996.

C. E. Ferreira, A. Martin, C. C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem: A computational study. *Mathematical Programming*, 81:229–256, 1998.

P. O. Fjallström. Algorithms for graph partitioning: A survey. *Linköping Electronic Articles in Computer and Information Science*, 10, 1998.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

F. Glover. Ejection chains, reference structures and alternating path methods for traveling salesman problems. *Discrete Applied Mathematics*, 65(1–3):223–253, 1996.

F. Glover and J. K. Hao. The case for strategic oscillation. *Annals of Operations Research*, 183:163–173, 2011.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

N. Guttmann-Beck and R. Hassin. Approximation algorithms for minimum *k*-cut. *Algorithmica*, 27(2):198–207, 2000.

T. Ideker, O. Ozier, B. Schwikowski, and A. F. Siegel. Discovering regulatory and signalling circuits in molecular interaction networks. *Bioinformatics*, 18(suppl 1):S233–S240, 2002.

C. Joncour, S. Michel, R. Sadykov, D. Sverdlov, and F. Vanderbeck. Primal heuristics for branch-and-price. In *European Conference on Operational Research (EURO'10)*, volume 1, page 2, 2010.

V. Kaibel and M. Pfetsch. Packing and partitioning orbitopes. *Mathematical Programming*, 114(1):1–36, 2008.

J. Kim, I. Hwang, Y. H. Kim, and B-R. Moon. Genetic approaches for graph partitioning: a survey. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, GECCO '11, pages 473–480. ACM, 2011.

T. L. Magnanti and L. A. Wolsey. Optimal trees. In C. L. Monma M. O. Ball, T. L. Magnanti and G. L. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503 – 615. Elsevier, 1995.

S. Martello and P. Toth. *Knapsack problems: algorithms and computer implementations*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1990.

D. W. Matula and F. Shahrokhi. Sparsest cuts and bottlenecks in graphs. *Discrete Applied Mathematics*, 27(1-2):113 – 123, 1990.

N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

V. Osipov, P. Sanders, and C. Schulz. Engineering graph partitioning algorithms. In Ralf Klasing, editor, *Experimental Algorithms*, volume 7276 of *Lecture Notes in Computer Science*, pages 18–26. Springer, 2012.

D. Pisinger and S. Røpke. Large neighborhood search. *Handbook of metaheuristics*, pages 399–419, 2010.

E. Prescott-Gagnon, G. Desaulniers, and L. M. Rousseau. A branch-and-price-based large neighborhood search algorithm for the vehicle routing problem with time windows. *Networks*, 54(4):190–204, 2009.

S. Røpke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4): 455–472, 2006.

P. Sanders and C. Schulz. Engineering multilevel graph partitioning algorithms. In C. Demetrescu and M. Halldórsson, editors, *Algorithms - ESA 2011*, volume 6942 of *Lecture Notes in Computer Science*, pages 469–480. Springer, 2011.

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Principles and Practice of Constraint Programming — CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Springer, 1998.

P. M. Thompson and H. N. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41(5):935–946, 1993.

F. Vanderbeck. On dantzig-wolfe decomposition in integer programming and ways to perform branching in a branch-and-price algorithm. *Operations Research*, 48 (1):111–128, 2000.

Chapter 6
The Train Design Optimization Problem

6.1 Introduction

Freight rail companies operate on rail networks consisting of rail yards connected by rail tracks. They receive requests from customers to transport railcars and they generate the so called *trip plan* which details the movement of each railcar on the network from its origin location to its final destination. The generation of trip plans is one of the most fundamental and difficult problems encountered in the rail industry. In the USA rail companies can daily operate up to 200 merchandise trains, which involve approximately from 400 to 500 crews and transport about $1,000$ blocks of railcars, loading and unloading them at 180 to 200 locations.

This very complex decision process implies several steps, that have been considered in the literature either separately or in various combinations. The *Block Design* (BD) problem, see Ahuja et al. (2007), requires to group the railcars into *blocks*, which will then be treated as unsplittable units for the sake of efficiency. The *Block-to-Train Assignment* (BTA) problem, see Jha et al. (2008), given a set of blocks and trains, requires to decide which blocks will be transported by which train. Notice that in general a block uses more than one train to reach its destination, because it is common to unload entire blocks of railcars from a train stopping at a rail yard and load them on another train, thus performing a so called *block swap*. The *Train Routing* (TR) problem, see Goossens et al. (2004), requires to decide the route followed by each train (origin, destination and stops), consistently with the rail network and with the blocks to be transported. Sometimes, the TR problem is also concerned with the frequency with which a train is operated during the time horizon. Other relevant decisions concern the empty car distribution and the variability of demand over time (Haghani, 1989), the loading pattern of containers on railcars (Corry and Kozan, 2008) and the definition of the crews operating the trains (Albers, 2009). At a later stage, an additional scheduling problem, to define the timetable of each train, and a dispatching problem, to define meets and overtakes between different trains, must also be solved. These problems possibly interact with the planning of passenger trains which share the same network (Cordeau et al., 1998).

As properly highlighted by all surveys, from the early ones (Assad, 1980; Haghani, 1987) to the more recent one (Ahuja et al., 2005), these aspects of rail transportation have strong mutual interactions. However, models that take them into account quickly get extremely complex, if not simply intractable. The traditional approach has thus been to isolate and solve independently the single stages of the overall planning problem, which are often themselves hard and interesting, and require sophisticated algorithmic approaches. This leads naturally to decomposition methods, in which the solution of each stage provides the data for the following ones. The obvious drawback of such methods is that they necessarily lead to suboptimal decisions, though usually much better than hand-made decisions. For example, Crainic and Rousseau (1986) implement a decomposition method to generate optimal operating plans through a Column Generation method, while Jha et al. (2008) focus on the BTA problem, presenting another method based on Column Generation. Several methods aim to a partial integration of different stages. Keaton (1989) applies the Lagrangian Relaxation to solve a combination of the BD and BTA problems for a given set of potential trains, whose service frequency must also be determined. This solution technique is later improved in Keaton (1992). Haghani (1989) solves a train routing and makeup problem with a nonlinear objective function and linear constraints, using a heuristic decomposition technique based on a Mixed Integer Programming (MIP) model. Ahuja et al. (2005) solve a *train scheduling problem*, which combines the BTA and the TR problems with the determination of the weekly and daily schedule of the trains. However, in spite of this and several other nice contributions, the field of railway optimization is far from being exhausted. A recent survey, in fact, still regrets that "none of these attempts has produced a solution procedure that rails can use, because they are not scalable for realistically large train scheduling problems, or they ignore the practical realities necessary to generate implementable solutions" (Ahuja et al., 2005).

Starting from 2010, the Railway Application Section of INFORMS sponsored each year a problem solving competition aiming to introduce participants to the many challenging Operations Research problems arising within railroad companies[1]. In 2011, the subject of the competition has been the the *Train Design Optimization* (TDO) problem. This problem has been chosen and formally described by a group of Operations Research professionals working in the major railroad companies of the USA. Among the companies which sponsored the competition appear Norfolk Southern Corporation, BNSF Railway Corporation, CSX Corporation, Union Pacific Corporation (i.e. 4 of the 8 first class railroad companies operating in the United States) and Gurobi Optimization Inc. The competition started in May and all the teams needed to submit their final report within the beginning of September. Overall, 35 teams started the competition, but only 12 teams submitted their final reports within the deadline. The participants came from many different countries: United States, China, Taiwan, Singapore, Switzerland, Italy, Colombia and Mexico. After the deadline, the competition organizing committee selected three finalists that presented their works at the 2011 Informs Annual Meeting in Charlotte, NC.

[1] For further details, see the official website `http://www.informs.org/Community/RAS/Problem-Solving-Competition/`.

The TDO problem combines the BTA and the TR problems with the definition of feasible and efficient train routes with respect to the available *crew segments*. These are fixed pairs of locations between which crews are available to operate trains[2]. In more detail, the composition and the attributes of each block of railcars, such as origin, destination, number of cars, length and tonnage, are assumed to be known in advance. On the contrary, the assignment of blocks to trains, their swaps, the routes and stops of the trains, and the crews operating each train along each part of its route must be determined. In fact, a single train is usually operated by several crews, who take over from each other, so that its route is always a sequence of crew segments, even if this implies moving part of the way without carrying any block. As far as we know, this specific combination of block assignment and train routing is new for the literature.

In this work, we describe our Simultaneous Column and Row Generation heuristic (see Section 2.4), which won the second prize at the *RAS-2011* competition. In the Master Problem the main columns represent either trains or paths followed by a block: a train is represented as a sequence of crew segments, while a block-path is a sequence of arcs in an auxiliary graph. We generate the train columns with a two level Tabu Search heuristic (Glover and Laguna, 1997) and the block-path columns by solving a net-flow model on another auxiliary graph. At the end of the Column Generation method, we impose integrality to the variables of the Master Problem and solve the resulting MIP model. The algorithm here discussed has been strongly enhanced after the competition deadline, reducing the computational time by one order of magnitude while improving significantly the quality of the results.

The next section formally introduces the TDO problem. Section 6.3 describes the three methods which respectively won the first prize (Lozano et al., 2011), the second prize (Wang et al., 2011) and the honorable mention (Jin and Zhao, 2011; Jin et al., 2013) at the *RAS-2011* competition. Section 6.4 presents the MIP formulation of the problem we used in our approach. Section 6.5 presents our Simultaneous Column and Row Generation approach, and the last section discusses the computational results on the benchmark data sets used for the competition.

6.2 Problem description

A rail network consists of rail yards connected by rail tracks. It can be modeled as an undirected graph where the nodes represent rail yards and the edges rail tracks. However, since the orientation of train routes and block-paths is relevant for the TDO problem, we adopt a directed graph $G = (V, A)$ where the node set V is the set of rail yards and the arc set A is derived from the set of the rail tracks associating a couple of arcs with opposite directions to each rail track. Without ambiguity, we can still call rail track an arc. Each rail track $a \in A$ has a length l_a. Let S be the set of available crew segments: each crew segment $s \in S$ is associated with an unordered

[2] This is a simplified model, called single-ended territories, of the wide and complex variety of union agreements holding in the rail industry.

pair of rail yards, $v_1(s), v_2(s) \in V$, called *terminal nodes*. Given graph G and the lengths of its arcs, we associate to each crew segment s the set of directed shortest paths from $v_1(s)$ to $v_2(s)$ and the set of directed shortest paths from $v_2(s)$ to $v_1(s)$ (in general, there could be multiple shortest paths connecting the same pair of vertices). Let B be the set of blocks to be transported: each block $b \in B$ contains N_b railcars, with a total length L_b and a total weight W_b; its origin and destination rail yards are denoted, respectively, as o_b and d_b.

The route of a train is a path on G, obtained concatenating directed shortest paths associated with crew segments. In the following, if there is no ambiguity, we can also denote a train with the corresponding sequence of crew segments $(s_1, s_2, \dots, s_k)$. Note that the train route can contain cycles, and even several repetitions of the same shortest path are admitted.

The definition of a train route also includes a sequence of *work events*. These correspond to the stops in which the train loads or unloads blocks of railcars, with the exception of the origin and the destination of the route. In other words, every time a train makes an intermediate stop, all pickups and deliveries performed during that stop are considered as a single work event, while the departure of a train from the origin and its arrival to the destination are not considered as work events, even if they involve loading or unloading blocks.

Finally, each served block b is carried, by one or more trains, along a path from o_b to d_b on graph G which we call *block-path*. If more trains are used, the operation of unloading b from a train and loading it on another one is called *block swap*.

6.2.1 Operational and capacity constraints

The following operational and capacity constraints must be respected:

- **Maximum number of blocks**: a train carrying too many blocks along its trip can produce excessive slowdowns and can increase too much the length of crew shifts. As a consequence, the overall number of different blocks that a train can carry is limited to MB.
- **Maximum number of work events**: an excessive number of train stops is discouraged for operational efficiency. Thus the maximum number of work events for each train is set to ME.
- **Maximum number of block swaps**: each block admits at most MS block swaps along its block-path: block swaps increase flexibility in train routing, but require additional time and resources.
- **Rail tracks structural limits**: each arc $a \in A$ admits a maximum train length ML_a and a maximum train weight MW_a, depending on geographical and track attributes.
- **Rail tracks congestion avoidance**: each arc $a \in A$ admits a maximum number of trains, MN_a, passing through it, in order to avoid congestion.

Time is not a concern in this model: when a block is unloaded at an intermediate yard of its path, there is an implicit assumption that the train in charge of carrying it further will load it in a future time period, if it does not in the current one. Note that the congestion level of the rail network is globally handled by correctly setting the maximum number of trains MN_a that can cross each arc $a \in A$ during the whole time period scheduled by the current trip plan.

6.2.2 *Objective function*

The objective of the TDO problem is to minimize the sum of several cost components. To describe them, hereafter we denote with p a generic feasible block-path, with b_p the corresponding block and with t a generic feasible train. We also denote by $P(\cdot)$ a generic sequence of properly oriented (and possibly repeated) arcs from A. The arc sequence followed by train t is $P(t)$, that of block-path p is $P(p)$. Any shortest path associated to crew segment s is denoted as $P(s)$, specifying when necessary its starting and ending nodes. Lastly, $L_{P(\cdot)}$ is the total length of the arcs in $P(\cdot)$. The objective function has the following components:

- **train activation cost**: it is the product of the fixed train start cost CL times the total number of used trains.
- **train travel cost**: it is the product of the train cost per unit of distance CT times the total travel distance of a train t, i. e. $CT \cdot L_{P(t)}$.
- **work event cost**: we denote with $WE(t)$ the set of work events associated with train t. The work event cost for t is the product of the number of work events times a fixed cost term CW, i. e. $|WE(t)|CW$. Term CW models the cost of loading and unloading blocks, but also of delaying the train (railcars, locomotives, and crews) and the potential consumption of network capacity while the train is stopped.
- **block travel cost**: given a block-path p, it is the product of the railcar cost per unit of distance CR, times the number N_{b_p} of railcars grouped in block b_p, times the total railcar travel distance, i. e. $CR \cdot N_b L_{P(p)}$.
- **block swap cost**: we denote with $BS(p)$ the set of rail yards along block-path p in which a block swap occurs. The block swap cost for p is the sum of the swap costs, CS_v, for all these yards, i. e. $\sum_{v \in BS(p)} CS_v$.
- **crew imbalance cost**: we denote with $\Delta_s(t)$ the difference between the number of times in which train t goes through a shortest path associated with crew segment s from $v_1(s)$ to $v_2(s)$ and the number of times in which the same train t goes through a shortest path associated with the same crew segment s, but now going in the opposite direction, from $v_2(s)$ to $v_1(s)$. If we sum $\Delta_s(t)$ over all the trains and take the absolute value of the result, we obtain the *crew imbalance* Δ_s of crew segment s, which is the absolute difference between the number of times crew segment s is used from $v_1(s)$ to $v_2(s)$ and the number of times it is used in the opposite way. The crew imbalance penalty IS is the average cost per unit of this imbalance, corresponding to the additional expense for repositioning a crew with an over-the-road taxi service. The crew imbalance cost is the sum over all

crew segments s of the crew imbalance penalty times the *crew imbalance*, i. e. $\sum_{s \in S} IS\Delta_s$.

- **train imbalance cost**: let $\Delta_v(t)$ be equal to 1 if train t starts its route in yard v and equal to -1 if train t ends its route in v. If we sum $\Delta_v(t)$ over all the trains and take the absolute value of the result, we obtain the *train imbalance* Δ_v at rail yard v, which is the absolute difference between the number of trains originating and terminating at the yard (the intermediate stops of the trains do not contribute to this value). The train imbalance cost is the sum over all rail yards v of the train imbalance penalty IT times the *train imbalance*, i. e. $\sum_{v \in V} IT \cdot \Delta_v$.This component of the objective function takes into account the cost that a railroad company needs to pay to reposition its locomotives in order to have the same number of locomotives in the same rail yards w.r.t. the initial situation, when the trip plan started;
- **missed railcars cost**: it is the product of the cost per missed railcar MR times the number of missed railcars, which is the sum of N_b over all missed blocks (a block is missed if it is not transported from its origin to its destination).

Example

Figure 6.1 provides an example to clarify some of the concepts introduced above. It reports in (a) the graph G modeling a rail network, where the terminal nodes of the available crew segments are filled in black, and in (b) the set S of the available crew segments, described by the bold black edges. In (c), the shortest paths on G associated to each crew segment in S are represented as sequences of thinner arrows. Full and dashed lines are used to distinguish paths oriented in opposite directions. Notice that one of the crew segments admits four equivalent shortest paths, two in each direction. In (d), we show two trains. The former, in continuous bold lines, starts from node A, moves to B, comes back to A and ends in C, using twice crew segment (A,B). The latter, in dashed bold lines, starts from B, moves to D and ends back in B, using twice crew segment (B,D). In (e) a block is carried from its origin (marked in gray) to its destination (marked in black). The block-path is represented with a sequence of smaller arrows and uses both the trains introduced above, with a block swap in node B. This implies a work event for the second train, because the block is unloaded and then loaded in B, which is neither the destination nor the origin of the second train. No work event occurs for the first train, because B is both the origin and the destination of that train. Notice that the first train is unloaded during the initial part of its route, the second train is unloaded during the final part. The imbalance of crew segments (B,D) and (A,B) is zero, whereas the imbalance of (A,C) is one. The train imbalance at rail yard B is zero, since one train originates and one terminates in B (the same train). The train imbalance at rail yards A and C is one, since one train originates in A and one terminates in C.

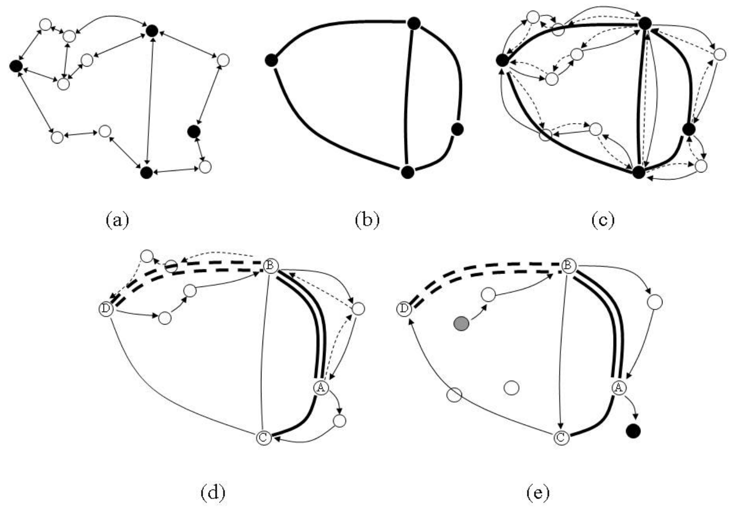


Fig. 6.1 The auxiliary graphs for a simple TDO instance. In (a) the vertices correspond to rail yards and the arcs correspond to rail tracks, black vertices denote the terminal nodes of the available crew segments. In (b) the nodes correspond to the terminal nodes of the available crew segments and the edges defines how these terminal nodes are connected. In (c) with thin and dashed arcs we denote the shortest paths associated with the available crew segments. In (d) using dashed bold lines and continuous bold lines we denote two different train routes. In (e) we show how a block originating in the gray vertex can be transported to its destination in the black vertex using two different trains.

6.3 Competing algorithms

Here we briefly summarize the three methods which won the first prize, the third prize and the honorable mention at the *RAS-2011* competition, as they are presented in Lozano et al. (2011), Wang et al. (2011) and Jin et al. (2013).

Lozano et al. (2011) propose a noniterative heuristic method working in four steps. The first three steps identify a set of train routes, the last step solves the BTA problem restricted to those routes. The first step (*Crew Demand Estimation*) focuses on selecting a set of routes that provide shortest paths for the most valuable blocks: after assigning a unique path to each crew, they propose a MIP model to compute the number of crews for each segment ideally required to route all (transportable) blocks through their shortest paths. The second step (*Train Route Generation*) creates a large pool of random train routes using the crews computed in the first step. The third step (*Train Route Selection*) selects from the pool, by solving an auxiliary set covering problem, a subset of train routes that minimizes the block-independent cost components (i. e., train start, crew imbalance, train imbalance, and train travel cost), subject to the crew demand satisfaction. The fourth and final step allocates the blocks to the selected train routes, tackling the BTA problem with a specialized multi-commodity flow model. This model provides the final solution.

Wang et al. (2011) generate sets of block-paths and associated train routes by applying *k*-shortest path algorithms on suitable graphs. Then, they calculate the best block swapping plan by solving a MIP model. They propose a specialized minimum cost integer multi-commodity flow model, in which each block is viewed as a commodity with a given origin and destination. The model involves two subproblems: (1) a vehicle routing problem that allows blocks to be swapped during the transportation, and (2) a fleet assignment and routing problem that defines the size and route of trains. If one could provide all the feasible block-paths and train routes, then the resulting model would provide optimal TDO solutions. In practice, the model cannot be solved due to its huge size, so that the authors first generate good block-paths, then use heuristics to generate good train routes and finally solve the corresponding reduced model. They repeat these steps by iteratively feeding new block-paths and train routes to the model, solving the reduced problem exactly or approximately, until the solution converges to a certain satisfactory level. New block-paths and new train routes correspond to new sets of variables and constraints. This approach is much similar to ours, with its iterative generation of blocks-paths and train routes. The main difference is that it does not use dual information, and therefore cannot be classified as a Simultaneous Column and Row Generation method.

Finally, Jin et al. (2013) use a hierarchical approach which consists of two stages. In the first one, they use a Column Generation method to find a small set of good train routes and, simultaneously, a feasible assignment of the crews to the trains. In the second one, they solve the remaining BTA problem using an *ad hoc* MIP model. The model used in the first stage requires to minimize all the cost components excluding those related to block swaps and work events, while servicing all blocks with the current generated trains. The Pricing Problem of this model is solved using a MIP formulation. This approach is similar to the approach that won the first prize. The two methods differ mainly in the train generation phase; probably, the multi-step procedure used by the winners is able to generate a better pool of trains, so that the final BTA problem contains better solutions. In our comparison we use the solution values as reported in Jin et al. (2013), the published version of their report, since in that work they improved their method.

6.4 A Mathematical Programming formulation

Let $\mathscr{T}$ denote the set of all feasible trains: each train $t \in \mathscr{T}$ is a sequence of crew segments, with a corresponding path $P(t)$ in G, and a sequence of work events $WE(t)$ associated to rail yards along the path. Function $c_{\mathscr{T}} : \mathscr{T} \to \mathbb{R}^+$ defines the cost of the train, which consists of the fixed train start cost, plus the train travel cost, plus the cost of the scheduled work events (see Section 6.2.2):

$$c_{\mathscr{T}}(t) = CL + CT \cdot L_{P(t)} + CW \cdot |WE(t)| \tag{6.1}$$

$\mathscr{T}_a \subseteq \mathscr{T}$ is the subset of trains which pass through arc $a \in A$ and K_t^a is the number of times train t passes through arc a. Starting from set $\mathscr{T}$ and graph G, one can obtain a multigraph $MG = (V, \mathscr{A})$, where $\mathscr{A}$ is obtained by replacing each arc $a \in A$ with $\sum_{t \in \mathscr{T}_a} K_t^a$ copies of it. If a train passes several times through arc a, each passage induces a different copy. The set of all the arcs induced on MG by train t is a path, denoted as A_t. Note that $\mathscr{A} = \cup_{t \in \mathscr{T}} A_t$.

Let $\mathscr{P}_b$ be the set of all feasible block-paths associated with block b on multigraph MG, $\mathscr{P}_b^t \subseteq \mathscr{P}_b$ the subset of paths in $\mathscr{P}_b$ that use train t, i. e. intersect A_t, and $\mathscr{P} = \cup_{b \in B} \mathscr{P}_b$ the whole set of block-paths. Note that set $\mathscr{P}_b$ represents all the ways in which block b can be feasibly routed in the rail network, starting from o_b, performing at most MS block swaps, and ending in d_b.

Function $c_{\mathscr{P}} : \mathscr{P} \to \mathbb{R}^+$ defines the cost of a block-path, consisting of the block travel cost plus the block swap cost (see Section 6.2.2):

$$c_{\mathscr{P}}(p) = CR \cdot N_{b_p} L_{P(p)} + \sum_{v \in BS(p)} CS_v \tag{6.2}$$

Finally, $\mathscr{P}^a$ is the set of block-paths which use arc a of MG, and, hence, the corresponding train. Tables 6.10 and 6.11, in the Appendix, summarize the notation introduced so far.

In the following, we provide a MIP formulation for the TDO problem. Binary variables λ_t and x_p state whether train $t \in \mathscr{T}$ and block-path $p \in \mathscr{P}$ are used or not in the solution. Variable Δ_s is the crew imbalance of each crew segment $s \in S$, variable Δ_v is the train imbalance of each rail yard $v \in V$.

$$\min \phi = \sum_{t \in \mathscr{T}} c_{\mathscr{T}}(t)\lambda_t + \sum_{p \in \mathscr{P}} c_{\mathscr{P}}(p) x_p + \sum_{s \in S} IS \cdot \Delta_s + \sum_{v \in V} IT \cdot \Delta_v \tag{6.3a}$$

$$\sum_{p \in \mathscr{P}_b} x_p \geq 1 \qquad b \in B \quad (y_b \geq 0) \tag{6.3b}$$

$$\sum_{p \in \mathscr{P}_b^t} x_p \leq \lambda_t \qquad t \in \mathscr{T}, b \in B \quad (\gamma_{tb} \geq 0) \tag{6.3c}$$

$$\sum_{b \in B} \sum_{p \in \mathscr{P}_b^t} x_p \leq MB \qquad t \in \mathscr{T} \quad (\eta_t \geq 0) \tag{6.3d}$$

$$\sum_{p \in \mathscr{P}^a} L_{b_p} x_p \leq ML_a \qquad t \in \mathscr{T}, a \in A_t \quad (h_{at} \geq 0) \tag{6.3e}$$

$$\sum_{p \in \mathscr{P}^a} W_{b_p} x_p \leq MW_a \qquad t \in \mathscr{T}, a \in A_t \quad (r_{at} \geq 0) \tag{6.3f}$$

$$\sum_{t \in \mathscr{T}_a} K_t^a \lambda_t \leq MN_a \qquad a \in A \quad (k_a \geq 0) \tag{6.3g}$$

$$\sum_{t \in \mathscr{T}} \Delta_s(t)\lambda_t \leq \Delta_s \qquad s \in S \quad (\delta_s^+ \geq 0) \tag{6.3h}$$

$$\sum_{t \in \mathscr{T}} -\Delta_s(t)\lambda_t \leq \Delta_s \qquad s \in S \quad (\delta_s^- \geq 0) \tag{6.3i}$$

$$\sum_{t \in \mathscr{T}} \Delta_v(t)\lambda_t \leq \Delta_v \qquad v \in V \quad (\delta_v^+ \geq 0) \tag{6.3j}$$

$$\sum_{t \in \mathscr{T}} -\Delta_v(t)\lambda_t \leq \Delta_v \qquad v \in V \quad (\delta_v^- \geq 0) \tag{6.3k}$$

$$x_p, \lambda_t \in \{0,1\} \qquad t \in \mathscr{T}, p \in \mathscr{P} \tag{6.3l}$$

Objective function (6.3a) sums up the cost of the selected trains, the cost of the selected block-paths and the crew and train imbalance costs. Constraint family (6.3b) imposes the existence of a block-path for each block: to allow the case in which block b is not served in the solution, $\mathscr{P}_b$ includes a dummy block-path of one arc with cost $MR \cdot N_b$ from o_b to d_b. Constraint family (6.3c) imposes to select train t if any of the selected block-paths uses it. Constraint family (6.3d) imposes the maximum number of blocks that a train can transport. Constraint families (6.3e,6.3f,6.3g) impose for each arc a the maximum length and weight of a train passing through a and the maximum number of trains that can pass through a. Constraint families (6.3h,6.3i) define the crew imbalance for each crew segment s, while constraint families (6.3j, ,6.3k) define the train imbalance for each yard v.

Formulation (6.3) can be rewritten setting all constraints in the $\geq$ form. Its LP-Relaxation, obtained replacing Constraints (6.3l) with nonnegativity constraints, admits the following dual formulation, whose variables are also reported in formulation (6.3) in round parenthesis.

$$\max \sum_{b\in B} y_b - \sum_{t\in\mathscr{T}} \left[MB\eta_t + \sum_{a\in A_t} (ML_a h_{at} + MW_a r_{at}) \right] - \sum_{a\in A} MN_a k_a \tag{6.4a}$$

$$y_b - \sum_{t\in\mathscr{T}_p} \left[\gamma_{tb} + \eta_t + \sum_{a\in A_t^p} (L_b h_{at} + W_b r_{at}) \right] \leq c_{\mathscr{P}}(p) \quad b\in B, p\in\mathscr{P}_b \tag{6.4b}$$

$$\sum_{b\in B} \gamma_{tb} - \sum_{a\in A_t} K_t^a k_a + \sum_{v\in V} \Delta_v(t)(\delta_v^- - \delta_v^+) + \sum_{s\in S} \Delta_s(t)(\delta_s^- - \delta_s^+) \leq c_{\mathscr{T}}(t) \quad t\in\mathscr{T} \tag{6.4c}$$

$$\delta_v^+ + \delta_v^- \leq IT \quad v\in V \tag{6.4d}$$

$$\delta_s^+ + \delta_s^- \leq IS \quad s\in S \tag{6.4e}$$

$$y_b \geq 0 \quad b\in B \tag{6.4f}$$

$$\eta_t \geq 0 \quad t\in\mathscr{T} \tag{6.4g}$$

$$\gamma_{tb} \geq 0 \quad b\in B,\ t\in\mathscr{T} \tag{6.4h}$$

$$r_{at}, h_{at} \geq 0 \quad t\in\mathscr{T}\ a\in A \tag{6.4i}$$

$$k_a \geq 0 \quad a\in A \tag{6.4j}$$

$$\delta_s^+, \delta_s^- \geq 0 \quad s\in S \tag{6.4k}$$

$$\delta_v^+, \delta_v^- \geq 0 \quad v\in V \tag{6.4l}$$

6.5 A Simultaneous Column and Row Generation method

This section provides an overall description of our Simultaneous Column and Row Generation heuristic *CRG-TDO*. The high level pseudocode is given in Figure 6.2, while the single steps of the algorithm are described in more detail in the following subsections. For the sake of briefness, the pseudocode denotes the given instance of the problem as *TDO*.

To solve the MIP model (6.3), we ideally relax the integrality conditions on the binary variables, generating the associated Master Problem (MP). Since the MP has an exponential number of variables and constraints, we apply a Simultaneous Col-

```
Algorithm CRG-TDO(TDO)
$(\overline{\mathscr{T}},\overline{\mathscr{P}}) := BuildDedicatedTrainsAndPaths(TDO)$;
$RMP := BuildReducedMasterProblem(\overline{\mathscr{T}},\overline{\mathscr{P}})$;
Repeat { Pricing loop: upper level }
    $(y,\eta,\gamma,r,h,k,\delta) := SolveLP(RMP)$;
    Repeat { Pricing loop: lower level }
        $\mathscr{P}' := GenerateBlockPaths(y,\eta,\gamma,r,h,\overline{\mathscr{T}})$;
        $\overline{\mathscr{P}} := \overline{\mathscr{P}} \cup \mathscr{P}'$;
        If $\mathscr{P}' \neq \emptyset$ then
            $RMP := BuildReducedMasterProblem(\overline{\mathscr{T}},\overline{\mathscr{P}})$;
            $(y,\eta,\gamma,r,h,k,\delta) := SolveLP(RMP)$;
        EndIf
    until $\mathscr{P}' = \emptyset$;
    $(\mathscr{T}',\mathscr{P}') := GenerateTrains(\gamma,k,\delta,\overline{\mathscr{T}})$;
    $\overline{\mathscr{T}} := \overline{\mathscr{T}} \cup \mathscr{T}'$;   $\overline{\mathscr{P}} := \overline{\mathscr{P}} \cup \mathscr{P}'$;
    if $\mathscr{T}' \neq \emptyset$ then $RMP := BuildReducedMasterProblem(\overline{\mathscr{T}},\overline{\mathscr{P}})$;
until $\mathscr{T}' = \emptyset$;
$(x,\lambda) := SolveMIP(RMP)$;
Return $(x,\lambda)$;
```

Fig. 6.2 Pseudocode of the Simultaneous Column and Row Generation heuristic *CRG-TDO*.

umn and Row Generation approach (see Section 2.4), which starts by considering a limited subset of variables and constraints, and iteratively adds new variables and constraints. In particular, procedure *BuildDedicatedTrainsAndPaths* generates for each block an *ad hoc* train, thus implicitly defining also the corresponding block-path. The route of the train is computed solving a minimum cost path problem on a suitable auxiliary graph. In this way, we obtain a small subset $\overline{\mathscr{T}}$ of feasible trains and a small subset $\overline{\mathscr{P}}$ of feasible block-paths. Procedure *BuildReducedMasterProblem* includes in the initial Reduced Master Problem (RMP) the train variables in $\overline{\mathscr{T}}$, the block-path variables in $\overline{\mathscr{P}}$ and all the imbalance variables Δ_s and Δ_v. Then we solve the RMP and use the optimal dual values to generate nonbasic variables with negative reduced costs. To generate new variables, we solve two Pricing Problems: the Block-path Pricing Problem (BPP) generates block-path variables, while the Train Pricing Problem (TPP) generates both train variables and block-path variables. We start by solving the BPP on the initial set of trains, and switch to the TPP when no new block-path variable with negative reduced cost can be identified. Since the two Pricing Problems are intertwined, the overall pricing phase has two nested feedback loops which we describe in the next sections. Notice that the introduction of a new train variable λ_t in our formulation implies the introduction of new constraints (6.3c)-(6.3f) and of the corresponding dual variables $\gamma_{tb}, \eta_t, h_{at}$ and r_{at}. At the end, when no more variables can be generated, we reintroduce the integrality constraints on the final RMP and solve it with procedure *SolveMIP*. The resulting solution is the output of the overall process. Note that, since the initialization procedure (see Section 6.5.1) generates a dedicated train and a corresponding block path for each block $b \in \mathscr{B}$, it is not possible that, at the end of the generation phase, the final RMP does not contain a feasible integral solution: the trains and the block-paths generated in the initialization phase constitute a (very poor) feasible integral solution.

6.5.1 *Initialization: procedure BuildDedicatedTrainsAndPaths*

We start by generating for each block b a train which carries b from its origin to its destination at minimum cost. To solve this problem we build for each block b an auxiliary node weighted graph $G'_b = (V'_b, A'_b)$. The node set V'_b contains a node for each crew segment s, plus a dummy origin node o and a dummy destination node d. If a crew segment admits more than one shortest path in each direction, we associate a different node in V'_b to each undirected shortest path. The arc set A'_b contains an arc from the origin o to each crew segment node s whose corresponding shortest path on G contains the origin of the block o_b, an arc from each crew segment node s whose shortest path $P(s)$ contains the destination of the block d_b to the destination d, and an arc between any two crew segment nodes s_1 and s_2 which have a common terminal node.

Each node is associated to a cost. Specifically a crew segment node s, whose corresponding shortest path (here, for simplicity, we consider a crew segment associated with a unique shortest path) $P(s)$ on graph G does not contain o_b or d_b, is associated to $c_s = CR \cdot N_b L_{P(s)}$, which is proportional to its total length. If o_b belongs to $P(s)$, we denote by $P(s)(o_b, v_2(s))$ the subpath of $P(s)$ between o_b and $v_2(s)$, and we set $c_s = CR \cdot N_b L_{P(s)(o_b, v_2(s))}$, plus an additional cost CW for a work event if $o_b \neq v_1(s)$. As well, if d_b belongs to $P(s)$, we set $c_s = CR \cdot N_b L_{P(s)(v_1(s), d_b)}$, plus an additional cost CW for a work event if $d_b \neq v_2(s)$. This is because the operations which occur at the origin or the destination rail yard of a train are not considered as work events. The dummy origin and destination nodes have zero cost. The minimum cost node-weighted path from o to d on G'_b simultaneously identifies a block-path for block b and the route of a dedicated train. Correspondingly, it provides an x_p variable and a λ_t variable.

Example

Figure 6.3 reports graph G'_b and a minimum cost path (in bold dashed lines) corresponding to the block-path (in thin dotted lines) reported in Figure 6.1(e) to carry block b from its origin o_b (marked in gray) to its destination d_b (marked in black). Notice that, since one of the crew segments in Figure 6.1(c) admits two alternative shortest paths in each direction, V'_b contains two nodes for that segment. Only one of them, however, is connected to the dummy node o, because only one of the shortest paths contains the origin yard o_b.

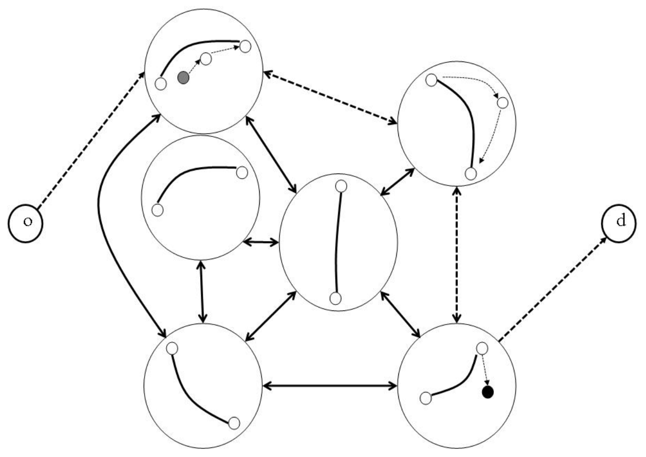


Fig. 6.3 Generation of auxiliary graph G'_b derived from rail network G to initialize the block-paths set $\overline{\mathscr{P}}$. We start from the rail network G depicted in Figure 6.1(e) and we generate G'_b by associating a vertex of G'_b to each shortest path associated with a crew segment, and an arc G'_b to each pair of consecutive shortest paths. The origin and destination vertices are directly connected with the vertices associated with the shortest paths that pass through them.

6.5.2 Generation of the block-paths: procedure GenerateBlockPaths

Given a pool of trains $\overline{\mathscr{T}} \subseteq \mathscr{T}$ and the optimal dual solution of the current RMP, we search for a negative reduced cost block-path p that uses only trains in $\overline{\mathscr{T}}$. The problem can be stated as that of finding on a suitable graph a minimum cost *constrained* path p with respect to a nonlinear objective function. It can be formulated as a MIP model as described in the following section. To solve it efficiently, we also consider a heuristic approach, described in Section 6.5.2.2.

6.5.2.1 A MIP formulation for the generation of block-paths

Given a block $b \in B$, the reduced cost $\overline{c}_b(p)$ of a generic block-path $p \in \mathscr{P}_b$, derives from constraint (6.4b) of the dual of the RMP:

$$\overline{c}_b(p) = c_{\mathscr{P}}(p) + \sum_{t \in \mathscr{T}_p} \sum_{a \in A_t^p} (L_b h_{at} + W_b r_{at}) + \sum_{t \in \mathscr{T}_p} (\eta_t + \gamma_{tb}) - y_b \tag{6.5}$$

If we find a block-path p such that $\overline{c}_b(p) < 0$, then the corresponding constraint is violated by the solution of the dual of the current RMP. Consequently, we can add variable x_p to the RMP. Hence, we look for a block-path of minimum reduced cost for each block b.

We start deriving from the multigraph MG associated with the current set of trains $\overline{\mathscr{T}}$ an auxiliary directed graph $\hat{G} = (\hat{V}, \hat{A})$. Each node v of MG is replaced in $\hat{V}$ by two node subsets U_v^- and U_v^+: U_v^+ contains a departure node v_t^+ for each train $t \in \overline{\mathscr{T}}$ that exits from v, i. e. for each arc in MG going out of v; U_v^- contains an arrival node v_t^- for each train $t \in \overline{\mathscr{T}}$ that enters v, i. e. for each arc in MG going into v. Finally, $\hat{V}$ includes a source node n_s and a sink node n_d. The arc set $\hat{A}$ includes an arc $a = (v_t^-, v_t^+)$ for each train t going both in and out of v. These arcs represent the permanence of a block on the train and have zero cost. $\hat{A}$ includes an arc $a = (v_{t_1}^-, v_{t_2}^+)$ for each pair of trains t_1 and t_2 such that t_1 goes into v, t_2 goes out of v and both trains can load/unload blocks in v (i. e. v is their origin or destination yard, or they both have a work event in v). These arcs represent block swaps. Since each arc (v, w) in MG is associated to a train t, $\hat{A}$ contains an arc $a = (v_t^+, w_t^-)$, which represents the movement of train t from yard v to yard w. The source node n_s is linked to all departure nodes v_t^+ and all arrival nodes v_t^- are linked to the sink node n_d.

Example

Figure 6.4 reports an example of graph $\hat{G}$. For the sake of clarity, we report only the arcs (n_s, v) and (v, n_d) associated to a given block b. The source node n_s is marked in gray, the sink node n_d in black. The bigger circles enclose the bipartite graphs $(U_v^-, U_v^+, \hat{A}_v)$ associated to each node v of the starting multigraph MG (not shown). The gray dashed arcs represent block swap arcs. The four available trains form set $\overline{\mathscr{T}} = \{t_1, t_2, t_3, t_4\}$, and the arcs associated to each train are represented by different line patterns. A path p from n_s to n_d is marked in fine dotted lines: it uses trains t_1, t_2, t_4 and finally again t_1. The three block swap arcs are depicted in bold.

The block-path p we are looking for corresponds to a path $P(p)$ from n_s to n_d in $\hat{G}$. The corresponding reduced cost $\overline{c}_b$ is the sum of four terms, as in (6.5): the cost $c_{\mathscr{P}}(p)$ as defined in (6.2), the term involving the dual variables h_{at} and r_{at}, the term involving the dual variables η_t and γ_{tb_p} for suitable indices a and t, and the constant term y_b. Given a block $b \in B$, the first two terms can be modeled by defining a suitable cost function $\hat{c}^b$ on $\hat{G}$ arcs: the arcs $a = (v_{t_1}^-, v_{t_2}^+)$ representing block swaps have a fixed cost $\hat{c}_a^b = CS_v$, the arcs $a = (v_t^+, w_t^-)$ representing train movements have cost $\hat{c}_a^b = CR \cdot N_b l_a + L_b h_{at} + W_b r_{at}$. The arcs (n_s, v) with $v \notin U_{o(b)}^+$ and the arcs (v, n_d) with $v \notin U_{d(b)}^-$ are forbidden by setting their cost to a value large enough to prevent their use in any optimal solution. All other arcs in $\hat{A}$ have zero cost. The cost $\sum_{a \in P(p)} \hat{c}_a^b$ of path $P(p)$ from n_s to n_d in $\hat{G}$ is then equal to

$$c_{\mathscr{P}}(p) + \sum_{t \in \mathscr{T}_p} \sum_{a \in A_t^p} (L_b h_{at} + W_b r_{at})$$

The third term, $\sum_{t \in \mathscr{T}_p} (\eta_t + \gamma_{tb})$, requires to pay a cost $\eta_t + \gamma_{tb}$ for each train t used by block b along path p. This cost should be paid exactly once, independently

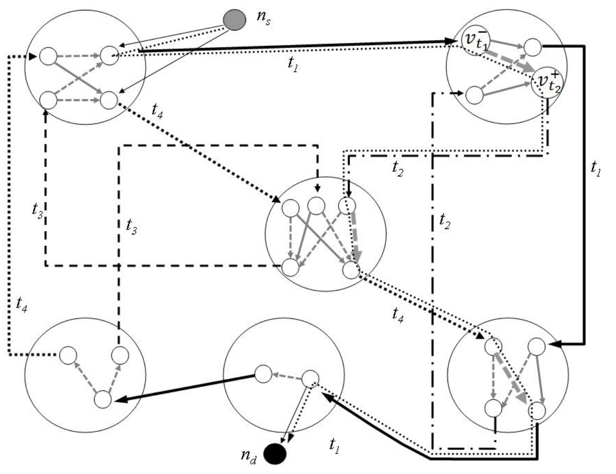


Fig. 6.4 An example of how a block-path can be represented in the auxiliary graph $\hat{G}$. The path followed by the considered block is drawn in fine dotted lines: it starts from the source node n_s, enters its origin rail yard, then takes trains t_1,t_2,t_3 and again t_1 to reach its destination rail yard, and finally enters the sink node n_d.

from the subset of arcs associated to train t and used by the path. To model this requirement, we introduce variables associated to the use of arcs and to the use of trains: $x_a = 1$ if the path uses arc $a \in \hat{A}$, $x_a = 0$ otherwise; $z_t = 1$ if the path uses train $t \in \overline{\mathscr{T}}$, $z_t = 0$ otherwise. The path with minimum reduced cost for block b is then the optimal solution of the following network flow formulation:

$$\min \ \bar{c}_b = \sum_{a \in A_b} \hat{c}_a^b x_a + \sum_{t \in \overline{\mathscr{T}}} (\gamma_{tb} + \eta_t) z_t - y_b \tag{6.6a}$$

$$\sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = 0 \qquad i \in \hat{V} \setminus \{n_s, n_d\} \tag{6.6b}$$

$$\sum_{a \in \delta^+(n_s)} x_a - \sum_{a \in \delta^-(n_s)} x_a = 1 \tag{6.6c}$$

$$\sum_{a \in \delta^+(n_d)} x_a - \sum_{a \in \delta^-(n_d)} x_a = -1 \tag{6.6d}$$

$$x_a \le z_t \qquad t \in \overline{\mathscr{T}} \quad a \in \hat{A}_t \tag{6.6e}$$

$$\sum_{a \in \hat{A}^{BS}} x_a \le MS \tag{6.6f}$$

$$x_a \in \{0,1\} \qquad a \in \hat{A} \tag{6.6g}$$

$$z_t \in \{0,1\} \qquad t \in \overline{\mathscr{T}} \tag{6.6h}$$

where $\hat{A}_t \subseteq \hat{A}$ contains all the arcs associated with train t, and $\hat{A}^{BS}$ contains all the block swap arcs, i. e. those such that $a = (v^-_{t_1}, v^+_{t_2})$ where $t_1, t_2 \in \overline{\mathscr{T}}$ and $t_1 \neq t_2$. Constraints (6.6b - 6.6d) require to send one unit of flow from n_s to n_d. Constraints (6.6e) activate a train if the path uses an arc associated with it and Constraint (6.6f) ensures that the block swap limit is satisfied.

The problem requires to find a minimum cost path $P(p)$ from n_s to n_d in G_b, including at most MS block swap arcs $a \in \hat{A}^{BS}$. The cost function is nonlinear, as it includes a fixed cost term $\gamma_{tb} + \eta_t$, which must be paid if $P(p)$ contains arcs in A_t, no matter how many of them.

Example continued

In Figure 6.4 block-path p is feasible only if $MS \geq 3$. Its fixed cost is $\sum_{i \in \{1,2,4\}}(\gamma_{t_i b} + \eta_{t_i})$, even if train t_1 is used twice.

6.5.2.2 Solution approach

Even if solving the MIP formulation is not a hard task for a commercial MIP solver, our computational experiments show that a relevant speed up can be obtained by solving the block-path generation problem with a heuristic approach. To take into account approximately the cost of the trains used, we modify the cost of some arcs in $\hat{G}$ as follows. The arcs (n_s, v^+_t) with $v^+_t \in U^+_{o(b)}$ have cost $\hat{c}^b_a = \eta_t + \gamma_{tb}$, instead of zero, to include the cost of the first train used. The arcs $a = (v^-_{t_1}, v^+_{t_2})$ representing block swaps have cost $\hat{c}^b_a = c_v + \eta_{t_2} + \gamma_{t_2 b}$ to include the cost of train t_2, which receives block b. Given the shortest path $P(p)$ in G_b from n_s to n_d with respect to the modified cost function, if p respects the maximum number of block swap arcs, we analyze its cost. If the block has been loaded on each train t exactly once along the whole path, the cost of $P(p)$ minus y_b is equal to $\overline{c}_b(p)$, and it is optimal for the Pricing Problem. If, on the contrary, b has been loaded on the same train more than once, the train has been paid more than once and the cost of p minus y_b overestimates $\overline{c}_b(p)$. By subtracting the extra cost paid, we obtain a feasible heuristic solution of the Pricing Problem, though not necessarily optimal. In both cases, if $\overline{c}_b(p) < 0$ the constraint of family (6.4b) corresponding to p is violated by the solution of the dual of the current RMP and we can add variable x_p to the RMP. If $P(p)$ is not feasible, or it is a heuristic solution with positive reduced cost, we move to the following block. If no other block is available, we move to the next step of the algorithm.

Implementation details

We solve the BPPs in order of increasing indices, adding to the RMP the block-paths generated when the reduced cost $\overline{c}_b$ is negative. The RMP is reoptimized only after considering all blocks.

For the sake of computational efficiency, when dealing with block b, we heuristically restrict graph $\hat{G}$ to a subset of arcs which are associated to promising trains, namely those already involved in carrying b in the current solution and the N_T trains whose route is closest to o_b and d_b.

6.5.3 Train generation: procedure GenerateTrains

Given a train $\tilde{t} \in \mathscr{T} \setminus \overline{\mathscr{T}}$ and a set of block-paths $\{\tilde{p}_1, \dots, \tilde{p}_k\}$ (at most one for each block) that use $\tilde{t}$ and possibly some other trains in $\overline{\mathscr{T}}$, to decide whether we need to insert the new train and the new block-paths in the RMP, we have to verify whether some of the associated dual constraints (6.4b, 6.4c) are violated. We mainly focus our attention on constraints (6.4c), i.e. we look for a new train $\tilde{t}$ such that these constraints are violated, while the possible violation of related constraints (6.4b) is considered a side effect. However, since train $\tilde{t}$ is not currently contained in the RMP (i. e. we are generating not only variable $\lambda_{\tilde{t}}$, but also the associated constraints), we do not know the values of the dual variables $\gamma_{\tilde{t}b}$, for all $b \in B$. These are the variables associated to constraints (6.4c). To face this problem, when building a new train $\tilde{t} \in \mathscr{T} \setminus \overline{\mathscr{T}}$ we impose a limiting condition.

Replacing condition. A newly generated train $\tilde{t}$ which replaces train t in the service of block b along part of its block-path, must replace it in all parts previously served by t.

By enforcing this condition we restrict the search of trains violating constraints (6.4c) only among those having routes composed by a concatenation of some parts of the routes of the already generated trains. In this way, we can estimate the unknown value of each variable $\gamma_{\tilde{t}b}$ using the current value of variable γ_{tb} where t is the train that the new train $\tilde{t}$ replaces in the service of block b. Intuitively, we can interpret the value of γ_{tb} as the prize that train t gains by carrying block b. Since train $\tilde{t}$ replaces train t in serving block b, the correct value of $\gamma_{\tilde{t}b}$ will not be very different from γ_{tb}. If $\tilde{t}$ replaces several trains, we estimate $\gamma_{\tilde{t}b}$ as the sum of γ_{tb} for each replaced train $t \in \overline{\mathscr{T}}$. Of course, the replacing condition restricts the set of possible trains and block-paths which can be generated.

The TPP consists in finding a new train $\tilde{t}$ and the corresponding new block-paths $\{\tilde{p}_1, \dots, \tilde{p}_k\}$ such that the new train constraint (6.4c) and possibly some of the new path constraints (6.4b) are violated. If we generate a new train $\tilde{t}$ and the corresponding new block-paths $\{\tilde{p}_1, \dots, \tilde{p}_k\}$ by respecting the replacing condition and the previous estimation of $\gamma_{\tilde{t}b}$, then the paths on G of all the blocks served by $\tilde{t}$ do not change and the new dual paths constraints (6.4b) are usually satisfied. There is an exception, when two or more *consecutive* subpaths previously assigned to two or more different trains are assigned to $\tilde{t}$. In this case, the cost $c_{\mathscr{P}}(\tilde{p})$ is actually lower than our estimate, because the new solution saves the cost of one or more block swaps.

If we succeed in identifying a new train $\tilde{t}$ whose corresponding constraint (6.4c) is violated, we can insert variable $\lambda_{\tilde{t}}$ and the corresponding new block-path variables into the RMP and reoptimize it. We solve the TPP using a Tabu Search heuristic (Glover and Laguna, 1997), limiting the search space to the trains and block-paths which satisfy the replacing condition, in order to use the above described estimation of $\gamma_{\tilde{t}b}$.

A two-level Tabu Search for the generation of trains

From Constraint (6.4c) of the dual of the RMP model, we derive the reduced cost $\overline{c}_{\tilde{t}}$ of variable $\lambda_{\tilde{t}}$ for all $\tilde{t} \in \mathscr{T} \setminus \overline{\mathscr{T}}$:

$$\overline{c}_{\tilde{t}} = c_{\tilde{t}} - \sum_{b \in B} \gamma_{\tilde{t}b} + \sum_{a \in A_{\tilde{t}}} K_{\tilde{t}}^{a} k_a - \sum_{v \in V} \Delta_v(\tilde{t})(\delta_v^- - \delta_v^+) - \sum_{s \in S} \Delta_s(\tilde{t})(\delta_s^- - \delta_s^+) \tag{6.7}$$

If we find a train $\tilde{t}$ with reduced cost $\overline{c}_{\tilde{t}} < 0$, then the corresponding constraint is violated by the solution of the dual of the current RMP and variable $\lambda_{\tilde{t}}$ can be added to the RMP. Hence, we look for a train of minimum reduced cost. To this purpose, we use a two-level Tabu Search heuristic. The first level represents trains as sequences of crew segments, and modifies the current train by adding or removing crew segments. For each train considered by the first level of the algorithm, the second level computes its optimal cost by heuristically identifying the best blocks to serve and the yards where to load and unload them.

Preprocessing

We first consider all the block-paths p such that $x_p > 0$ in the current solution of the RMP. For each block-path p, we identify the trains used, and for each used train $t \in \overline{\mathscr{T}}$ the (multi)set of subpaths ρ_{tp} which describes the portion of path $P(p)$ along which block b_p is served by train t. In the simplest case, ρ_{tp} is a simple path between two yards. However, more complex situations derive from the fact that the trains can follow multiple loops and that a block can be loaded and unloaded several times from the same train. For example, if the train makes a double loop along the same yards, ρ_{tp} contains two occurrences of the same subpath, corresponding to different passages of train t through the same arcs and distinguished with a numerical index. As well, when a train t loads and unloads the block b_p more than once, ρ_{tp} is a set of disjoint paths, which are portions of $P(p)$. The preprocessing phase generates all such objects ρ_{tp}.

First level Tabu Search

The first level Tabu Search designs the route of a train adding or removing crew segments at both ends of the current route. In the following, we denote by $start(t)$ and $end(t)$ the origin and the destination yard of train t.

The starting solution is given by one of the shortest paths associated to crew segment $s_{\text{in}} = 1$ (chosen at random if they are more than one).

The neighborhood is defined by *add moves*, which add to t a crew segment s such that one of its terminal nodes coincides with $start(t)$ or $end(t)$, and *drop moves*, which remove from t either the first or the last crew segment. The drop moves are not available if t contains a single crew segment. The cost of each solution is heuristically evaluated by running the second level Tabu Search, described in the following section.

The choice of the next solution in the neighborhood is controlled by a tabu mechanism, which has the purpose to avoid visiting previously obtained solutions by forbidding the use of the reversal of recently performed moves. Since the possible drop moves are at most two, they are always allowed, to avoid limiting excessively the search. As for the add moves, we apply a tabu mechanism to discourage the reintroduction of crews recently removed from the solution. The standard mechanism forbids the reintroduction of a crew until a certain number of iterations named *tenure* have elapsed after its removal. In the literature, the tenure is often updated over time, based on the quality of the last move performed or on the cardinality of the neighborhood. The aim of these mechanisms is to favor the exploration of more promising regions of the solution space and to drive the search away from less promising ones.

In the present case, the search is strongly limited by the fact that the solution is a path on a rather sparse graph. Since we visit only feasible solutions, which correspond to sequences of crew segments, and since in general rail networks are sparse, but with a strongly variable density over space, the size of the neighborhood defined above varies significantly from iteration to iteration, though it is always limited. A fixed tabu tenure is therefore unacceptable, but even applying a variable one we observed frequent cycling behaviors, due to the fact that the value of the tenure could not keep pace with the current situation. In some cases, all possible moves were tabu. In other cases, all tabu moves were unfeasible, and therefore unnecessarily tabu. The overall result was that the search moved alternatively between cycles and bad solutions. This suggested the introduction of a nonstandard scheme similar to the one used in Section 5.7.6 by the pricing heuristic for the HAP. For each crew segment $s \in S$, we save the last iteration i_s in which it was removed from the solution. At each iteration, we compute the number k of feasible add moves, and consider tabu the $\lfloor \alpha k \rfloor$ newer ones, i. e. those having the largest i_s values. Parameter $\alpha \in (0;1)$ is defined by the user, but rounding downwards guarantees that at least one add move is always nontabu. The algorithm selects the best nontabu move with respect to the objective function value, but it selects the best one, even if tabu, when it improves the best solution found so far.

When we need to diversify the explored space, i. e. when the best objective function value does not improve for D_{IT} iterations, we restart the search by replacing the current initial crew segment s_{in} with the following one and building a new solution from it.

The algorithm terminates when a fixed number of iterations M_{IT} have been executed. All the trains with a negative reduced cost are added to the RMP together with the corresponding block-paths generated in the second level Tabu Search to estimate their cost. We reoptimize the RMP only when the current generation phase stops.

Second level Tabu Search

This procedure receives from the first level Tabu Search (see the previous section) a feasible train, represented as a sequence of crew segments $\tilde{t} = (s_1, \ldots, s_k)$, and tries to maximize the collected prizes γ_{tb} gained by replacing an old train $t \in \overline{\mathscr{T}}$ in serving block b. To achieve this result satisfying the replacing condition, we select, among the sets ρ_{tp} with $p \in \overline{\mathscr{P}}, t \in \overline{\mathscr{T}}$, generated during the preprocessing phase, the ones which are fully covered by $P(\tilde{t})$. Each selected set defines a portion of a block-path $p \in \overline{P}$ on which an already generated train $t \in \overline{\mathscr{T}}$ can be replaced by $\tilde{t}$ satisfying the replacing condition. We call the selected sets *feasible* and we denote the set containing them as $\rho(\tilde{t})$.

Then, we initialize the solution by selecting a feasible set ρ_{tp} with the minimum difference between the possible work event cost and the value of γ_{tb_p}, i. e. the minimum contribution to the objective function value. The neighborhood is defined by *add moves*, which add a feasible subpath set ρ_{tp} to the ones currently replaced by $\tilde{t}$, and *drop moves*, which remove a subpath set ρ_{tp} from the ones currently replaced by $\tilde{t}$. In order to avoid visiting previously obtained solutions, we introduce a simple tabu mechanism with a tabu list of fixed length $TT = \lceil \psi |\rho(\tilde{t})| \rceil$. Parameter $\psi \in (0,1)$ is defined by the user. The attribute which characterizes a move is the name of the involved feasible set $\rho_{tp} \in \rho(\tilde{t})$, and a move is tabu if its attribute is contained in the corresponding list. In other words, an add move is tabu if the added feasible set has been removed during the last TT moves, a drop move is tabu if the removed feasible set has been added during the last TT moves. At each iteration, the algorithm selects the best nontabu move with respect to the objective function value. The algorithm terminates when a threshold number of iterations has been reached: this threshold is set equal to 3 times the size of $|\rho(\tilde{t})|$.

Example

Figure 6.5 reports an example on multigraph MG (for the sake of simplicity, we avoid representing in detail the interior of the rail yards). A candidate new train $\tilde{t} \in \mathscr{T} \setminus \overline{\mathscr{T}}$ is depicted in dashed thin lines, as a sequence of three crew segments, $\{(B,D),(D,F),(F,G)\}$. Five other trains, $t_1, t_2, \ldots, t_5 \in \overline{\mathscr{T}}$, are currently used to

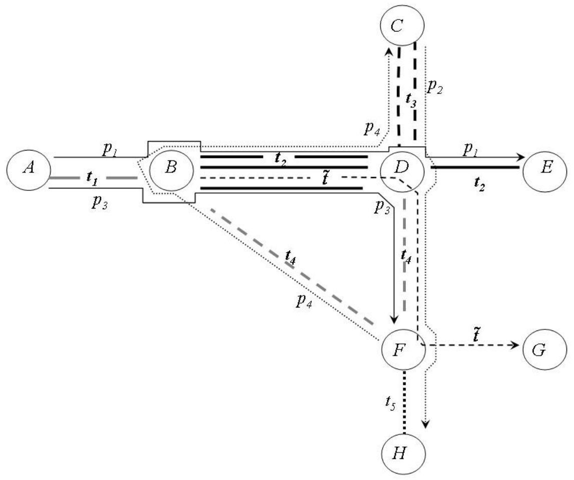


Fig. 6.5 Example of multigraph MG. A new train $\tilde{t}$ is depicted in dashed thin line. The current solution contains other five trains t_1, t_2, t_3, t_4 and t_5, which define four different paths p_1, p_2, p_3 and p_4.

move four blocks along paths p_1, p_2, p_3 and p_4. The second level of the Tabu Search procedure starts identifying the following sets: $\rho_{t_1 p_1} = \{(A,B)\}$, $\rho_{t_2 p_1} = \{(B,D,E)\}$, $\rho_{t_3 p_2} = \{(C,D)\}$, $\rho_{t_4 p_2} = \{(D,F)\}$, $\rho_{t_5 p_2} = \{(F,H)\}$, $\rho_{t_1 p_3} = \{(A,B)\}$, $\rho_{t_2 p_3} = \{(B,D)\}$, $\rho_{t_4 p_3} = \{(D,F)\}$, $\rho_{t_4 p_4} = \{(F,B)\}$, $\rho_{t_2 p_4} = \{(B,D)\}$ and $\rho_{t_3 p_4} = \{(D,C)\}$. Then, it selects the feasible sets $\rho_{t_4 p_2}$, $\rho_{t_2 p_3}$, $\rho_{t_4 p_3}$ and $\rho_{t_2 p_4}$, on which the search will take place. In particular, the replacing condition forbids to assign block b_{p_1} to train $\tilde{t}$ on subpath (B,D), because train t_2 serves b_{p_1} also along crew segment (D,E), which is not covered by the route of $\tilde{t}$.

Neighborhood exploration speed up

Since the first level Tabu Search adopts a global-best strategy in the neighborhood exploration and the evaluation of each single move is based on a nested second level Tabu Search, on the first level we need to avoid the expensive evaluation of moves that cannot increase the prize collected by the current train. To obtain this speed up, we compute a simple lower bound on the minimum reduced cost that the given train can assume. This is obtained subtracting from the reduced cost of the empty train t the maximum collectible prize, i. e. the sum over all blocks b of the largest prize of a feasible subpath associated with b. If this lower bound is not smaller than the minimum reduced cost found so far, the considered train cannot improve the best known result, and therefore can be discarded.

6.6 Computational results and final remarks

We developed all the procedures previously described using the C++ programming language and built with gcc 4.6 setting the optimization flag -O3. We used ILOG CPLEX 12.5 with the ILOG CONCERT libraries to solve the generated LP/ILP problems. We performed the computational experiments on a computer equipped with an Intel Xeon Processor E5-1620 Quad Core 3.60GHz and 16 GB of RAM.

6.6.1 Benchmark data sets

CRG-TDO has been tested on the two data sets provided by the organizers of the *2011 INFORMS RAS Problem Solving Competition*[3], who built them on purpose to represent problems of a realistic size and structure. Table 6.1 summarizes the main features of these data sets.

Data set	Yards	Blocks	Tracks	Segments
1	94	239	134	154
2	221	369	294	154

Table 6.1 Summary of the benchmark data sets characteristics.

Table 6.2 summarizes the operational costs and constraints which characterize both data sets.

Parameter	Value	Description
MB	8	Maximum blocks per train
MS	3	Maximum block swaps per block
ME	4	Maximum work events per train
CL	400	Train start cost
CT	10	Train travel cost per unit of distance
CW	350	Cost per work event
CR	0.75	Car travel cost per unit of distances
CS_v	$[30, 100]$	Cost per block swap (range)
IS	600	Crew imbalance penalty
IT	1000	Train imbalance penalty
MR	5000	Cost per missed railcar

Table 6.2 Summary of the operational costs and constraints of the tested data sets.

[3] See `http://www.informs.org/Community/RAS/Problem-Solving-Competition/2011-RAS-Problem-Solving-Competition`

6.6.2 Parameter tuning

Our algorithm is divided in two main phases: the iterative Simultaneous Column and Row Generation phase and the resolution of the final MIP problem, obtained reintroducing the integrality constraint on the last RMP generated. Table 6.3 reports the parameter values used for the first phase, while Table 6.4 reports those used for the second one. The latter are CPLEX parameters (see IBM User's Manual for CPLEX 12.5), modified with respect to their default values, in order to increase the time spent looking for better feasible solutions and decrease the time spent proving the optimality of the solutions. This is because the method is overall heuristic, and there is no strong advantage in solving its second phase exactly.

Parameter	Value	Description
N_T	50	Number of trains considered during the block-path generation
M_{IT}	1500	Number of iterations of the first level Tabu Search
D_{IT}	30	Number of nonimproving iterations before restart in the first level Tabu Search
α	0.4	Ratio of tabu moves to feasible moves in the first level Tabu Search
ψ	1/3	Multiplicative factor defining the tabu tenure in the second level Tabu Search

Table 6.3 Parameter setting for the Simultaneous Column and Row Generation phase.

Parameter	Value	Description
`CPX_PARAM_POLISHAFTER`	800	Time before solution polishing
`CPX_PARAM_HEURFREQ`	5	Frequency of the standard heuristic
`CPX_PARAM_MIPEMPHASIS`	4	Emphasize finding hidden feasible solutions
`CPX_PARAM_RINSHEUR`	10	Frequency of the RINS heuristic

Table 6.4 Parameter setting for the final MIP solution phase.

6.6.3 Computational results

When applied to real world data sets, such as the ones provided for the competition, both phases take a significant computational time. The effective balance between the time spent in each of them is crucial to determine the effectiveness of the overall algorithm. In fact, if the first phase generates in early iterations trains and block-paths forming a satisfactory solution, the best strategy is to immediately apply the second phase, so that the MIP solver can quickly extract such a solution. Performing several iterations not only takes a longer time, but also produces a larger final MIP problem, which is in itself slower to solve. On the other hand, performing several iterations also generates many more trains and block-paths, which could combine into solutions of better quality. Thus, the number of train generation iterations, TGL, is an influential parameter. To investigate the compromise between a larger pool of columns and a more easily manageable one, we launched different runs of the algorithm, progressively increasing TGL from 1 to 7. In all runs, we limited to 1 200 seconds the CPU time for the MIP model.

TGL	ϕ_{RMP}	T_{RMP}	ϕ^*_{MIP}	LB_{MIP}	Δ_{MIP}
1	2 221 916.4	1	2 165 522.6	2 165 523.7	0.00%
2	2 011 146.5	38	2 069 066.3	2 062 163.3	0.33%
3	1 971 325.5	180	2 049 600.4	2 033 074.2	0.81%
4	1 960 760 3	328	2 043 786.4	2 023 245.7	1.02%
5	1 952 871.8	531	2 034 870.0	2 012 509.6	1.11%
6	1 946 872.7	846	2 032 725.6	2 008 770.9	1.19%
7	1 946 302.1	1001	**2 032 515.6**	2 007 961.5	1.22%

Table 6.5 Results found at different iterations on data set 1.

TGL	ϕ_{RMP}	T_{RMP}	ϕ^*_{MIP}	LB_{MIP}	Δ_{MIP}
1	3 353 281.9	6	3 326 932.4	3 326 933.4	0.00%
2	3 152 746.5	142	3 217 136.6	3 207 162.1	0.31%
3	3 102 344.4	541	3 205 195.9	3 181 282.6	0.75%
4	3 081 430.7	867	**3 188 500.1**	3 160 019.9	0.89%
5	3 073 959 1	1609	3 191 017.4	3 152 668.7	1.22%
6	3 070 872.2	1924	3 200 528.1	3 151 721.1	1.55%
7	3 070 859.3	1997	3 199 701.3	3 150 620.6	1.56%

Table 6.6 Results found at different iterations on data set 2.

Tables 6.5 and 6.6 report the following information on the results obtained during this experiment, respectively on the first and the second data set:

- TGL: number of train generation iterations performed.
- ϕ_{RMP}: optimal value of the RMP at the end of the Simultaneous Column and Row Generation phase.
- T_{RMP}: total computational time in seconds required by the Simultaneous Column and Row Generation phase.
- ϕ^*_{MIP}: best solution found solving the final MIP problem.
- LB_{MIP}: best lower bound found solving the final MIP problem.
- Δ_{MIP}: percentage gap between the best solution and the lower bound.

Our best result for data set 1, $\phi^*_{MIP} = 2032515.6$, is obtained at iteration $TGL = 7$, after 2201 seconds, i. e. 1 001 seconds spent generating columns and rows and 1 200 seconds spent solving the final MIP problem. Our best result for data set 2, $\phi^*_{MIP} = 3188500.1$, is obtained at iteration $TGL = 4$, after 2067 seconds, i. e. 867 seconds spent generating columns and rows and 1 200 seconds spent solving the final MIP problem. Incidentally, it is worth mentioning that these results strongly improve upon those awarded by the *RAS-2011* competition (see Colombo et al. 2011):

the cost, in fact, is 8.12% lower for data set 1 and 9.90% lower for data set 2, while the computational time reduced from 10 hours for data set 1 and 4 hours for data set 2 to $30-40$ minutes in both cases.

The values reported in subsequent rows of column ϕ^*_{MIP} describe the progressive improvement of the solutions found as more train generation iterations are performed. On the second data set, these values stop improving after the fourth iteration. This is due to the increasing size of the MIP model, which restrains CPLEX from finding better solutions in the allotted time, even if at least the same solution is still available. Indeed, only at the first iteration the MIP model can be solved exactly (see the gaps reported in the last column). The increasing size of the final MIP problem is reflected by the increasing residual gap left, as well as by the larger gap of the second data set with respect to the first one. An *ad hoc* algorithm to solve the final MIP problem could perhaps improve this gap and reduce this component of the overall computational time. Notice that neither ϕ_{RMP} nor LB_{MIP} are guaranteed lower bounds on the optimum of the whole problem, because the train generation phase is heuristic, and it is terminated when negative cost columns can still be found. However, they are lower bounds on the best value which can be found at each iteration.

Tables 6.7 and 6.8 compare the four prize-winning methods of the *RAS-2011* competition, referring to each of the two benchmark data sets. Columns M_1, M_2, M_3 and M_4 report, respectively, the best solutions obtained by the methods winning the first, second, third prize and the honorable mention. Method M_1 is described in Lozano et al. (2011), method M_2 is our algorithm in the enhanced version here presented, methods M_3 and M_4 are described in Wang et al. (2011) and Jin et al. (2013). The rows of these tables provide the different cost components for the solutions achieved, the total cost and the total execution time in seconds.

Cost	M_1	M_2	M_3	M_4
Train start	417 031.000	35 600.000	344 381.000	29 200.000
Train travel		331 910.000		322 821.000
Train imbalance		14 000.000	30 078.000	34 000.000
Crew imbalance		21 000.000	20 186.000	34 200.000
Work event	55 650.000	40 950.000	53 898.000	53 200.000
Car travel	1 565 730.000	1 587 878.625	156 7680.755	1 590 491.100
Block swap	5 060.000	3 100.000	2 422.000	5 840.000
Missed cars	0.000	0.000	0.000	0.000
Total	2 043 471.000	2 032 725.625	**2 018 643.755**	2 069 752.100
CPU Time	**45**	2 201	80 000-90 000	342

Table 6.7 Comparison of the four prize-winning algorithms in the *RAS-2011* competition on data set 1.

Methods M_1 and M_3 appear to be nondominated in both data sets. In particular, the method that won the contest (M_1) finds good solutions in a matter of minutes,

Cost	M_1	M_2	M_3	M_4
Train start	485 277.000	52 400.000	440 652.200	39 600.000
Train travel		417 926.000		446 203.000
Train imbalance		10 000.000	15 760.090	26 000.000
Crew imbalance		9 600.000	13 553.680	25 200.000
Work event	82 950.000	63 000.000	78 170.060	90 300.000
Car travel	2 191 692.000	2 211 394.125	2 180 566.000	2 186 604.450
Block swap	8 080.000	4 180.000	3 467.220	6 270.000
Missed cars	420 000.000	420 000.000	420 000.000	420 000.000
Total	3 187 999.000	3 188 500.125	**3 152 019.000**	3 240 177.000
CPU Time	**225**	2 067	75 000-80 000	2700

Table 6.8 Comparison of the four prize-winning algorithms in the *RAS-2011* competition on data set 2.

whereas M_3 finds the best known solutions in about one day of computation. When focusing on data set 1, also the method here proposed (M_2) is nondominated, with an intermediate performance with respect to M_1 and M_3 and a computational of about $30-40$ minutes. Method M_4 is dominated by M_1, but also our method can find a better solution in the same time ($\phi^*_{MIP} = 2052957.225$, with $TGL = 2$ and 300 seconds of MIP computation). As for data set 2, method M_1 dominates our method and both dominate M_4.

A detailed comparison, however, is complex since all the computational times reported refer to different machines, though not dramatically different. Therefore, only very large differences can be considered significant. Moreover, Wang et al. (2011) do not explicitly report the CPU time required by M_3 to find the best solution; we had to extrapolate an approximate range from the plots provided in their report. Since the values in that range are anyway fairly large, the relative error introduced is probably limited, but the result is not precise.

Disaggregate analysis of the cost components

Tables 6.7 and 6.8 provide row by row the values of the cost components for methods M_2 and M_4; for the other two methods, not all components are available: some components are aggregated with different levels of detail. An interesting remark is that on both data sets our method found solutions with a smaller work event cost and a larger car travel cost. These two aspects are related: reducing the number of work events implies that blocks are loaded and unloaded less frequently, and therefore travel along longer paths. It should be observed that in the given benchmarks the car travel cost is the dominating cost component. Therefore, it is likely that the introduction of additional work events in the train generation phase could allow to improve the solution in these specific data sets. Since we prefer to preserve the generality of our method, we decided not to introduce modifications dictated by remarks on the specific structure of the given data sets. In our opinion, a rail company manager who would like to focus on specific cost components could easily do that by manipulating the cost coefficients in the input data. This would be more complex when using methods intrinsically biased towards the minimization of some components.

A lower bound on the optimum

All methods proposed for the TDO problem are heuristic in principle. None of them, therefore, can provide the optimal solution. It is however, possible to introduce a simple lower bounding procedure, which provides an estimate from below of the value of the optimum, so as to evaluate the quality of the available heuristic solutions.

The idea is to consider separately the components of the cost function listed in Section 6.2.2, to compute a lower bound for each of them, and to sum the bounds. Table 6.9 provides the components of this bound, and its overall value, for the two data sets considered. First of all, the missed railcars cost can be underestimated determining whether there are blocks whose origin or destination does not belong to any of the shortest paths of the available crew segments. If this occurs, those blocks cannot be serviced in any way, their rail cars will be certainly missed, and their cost will be included in the overall objective function. This is unlikely to occur in practice, but it is the case of 5 blocks in data set 2. Otherwise, the lower bound is set to zero, as in data set 1. Then, we consider the shortest path from the origin to the destination of each block, except for those which are necessarily missed. Assuming that they could be serviced by dedicated trains, we obtain a lower bound on the total length of the corresponding block-path, and consequently on the block travel cost. Then, we take into account that each train can carry at most MB blocks along its route. The total number of nonmissed blocks divided by MB implies a lower bound on the number of trains required to carry all the nonmissed blocks, and consequently on the total train start cost. We could actually increase the number of missed blocks, thus decreasing this bound, but that would correspondingly increase the bound on the cost of missed cars by a larger amount. As well, the total travel distance for the nonmissed blocks divided by MB provides a lower bound on the total travel distance for the trains, which implies a bound on the train travel cost. We set to zero the bounds for work events, block swaps, train imbalance and crew imbalance.

Based on these lower bounds, we can make a rough estimate of the quality of the solutions obtained. In particular, on data set 1 our solution is 35.44% worse than the lower bound, and the best known solution is 34.50% worse. On data set 2, our solution is 28.10% worse and the best known one is 26.63% worse. These gaps show that the heuristics succeed in providing at least reasonable solutions to a very complex problem. The largest fraction of these gaps is probably due to the simple nature of the lower bound. However, given the large money investments required by the dispatch of freight trains, even a small reduction of the gap between the best known solution and the optimal one could correspond to a significant saving.

Cost component	Data set 1	Data set 2
Missed cars	0	420 000
Block travel	1 399 840	1 919 660
Train start	12 000	32 000
Train travel	89 014	117 443
Work events	0	0
Block swaps	0	0
Train imbalance	0	0
Crew imbalance	0	0
Total	1 500 853	2 489 103

Table 6.9 Lower bounds on the optimal value of the single cost components and of the overall cost on the two data sets of *RAS-2011* competition.

6.7 Conclusions

In this chapter we described a Simultaneous Column and Row Generation (see Section 2.4) heuristic to solve a *Train Design Optimization* problem. This problem arises in the freight rail industry and it was the subject of the *2011 INFORMS RAS Problem Solving Competition*. The method described in this chapter is an improved version of the one we developed to win the second prize in this competition. This method consists of two sequential phases:

1. In the first phase, a Simultaneous Column and Row Generation approach identifies good routes for trains and for blocks.
2. In the second phase, we solve using a standard MIP solver the final restricted Master Problem enforcing the integrality constraint on all variables.

The first phase is based on the resolution of two different Pricing Problems, one to generate routes for the blocks and one to generate routes for the trains. We tackled the first problem by modeling it as a constrained shortest path and by solving it with an ad-hoc heuristic. For what concerns the second Pricing Problem, we developed a two-level Tabu Search heuristic where in the first level we determine the structure of the new train route and in the second level we load it with blocks.

We compare the solutions obtained by our method to those obtained by the three methods developed by finalist participants. The comparison is based on the two data sets provided by the competition organizers and the costs of the solutions found by the competing methods were provided by the other participants.

From this comparison, our method proves able to obtain very good solution in a reasonable time (less than one hour). On data set 1, only one of the tested methods is able to find a better solution, but requires a huge amount of CPU time. On data set 2, a slightly better solution can be quickly found by one of the competing methods, and a better one in a very long time by another method. We think that further improvements of our heuristic can be obtained by developing a superior method to filter the generated routes before the second phase (for example, defining effective nontrivial dominance criteria) or by introducing an *ad hoc* algorithm to solve the final MIP model.

6.8 References

R. K. Ahuja, C. B. Cunha, and G. Şahin. Network Models in Railroad Planning and Scheduling. In J. Cole Smith, editor, *Tutorials in Operations Research: Emerging Theory, Methods, and Applications*, chapter 3, pages 54–101. INFORMS, 2005.

R. K. Ahuja, K. C. Jha, and J. Liu. Solving real-life railroad blocking problems. *Interfaces*, 37:404–419, 2007.

M. Albers. *Freight Railway Crew Scheduling: Models, Methods, and Applications*. Logos Verlag, Berlin, 2009.

A. A. Assad. Models for rail transportation. *Transportation Research Part A*, 14(3): 205–220, 1980.

F. Colombo, R. Cordone, and M. Trubian. A network-oriented formulation and decomposition approach to solve the 2011 RAS problem. `http://www.informs.org/Community/RAS/Problem-Solving-Competition/2011-RAS-Problem-Solving-Competition`, 2011.

J. F. Cordeau, P. Toth, and D. Vigo. A survey of optimization models for train routing and scheduling. *Transportation Science*, 32(4):380–404, November 1998.

P. Corry and E. Kozan. Optimised loading patterns for intermodal trains. *OR Spectrum*, 30(4):721–750, 2008.

T. G. Crainic and J. M. Rousseau. Multicommodity, multimode freight transportation: A general modeling and algorithmic framework for the service network design problem. *Transportation Research Part B*, 20(3):225–242, 1986.

F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

J. W. Goossens, S. van Hoesel, and L. Kroon. A branch-and-cut approach for solving railway line-planning problems. *Transportation Science*, 38(3):373–393, 2004.

A. E. Haghani. Rail freight transportation: a review of recent optimization models for train routing and empty car distribution. *Journal of Advanced Transportation*, 21:147–172, 1987.

A. E. Haghani. Formulation and solution of a combined train routing and makeup, and empty car distribution model. *Transportation Research Part B*, 23(6):433–452, 1989.

IBM User's Manual for CPLEX 12.5. International Business Machines Corporation.

K. C. Jha, R. K. Ahuja, and G. Sahin. New Approaches for Solving the Block-to-Train Assignment Problem. *Networks*, 51:48–62, 2008.

J. G. Jin and J. Zhao. Train design optimization. `http://www.informs.org/Community/RAS/Problem-Solving-Competition/2011-RAS-Problem-Solving-Competition`, 2011.

Jian Gang Jin, Jun Zhao, and Der-Horng Lee. A column generation based approach for the train network design optimization problem. *Transportation Research Part E: Logistics and Transportation Review*, 50(0):1 – 17, 2013.

M. H. Keaton. Designing optimal railroad operating plans: Lagrangian relaxation and heuristic approaches. *Transportation Research Part B*, 23(6):415–431, 1989.

M. H. Keaton. Designing optimal railroad operating plans: A dual adjustment method for implementing Lagrangian relaxation. *Transportation Science*, 26: 262–279, 1992.

L. Lozano, J. E. Gonzalez, and A. L. Medaglia. A column-row generation heuristic for the train design optimization problem. `http://www.informs.org/Community/RAS/Problem-Solving-Competition/2011-RAS-Problem-Solving-Competition`, 2011.

I. L. Wang, H. Y. Lee, and Y. T. Liang. Train design optimization. `http://www.informs.org/Community/RAS/Problem-Solving-Competition/2011-RAS-Problem-Solving-Competition`, 2011.

Appendix

Tables 6.10 and 6.11 summarize the notation adopted in the chapter to describe, respectively, the data and the solutions of the problem.

Name	Description
V	set of the rail yards
A	set of the rail tracks
l_a	length of rail track a
S	set of the crew segments
$v_1(s)$	initial rail yard of crew segment s
$v_2(s)$	final rail yard of crew segment s
B	set of the blocks
o_b	origin rail yard of block b
d_b	destination rail yard of block b
N_b	number of railcars in block b
L_b	length of block b
W_b	weight of block b
MB	maximum number of blocks in a train
MS	maximum number of block swap on a block-path
ME	maximum number of work event on a train route
ML_a	maximum length of a train on rail track a
MW_a	maximum weight of a train on rail track a
MN_a	maximum number of trains admitted on rail track a
CL	train start cost
CT	train cost per unit of distance
CR	railcar cost per unit of distance
CW	cost per work event
CS_v	cost per block swap in rail yard v
IS	crew imbalance penalty
IT	train imbalance penalty
MR	cost per missed railcar

Table 6.10 List of the notations used throughout the chapter to describe the data.

Name	Description
$\mathscr{T}$	set of the feasible trains
$\mathscr{T}_a$	set of trains passing through rail track a
$\mathscr{P}$	set of the feasible block-paths
$\mathscr{P}_b$	set of the feasible block-paths associated with block b
$\mathscr{P}_b^t$	set of paths in $\mathscr{P}_b$ that use train t
$\mathscr{P}^a$	set of the block-paths that use rail track a
$P(\cdot)$	sequence of rail tracks defining its argument
$P(t)$	sequence of rail tracks crossed by train t
$P(p)$	sequence of rail tracks crossed by block path p
$P(s)$	sequence of rail tracks crossed by the shortest path associated with crew segment s
$L_{P(\cdot)}$	total length of rail tracks associated with $P(\cdot)$
K_t^a	number of times train t passes through rail track a
$WE(t)$	set of work events associated with train t
$\Delta_v(t)$	equal to 1 if the route of train t starts in v, -1 if it ends in v
$\Delta_s(t)$	difference between the number of times in which train t uses crew segment s from $v_1(s)$ to $v_2(s)$ and from $v_2(s)$ to $v_1(s)$
$BS(p)$	set of nodes in which block-path p performs a block swap
b_p	block associated with block-path p
$c_{\mathscr{T}}(t)$	cost of train t: $c_{\mathscr{T}}(t) = CL + CT \cdot L_{P(t)} + CW\lvert WE(t)\rvert$
$c_{\mathscr{P}}(p)$	cost of block-path p: $c_{\mathscr{P}}(p) = N_{b_p} CR \cdot L_{P(p)} + \sum_{v \in BS(p)} c_v$

Table 6.11 List of the notations used throughout the chapter to describe the solutions.