Université Paris 13 - Université Sorbonne Paris Cité
Laboratoire d'Informatique de Paris Nord - LIPN
Équipe: Algorithmes et Optimisation Combinatoire - AOC

Exact and Heuristic Methods for Multi-Activity Tour Scheduling Problems

Thèse de doctorat
présentée par

Stefania Pan

à l'École Doctorale Galilée

pour obtenir le grade de
Docteur en Informatique

Soutenue à Villetaneuse le 13 de décembre 2018
devant le jury composé de:

François Clautiaux	Université de Bordeaux	Rapporteur
Bernard Gendron	Université de Montréal	Rapporteur
Sophie Demassey	Mines ParisTech	Examinatrice
Safia Kedad-Sidhoum	CNAM	Examinatrice
Louis-Martin Rousseau	École Polytechnique de Montréal	Examinateur
Roberto Wolfler Calvo	Université Paris 13	Directeur de thèse
Lucas Létocart	Université Paris 13	Co-directeur de thèse
Mahuna Akplogan	Horizontal Software	Co-encadrant de thèse
Nora Touati	Horizontal Software	Co-encadrante de thèse

Abstract

Personnel scheduling problems encompass a large collection of optimization challenges that several organizations need to face. One of the first classifications proposed divides these problems into: days-off scheduling, shift scheduling, and tour scheduling. The first concerns the determination of working and rest days; the second defines the shifts to be assigned; the third integrates days-off into shift scheduling.

The current thesis addresses the tour scheduling problem with a multi-activity context arising in restaurant business, which was provided by the company Horizontal Software. As such, the main goal is to define the working days and the shifts, along with the specification of the activities in each time period. This problem is also characterized by a high degree of flexibility, mainly due to the introduction of a long pause (interruption), and heterogeneity, determined by conditions from employees (skills, availabilities, contract regulations, and pre-assignments).

The problem is tackled by a Branch-and-Price algorithm, which is based on column generation. A dual ascent heuristic is implemented to speed up its convergence, and a constraint and dynamic programming based method is proposed for generating schedules. To address the large-scale instances, various heuristics are presented, based on column generation, large neighborhood search and tabu search. To assess the performance of the proposed method, computational experiments are conducted on instances from the literature and on real-world instances that were provided by Horizontal Software.

Keywords: Multi-activity tour scheduling problem; Dual ascent; Regular grammar; Resource constrained shortest path; Branch-and-Price; Heuristics.

Résumé

La planification du personnel constitue une vaste classe de problèmes d'optimisation rencontrés dans différentes organisations. L'une des premières classifications proposées divise ces problèmes en *days-off scheduling*, *shift scheduling* et *tour scheduling.* Le premier concerne la détermination des jours de travail et de repos, le deuxième définit les horaires de travail, et le troisième intègre les deux.

Cette thèse découle de la volonté de la société Horizontal Software de résoudre l'un des problèmes difficiles rencontrés dans la restauration. Le problème relève de la catégorie du *tour scheduling* dans un contexte multi-activités, où l'objectif est la définition des jours et des horaires de travail, ainsi que la spécification des activités dans chaque période. Ce problème présente un degré élevé de flexibilité et d'hétérogénéité. La première est causée par l'introduction d'une longue pause (coupure), tandis que la deuxième est due aux compétences, aux disponibilités, aux contrats et aux pré-affectations des employés.

Le problème est résolu par une méthode du type *Branch-and-Price*, dont l'élément clé est la génération de colonnes. Une heuristique *dual ascent* a été proposée pour accélérer sa convergence, et une méthode basée sur la programmation dynamique et par contraintes a été proposée pour générer des plannings individuels. Afin de traiter les instances de grande taille, différentes heuristiques ont été développées, basées sur la génération de colonnes, la recherche à voisinage large et tabou. Des tests expérimentaux sur des instances de la littérature et réelles ont été effectués afin dévaluer les performances des méthodes proposées.

Mots clés: Problème de planification du personnel avec plusieurs activités; Dual ascent; Grammaire régulière; Plus court chemin avec contraintes; Branch-and-Price; Heuristiques.

Contents

List of Figures

List of Tables

Abbreviations

B&B	**B**ranch-and-**B**ound algorithm
B&P	**B**ranch-and-**P**rice algorithm
CG	**C**olumn **G**eneration
DA	**D**ual **A**scent
DAG	**D**irected **A**cyclic **G**raph
DFA	**D**eterministic **F**inite **A**utomaton
IP	**I**nteger **P**rogramming
LNS	**L**arge **N**eighborhood **S**earch
LP	**L**inear **P**rogramming
MIP	**M**ixed **I**nteger **P**rogramming
MILP	**M**ixed **I**nteger **L**inear **P**rogramming
RMP	**R**educed **M**aster **P**roblem
SC	**S**et **C**overing
SP	**S**et **P**acking
SPT	**S**et **P**ar**T**itioning
TS	**T**abu **S**earch

Introduction

The management of human capital is a crucial factor affecting the productivity and the service quality of an organization. It plays an important role in different processes, such as the recruitment, talent management (training, interviews and careers management), time management and workforce management. Personnel scheduling problems arise in the latter process. Thompson (2003) points out two primary reasons for caring about personnel scheduling. The first reason is related to the time spent defining a planning by hand that leaves the manager less time for actually managing the employees and interacting with the customers. The second reason concerns profitability and effectiveness, since an efficient planning potentially leads to an overall cost reduction and performance improvement.

Over time, the evolution of management practices and the scientific advancements brought researchers to focus on more and more complex problems with more and more diverse applications. Nowadays, many real-world personnel scheduling problems concern companies where schedules differ from the standard eight hours per day and five days per week. The problem becomes challenging, due to several factors such as multiple contracts, work regulations, multiple activities, employee skills and availabilities, employee preferences, etc. As consequence, it becomes difficult finding a solution satisfying all the constraints of the resulting complex large-scale problem. It is even more difficult finding an optimal solution minimizing a given criterion, such as the total cost of the planning or under and over coverage of the demand. In this case, it is almost impossible to define a planning by hand, and the use of specific mathematical models and algorithms becomes essential.

Personnel scheduling concerns various areas of applications, where we find transportation systems (airline, railway, bus), call centers, health care systems (hospital, private clinic), emergency services (ambulance, police, firefighter), civic utilities (post office), main events (sports competitions, casino), financial services (bank), tourism (museum), hotels, restaurants, retail, among others. Each company has specific requirements and rules, even tough it may be grouped together with other companies for sectoral affinity. This diversity, along with the development and the launch of new businesses, increases the interest addressed to personnel scheduling.

This work has been funded by BPIfrance in the frame of the French cooperative project ADAMme, Projet Investissement Avenir (FSN: AAP CISN2).

Context and motivations

The human capital management is a complex and transverse task that managers need to do, and it requires specific actions depending on the nature of each company. When a performing tool is provided, managing employees becomes easier, guaranteeing their well-being and the quality of the services supplied, without jeopardizing the financial health of the company. These tools need not only to provide a planning satisfying all constraints (work regulations, preferences, equity, etc.), but also a planning that is optimal or near-optimal according to some given criterion (costs, demand coverage, etc.). In addition, they need to allow the visualization and the analysis of the resulting planning, in order to let the manager benefit from his know-how and make the appropriate corrections.

In general, this type of tools is marketed by specialized companies, such as Horizontal Software. This company offers a software dedicated to the management of the human capital in various areas, such as tourism, leisure activities, retail, health care, legal, restaurants, etc. The modularity of the software allows to cover the different processes of human capital management, from the recruitment and the talent management, to the time and the workforce management. The module concerned with personnel scheduling is named E-Optim, and it appears as in Figure 1. The employees and the corresponding planning are shown in the central part of the figure, while some statistics are displayed on the bottom and on the right allowing a rapid evaluation of the quality of the planning. In addition, different features allow the manager to modify the solution if necessary.

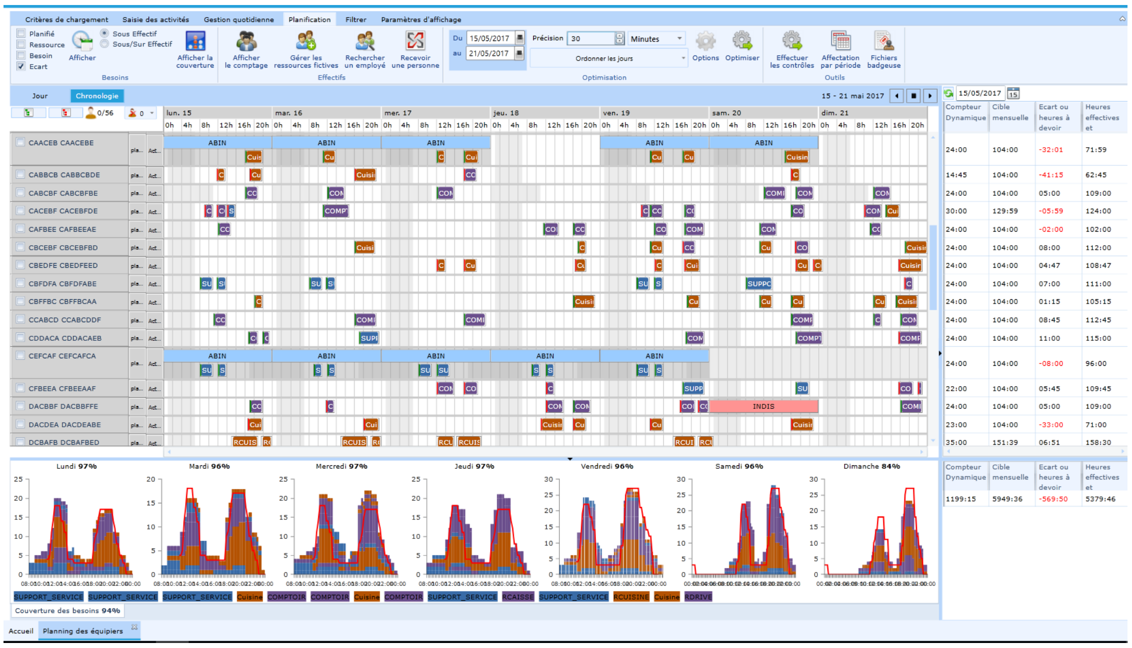


FIGURE 1: E-Optim tool.

E-Optim has the goal of defining a planning that satisfies the different constraints characterizing the problem, covers the demand, meets the needs and the preferences of the employees. It also allows to automatically generate the demand based on business indicators, such as the company's turnover or the historical data on the attendance rate, and to carry out planning simulations based on different criteria.

This research project is originated by the will of Horizontal Software in solving efficiently the scheduling problems arising in the context of the restaurant business, which are characterized by various complex work regulations and a high degree of flexibility in defining the planning. The project relies on a partnership between the company Horizontal Software and the LIPN (Laboratoire d'Informatique Paris Nord) of the Université Paris 13.

Research objectives

Two research lines are developed along this thesis. An academic line aims at mathematically characterize the problem that concerns Horizontal Software and its placement in the literature of personnel scheduling. Furthermore, the goal is to propose efficient solving methods able to deal with the specificities of the problem addressed.

The second line of research is industrial in partnership with Horizontal Software. It explores the workforce management process, and it is devoted to the improvement of the E-Optim tool through the algorithms developed in this thesis in order to speed up and optimize the personnel scheduling in the context of restaurant business. All these algorithms have been implemented in the E-Optim tool.

One of the operational constraints of Horizontal Software imposes to avoid the use of a commercial solver, since they cannot be embedded in E-Optim tool. However, in this thesis we do not resort to open-source LP solvers when needed for developing an algorithm, and we employ a state-of-the-art solver. The main reason relies on the fact that we wanted to evaluate the performance of the developed methods, without being affected by the performance of an open-source and not commercial solver, which is usually less efficient. In addition, we wanted to see "how far we could go" with the real instances provided by Horizontal Software with the use of a state-of-the-art solver.

Thesis outline

The thesis is essentially structured in three parts:

- Part I is dedicated to the definition of the multi-activity tour scheduling problem, i.e., one of the problems that concerns the company Horizontal Software. After a first chapter describing the problem addressed and its placement in the literature, different formulations are proposed.
- Part II presents an exact approach for the multi-activity tour scheduling problem. After a focus on two elements which are at the basis of the algorithms, the problem is tackled by means of Branch-and-Price (B&P).
- Part III proposes four heuristic approaches for the multi-activity tour scheduling problem, with the goal of finding good solutions to the large-scale instances.

Finally, conclusions of this work and future perspectives are drawn. In the following we briefly introduce the chapters of this thesis.

Chapter 1: Problem Description

The multi-activity tour scheduling problem addressed in this thesis is introduced, giving the basic definitions and describing the constraints and the objective considered. A review of related problems, models and solving methods reported in the literature is presented. Furthermore, the instances used to evaluate the performance of the algorithms that will be proposed in this thesis are described.

Chapter 2: Mathematical Model

A compact formulation in the form of Mixed Integer Linear Program (MILP) is introduced, along with a Dantzig-Wolfe decomposition which is well-suited for the multi-activity tour scheduling problem. Finally, a reminder of Column Generation (CG) is presented.

Chapter 3: Dual Ascent Heuristic

Since CG is one of the key elements in this thesis, a special attention is paid to the resolution of the reduced master problem, and a Dual Ascent (DA) heuristic for solving it is presented in this chapter. The generality of this procedure is highlighted, and computational results are presented in different contexts.

Chapter 4: Pricing Problem

As complement of the previous chapter, we focus on the resolution of the pricing problem, where constraint and dynamic programming techniques are combined together. To deal with large-scale instances, various strategies are proposed for heuristically solving the pricing problem. Extensive computational sessions are conducted to assess the performance of the different strategies.
This research has been done together with Louis-Martin Rousseau, during a three-months internship at École Polytechnique de Montréal.

Chapter 5: Branch-and-Price

The algorithms presented in Chapter 3 and Chapter 4 are embedded in a CG framework, and different strategies for speeding up its convergence are described. CG is then combined with a branching rule into a B&P algorithm for the exact resolution of the multi-activity tour scheduling problem. Computational experiments are performed on the testbed defined in Chapter 1.

Chapter 6: Heuristic Methods

In order to find feasible solutions to large-scale instances of the multi-activity tour scheduling problem, four heuristic algorithms based on Column Generation (CG), Large Neighborhood Search (LNS), and Tabu Search (TS) are proposed. Computational experiments are conducted on the testbed defined in Chapter 1 in order to evaluate the performance of each heuristic.

Appendix

Appendix A is devoted to the presentation of a work which has been studied in parallel to the aforementioned main research topic. This work concerns the energy management in a future decentralized system, where the interactions between a power generation company and various micro-grids are taken into consideration.
This research has been done together with Wim van Ackooij, Jérôme De Boeck, Michael Poss and Boris Detienne.

Part I

Personnel scheduling

Introduction to Part I

Personnel scheduling problems arise in many different areas, such as airline, railway and bus companies, call centers, health care systems, restaurant businnes, retail stores and many others. Each area has particular requirements that make the concerning personnel scheduling problem specific. As a consequence, the literature on this topic is extensive. To support this, we mention the review of Ernst et al. (2004a), where more than 700 papers are analyzed and classified. The application we are interested in this thesis concerns the fast food restaurant chains, which is one of the most challenging scheduling problem that the company Horizontal Software has to face. This problem is characterized by a high degree of flexibility in defining personnel planning in order to respond to the particular requirements arising in this area. In the following we describe the problem addressed in this thesis. In particular, Part I is organized as follows:

- Chapter 1 gives the basic definitions needed to describe the problem and it introduces the constraints considered. Furthermore, the literature review concerning personnel scheduling is presented, together with the description of the instances used for the computational experiments all along the thesis.
- Chapter 2 introduces a compact MILP model and a Dantzig-Wolfe decomposition of the considered problem. In addition, the main principles of columns generation are recalled.

Chapter 1

Problem Description

The purpose of this chapter is to describe and formalize the personnel scheduling problem addressed in this thesis. One of the first classifications for personnel scheduling problems was proposed by Baker (1976). According to Baker, three main groups can be distinguished: days-off scheduling, shift scheduling, and tour scheduling.

- *Days-off scheduling* concerns the determination of working and rest days all over the planning horizon.
- *Shift scheduling* is also called time-of-day scheduling and it concerns the definition of the working periods during the working days of each employee.
- *Tour scheduling* integrates days-off and shifts scheduling by specifying both the working days and the working periods of each employee.

In general, the groups presented above consider only one work activity. However, we may need to specify what employees do in each period since more than one work activity has to be scheduled. In this case, the shift and the tour scheduling problems become respectively the *multi-activity shift scheduling* and *multi-activity tour scheduling* problems. The problem addressed in this thesis falls into this last category, where the goal is not only the definition of the working days and the working periods but also the specification of work activities in each time period. The presented classification is not the only one that can be found in the literature. Ernst et al. (2004b) and Van den Bergh et al. (2013) categorize personnel scheduling problems according the problem addressed, the solving method and the application area. Further details are given in Section 1.3.1.

We formalize the problem by introducing the basic notions in Section 1.1, we describe the characteristics and the constraints of the problem in Section 1.2, and we give an insight into the existing literature in Section 1.3. Section 1.4 details the instances that are used for the computational experiments all along the thesis, and conclusions are drawn in Section 1.5.

1.1 Definitions

This section describes the main elements that appear in the multi-activity tour scheduling problem addressed.

- *Time horizon*: the time span that needs to be scheduled. It is usually fixed to one week, as in our case.
- *Slots*: the time periods of equal length that partition the planning horizon. Usually, the length is fixed to 15 minutes, 30 minutes or 1 hour.
- *Task*: a sequence of consecutive slots where the same activity is performed without being interrupted by any other activity or pause.
- *Timeslot*: a sequence of activities performed consecutively. It may contain only one activity, which is performed for the entire duration of the timeslot, or multiple different activities, which are assigned successively without any pause on the transitions between them.
- *Break*: a short pause can be assigned during the working day and it is called break. It has a duration usually between 30 minutes and 1 hour, and if it falls during lunchtime, the employee may have lunch during the break. For this reason, no notion of lunch break will be introduced, in contrast to what is usually done in literature.
- *Interruption*: in addition to breaks, a long pause can be assigned during the working day, and it is named interruption. Its duration usually goes from 2 hours to 5 hours. This type of pause arises in fast food restaurant chains, where rush periods during lunchtime and dinner are ordinary and a higher flexibility in scheduling is required. Further details are given in Section 1.2.1.
- *Shift*: a combination of timeslots, breaks, and interruptions forms a daily shift. More in details, a daily shift is made of one, two or three timeslots, where two consecutive timeslots are separated by either a break or an interruption.
- *Daily shift*: a shift that covers only *daily slots* is called daily shift. For instance, daily slots may consider all time periods from the opening time to 9pm.
- *Night shift*: a shift that covers at least one of the *night slots* is called night shift. For instance, night slots may consider time periods in the late evening, from 9pm to the closing time. The distinction between daily and night shift is necessary due to the fact that some constraints depend on which shift the employee has worked.
- *Rest between shifts*: between two consecutive shifts a period where the employee does not work needs to be assigned. This time period is called rest.
- *Day-off*: a day where the employee does not work is called day-off. Since our problem falls into the category of tour scheduling, days-off are not pre-assigned and need to be considered while defining the schedule of each employee.
- *Schedule*: a sequence of shifts and days-off covering the whole planning horizon is called schedule.
- *Planning*: a set of schedules, one for each employee, is called planning.

1.2 Problem statement

This section gives the description of the problem. Many characteristics may arise in personnel scheduling, such as overlapping shifts that span multiple days, different degrees of schedule flexibility and employees heterogeneity. Our problem focuses on the two last characteristics, which are discussed in Section 1.2.1 and in Section 1.2.2. Then, the constraints considered in the problem are detailed in Section 1.2.3 and how to evaluate the quality of a planning is explained in Section 1.2.4.

1.2.1 Schedule flexibility

One of the most important features of the multi-activity tour scheduling problem under study relies on the possibility of defining schedules that considerably differ one from the other, and are able to adapt to the needs arising in fast food restaurant chains. Indeed, fast food restaurant chains are characterized by a demand having two rush periods during lunchtime and dinner, that follow habits. Therefore, service needs to be guaranteed in these two periods. In order to deal with the peaks of demand, employees may be assigned to timeslots that cover distant periods of the day. For instance, the employee works from 9am to 1pm, and from 5pm to 8pm. To the best of our knowledge, the multi-activity tour scheduling problems in the literature do not allow this type of shifts, since the breaks separating two timeslots are usually short breaks or lunch breaks of about one hour.

In order to guarantee such flexibility in defining schedules, the assignment of a long pause is necessary. This pause is called interruption and it is subject to the following constraints:[1]

- only one interruption can be assigned each day;
- an interruption has a minimum duration of 2 hours and it cannot exceed 5 hours;
- unlike normal breaks, interruptions have a cost that reflects the fact that this type of pause may be undesirable for the employee.

The drawback of such flexibility relies on the increase in the number of feasible shifts and, therefore, the number of feasible schedules.

1.2.2 Employees heterogeneity

Besides the schedule flexibility presented above, another feature of the multi-activity tour scheduling problem addressed consists in the heterogeneity of the employees. That is, each employee has his own specificities that distinguish him from the others. As a consequence, shifts and schedules need to be designed for each employee. In this case, the problem is called *heterogeneous multi-activity tour scheduling*. Four main factors are responsible for the heterogeneity of the problem.

[1] https://www.legifrance.gouv.fr/affichIDCC.do;jsessionid=3223FE83B3F0267DEC1439A0D64D5517.tpdjo16v_1?idConvention=KALICONT000005635596&cidTexte=KALITEXT000028900902

- *Skills*: The variety of the activities considered causes the requirement of particular skills to the employees that perform them. Furthermore, employees have different competencies and, therefore, they can perform only a specific subset of activities.
- *Availability*: Each employee is associated with a set of slots where he is unavailable. For instance, the employee has taken a day-off on Friday, or he has agreed with the manager on not working all Thursday afternoon, or he has taken the morning off due to a medical exam. As results, he is not available all days and all time. Many different factors affect the availabilities, which change from one week to another.
- *Contract type*: Work regulations of each employee are defined by his contract. Two types of contract are typically proposed: part-time and full-time. In general, the work regulations are the same, but the bounds imposed are different. For instance, with a part-time contract, an employee can work 24 hours per week, against the 35 weekly working hours of the full-time contract.
- *Pre-assignments*: Employees may be pre-assigned to activities in specific time periods. As a consequence, the schedules designed need to take into account these pre-assignments, which differ from employee to employee.

When all the employees are equivalent concerning skills, availability, contract type, and pre-assignments, the problem is called *homogeneous multi-activity tour scheduling.*

1.2.3 Constraints

This section presents the constraints of the multi-activity tour scheduling problem addressed. They are mainly grouped into five categories: workload, legal, activities cardinality, succession and distribution constraints.

Workload. They concern the demand and they are expressed as the number of employees required for each activity in each slot. Demand modeling may be part of the scheduling process (Ernst et al. (2004b)). However, in this thesis, we assume that the demand has already been investigated before scheduling the employees, and it is an input data of the problem. In addition, both under and over coverage are allowed, meaning that a planning is feasible even though it does not satisfy exactly the demand.

Legal. They concern the definition of rules for designing timeslots, shifts, and schedules. They mainly determine the feasibility of each of them, and they depend on the work regulations defined by the contract.

- **Legal for timeslot.**

  (L1) *Activities duration.* When multiple activities are considered, each of them has its own rules related to the duration. This means that each activity can be assigned a number of consecutive time periods that falls between a minimum and a maximum number of slots.

(L2) *Consecutive working time.* The length of a timeslot must fall between a minimum and a maximum number of slots, imposing that the number of consecutive working time is bounded both from below and from above. These bounds aim at avoiding unproductive and tiring timeslots by assigning working periods respectively too short and too long.

- **Legal for shift.**

(L3) *Break and interruption duration and number.* The pause length must fall between a minimum and a maximum number of slots. The pause length is associated with its type: an interruption is longer than a break. Similarly, these constraints restrict the number of breaks and interruptions that can be assigned into a shift.

(L4) *Daily working time.* Employees must work a number of time periods that falls between a minimum and a maximum number of slots. The working time does not consider the pauses duration and it corresponds exactly to the total length of the timeslots assigned during the day.

(L5) *Amplitude of the working day.* Shifts length defines the amplitude of the working day, and it is limited both from below and from above. This constraint avoids that the shift spans a period of time too long, mostly when interruptions are assigned between two timeslots in the shift.

(L6) *Starting and finishing time.* Shifts start and finish respectively after and within specific time periods. For instance, in a fast food restaurant chain, shifts have to start after the opening time and they need to be completed before the closing time. However, there might be the need of imposing specific starting time slots in order to simplify the problem or due to particular work regulations (Restrepo et al. (2016)).

- **Legal for schedule.**

(L7) *Weekly working time.* Beside the daily working time restriction, employees must work a number of weekly time periods that falls between a minimum and a maximum number of slots.

(L8) *Consecutive working days.* This constraint specifies the number of days that an employee works without having any day-off. This number is limited both from below and from above. Even though both bounds can be imposed, the restriction on the maximum number of consecutive working days is essential, since it forbids that an employee works too many days in a row.

(L9) *Number of working days.* The total number of working days is restricted by a minimum and a maximum bound. These constraints can also be seen as the number of days-off that needs to be assigned in a schedule.

(L10) *Rest between consecutive shifts.* This constraint specifies the number of time periods between two consecutive shifts. Both lower and upper bounds can be imposed on this rest period, and they depend on the type of the first shift. Indeed, the rest imposed after a night shift is generally longer than the one after a daily shift.

Activities cardinality. They concern the number of activities that an employee can perform in each slot, and this number is fixed to 1. As a consequence, in each slot an employee either does not work or he is assigned to one single activity.

Succession. They concern the assignment of two different activities in consecutive time periods, and they specifically forbid that one activity is assigned just after the other. This type of constraints commonly arises in the nurse rostering problem, where a nurse cannot perform two specific shifts successively. However, the forbidding of a particular succession of two activities can be imposed also in multi-activity tour scheduling.

Distribution. They concern the load of activities during a specific period of time. It differs from workload since the load of activities does not come from external demand, but it is derived from other factors or needs. For instance, an employee has to perform some training hours during the week. The courses are flexible and they can be attended in different periods.

1.2.4 Planning quality evaluation

A planning is considered feasible if each employee is assigned to a schedule that satisfies all legal, cardinality, succession and distribution constraints presented in the previous section. More in details, activities, breaks and interruptions are grouped together into a complete schedule that respects the constraints. In addition, the schedule is assigned to an adequate employee, meeting his competencies, availability and pre-assignments.

Beside feasibility, we can evaluate the quality of a planning with the purpose of defining the criteria that state if a solution is optimal or not. Different factors may contribute to the quality of a planning, such as the total number of employees scheduled, the equity in scheduling hard activities or the employees preferences. In this thesis, we focus on three factors: the penalty for under and over coverage of the demand, the total cost of the planning and the penalty for transitions between activities.

- *Under and over coverage.* An activity is under (over) covered in a slot when the number of employees performing that activity in that slot is lower (greater) than the required number defined by the demand. The costs of both under and over coverage are considered in the objective function to evaluate how the planning satisfies workload constraints and covers the demand. Even though the demand is given as input data, requiring the exact coverage may cause infeasibility when solving the problem. One possible solution is to allow only over coverage and to penalize it. However, the demand fluctuation may result in a planning with a high number of periods over covered and a high cost. In addition, there are days where many employees are unavailable and the demand is overestimated compared to the real workforce that can be on duty. In this case, the coverage of the demand may be impossible. For all these reasons, both under and over coverage are allowed.

- *Planning costs.* Each activity is associated with a cost when performed on a specific slot and by a specific employee. These costs allow the definition of each schedule cost and, therefore, the complete planning cost. For instance, an activity harder than the others to be performed due to an intense physical or mental requirement has a higher cost; an activity performed by an employee who has the skills needed but is not the most qualified, has a higher cost than if it is performed by another employee.
- *Transitions between activities.* Even when no constraint is imposed on the number of times that an employee can change activity during a timeslot, a schedule with a low number of transitions between activities might be preferable. For this reason, a penalty is added to the cost of a schedule when transitions occur.

1.3 State of the art

Personnel scheduling has been studied for decades and the literature concerning this problem is extensive. However, researchers are still very interested in personnel scheduling. This is explained especially by a socio-economic context in constant evolution, that leads to the need of renewing and adapting planning techniques. Section 1.3.1 presents the different classifications that can be found in the literature of personnel scheduling problems. Then, Section 1.3.2 and Section 1.3.3 focus on the review of shift scheduling and tour scheduling respectively.

1.3.1 Classifications of personnel scheduling

To deal with the extensive literature of personnel scheduling, different classifications have been proposed according to problem type, solution approach, and application area.

Baker (1976) proposes one of the first classifications where three main classes of problems are identified: day-off scheduling, shift scheduling and tour scheduling. As previously mentioned, day-off scheduling concerns the determination of rest and working days in a seven-a-day working context, over a planning horizon of usually multiple weeks. Shift scheduling aims at defining the working periods during the working days of the employees. Finally, tour scheduling combines the previous two problems by specifying both working days and working periods of the employees. A detailed classification focusing on the tour scheduling problem has been proposed by Alfares (2004). The author reviews more than 100 papers into ten classes, according to the solution method: manual solution, integer programming, implicit modeling, decomposition, goal programming, working set generation, LP-based solution, construction and improvement, metaheuristics and other methods.

Ernst et al. (2004b) and Van den Bergh et al. (2013) present comprehensive survey of general personnel scheduling problems. Ernst et al. (2004b) classify more than 700 published papers according to problem type, application area, and solution approach. In particular, they present the scheduling process as a combination of six basic modules: demand modeling, days-off scheduling, shift scheduling, line of work construction, task assignment and staff assignment. Recently, Van den Bergh et al. (2013) review more than 300 articles published between 2004 and 2012. Papers are categorized according to four main fields: 1) personnel characteristics (contract type, skills, individual or crew), decision delineation and shifts definition; 2) different constraints (hard or soft, coverage, time related, fairness and balance), performance measures and flexibility; 3) solution method and uncertainty incorporation; 4) application area and applicability of research.

In the following we lean on the classification proposed by Baker (1976) and we present the literature review of the two classes closer to the problem addressed in this thesis.

1.3.2 Shift scheduling

The origin of shift scheduling problems goes back to Edie (1954) in the context of scheduling toll booth operators. Even though the author introduces the problem, no mathematical formulation is given. The first integer programming formulation was introduced just after by Dantzig (1954), where the proposed set covering model defines one variable for each possible shift that can be assigned to the employees. The objective is to select the set of shifts that covers the demand and minimize the overall cost. In general, the shift scheduling problem consists in designing the shifts to be assigned to employees, by choosing starting time, length, breaks placement, breaks duration, and all other flexible aspects that may be included in the problem. However, when the degree of flexibility introduced in the model increases, the number of possible shifts becomes very large and their complete enumeration is not possible. This limit has encouraged researchers to study other formulations and to model shifts implicitly rather than explicitly.

The first implicit formulation was presented by Bechtold and Jacobs (1990) for shift scheduling with break placement. Instead of defining explicit variables for each possible combination of shift and break as for the explicit set covering formulation, the authors use a shift variable for each shift type, and a break variable for each period during which a break can start. The matching between shifts and breaks is guaranteed by the so-called forward and backward constraints, which ensure that implicit and explicit formulations are equivalent (Bechtold and Jacobs (1996)). The implicit model allows reducing the number of variables, while it requires a higher number of constraints when compared with the explicit model. Computational results show that only using the implicit formulation they were able to solve all the considered instances.

Implicit modeling was also used to handle flexibility on shift start time and shift lengths. Thompson (1995) combines previous works on implicit shift representation and on implicit break placement. Shifts having the same cost per working period, the same break duration, identical restrictions on minimum and maximum shift length, and identical restrictions on the minimum and maximum pre- and post-break work stretch durations belong to the same shift type. Furthermore, each shift is supposed to receive at most a single break. The implicit model uses variables concerning shift starting time, shift finishing time, and break for each shift type and possible starting time. A set of constraints imposes restrictions on the minimum and maximum length for each shift type. Another set of constraints ensures the minimum and maximum pre- and post-break work stretch durations for each shift type.

The model is solved by means of B&B, and it is followed by a first-in-first-out procedure that matches shift starts to shift finishes, and shift starts to breaks to construct explicit shifts and assign breaks to them. Thompson (1995) points out that the implicit nature of the model imposes some restrictions on the shift cost structure. Indeed, the cost of shifts is assumed to be a linear function of the number of working periods, since the author states that an explicit model is more appropriate if the cost of a shift depends on starting time, duration, break placement, etc.

Unlike the work of Bechtold and Jacobs (1990) where employees are allowed only a single lunch break, Aykin (1996) considers a more general shift scheduling problem with multiple breaks. In particular, they consider a 30 minutes lunch break and two 15 minutes short breaks to be assigned in disjoint break windows. The implicit model presented uses shift variables for each shift type, and break variables for each shift and each possible starting time within its break windows. Computational results show that break flexibility lowers the number of employees needed. Based on the implicit model, Aykin (1998) proposes a Branch-and-Cut algorithm able to optimally solve the largest instances addressed until then.

Rekik et al. (2010) extend previous works on implicit models by considering a shift scheduling problem with a high degree of flexibility. In addition to the classical flexibility sources (i.e., shift starting time, shift length, multiple breaks and break windows), the authors incorporate multiple and fractionable breaks, and break windows with restrictions on pre- and post-break work duration. The authors propose two implicit formulations that use forward and backward constraints. Furthermore, they show that a reformulation of these constraints that uses slack variables reduces the density of the constraint matrix. Computational results show that the high degree of flexibility considered in the problem reduces the workforce size when compared with other approaches. For instance, they compare the performance of using work duration restrictions for fixing the break windows against the ideal break start time approach proposed by Aykin (1996).

Besides implicit formulations, other methods have been proposed to solve the shift scheduling problem. They are based on the explicit set covering formulation proposed by Dantzig (1954), and they overcome the problems in solving it arising from the large size by means of CG.

A B&P algorithm is proposed by Mehrotra et al. (2000) for shift scheduling with multiple breaks, meal break and break windows. They propose a generalized set covering formulation that includes upper and lower bounds both on the number of employees needed in each time period and on the number of employees who can be on a break in any time period. This last restriction can be imposed by security regulations, company policy or limited space where taking a break. They propose and implement three branching rules based on work periods, break periods or two specific periods. Computational experiments compare the B&P with the implicit formulations proposed by Aykin (1996). The results show that B&P is not only able to solve more instances to optimality, but it is in general significantly faster.

Heuristic approaches based on local search have also been proposed in the literature for solving the shift scheduling. Indeed, Musliu et al. (2004) investigate local search proposing a tabu search approach and several neighborhood relations for solving the shift design problem.

The addressed problem concerns the design of shifts in a multiple days planning horizon and the determination of the number of employees assigned to each shift. It differs from the classical shift scheduling problem for a planning horizon that spans up to a week and in the fact that schedules are cyclic. Even though the authors consider a multiple days planning horizon, they do not address the assignment of real employees, with their days-off, rest periods and availabilities, and the problem cannot be classified as a tour scheduling problem. The developed method is based on local search, where several moves are introduced to define the neighborhood of a solution, which is explored according to the moves considered or according to a first descent strategy (i.e., the exploration is interrupted as soon as an improved solution is found). In order to reduce the portion of the neighborhood to be explored, they focus on days and time periods with under and over coverage. The algorithms have been included in a commercial software, which is used for solving shift design problems in several companies.

An extension of the shift design problem is considered in Gaspero et al. (2010), where the authors address the shift and break design problem. The issue is to find a minimum number of shifts, the number of workers assigned to them, the number of breaks and the intervals that they span in each shift. They propose a hybrid method that combines features of local search and constraint programming techniques. In particular, local search is used to design shifts and determine the number of employees assigned to shifts and breaks. Then, a constraint programming model determines the interval variables (i.e., start and length) of each break. This model is embedded in the local search procedure in two stages: after constructing an initial random solution and after performing a move. Computational results are performed to analyze the behavior of the developed hybrid method on the basis of its components. An overview of works on shift design and break scheduling is given in Gaspero et al. (2013), with a particular emphasis on approaches based on local search techniques.

The works presented in this section concern the definition of working and break periods during the working days of the employees. The next section deals with the shift scheduling problem in a multi-activity environment.

1.3.2.1 Multi-activity shift scheduling

When more than one activity is considered, the shift scheduling problem is referred to as multi-activity shift scheduling. The goal is not only the definition of working and break periods but also the specification of which activity each employee performs in each slot. In the following, papers addressing this problem are reviewed.

Côté et al. (2011a) propose two IP formulations inspired by two constraints using formal languages to solve the multi-activity shift scheduling problem. One model accepts any sequence recognized by an automaton, while the other model accepts any sequence belonging to a context-free grammar. The authors present, for both grammars, a system of linear equations in 0-1 variables able to identify any word recognized by the corresponding grammar.

Later, Côté et al. (2011b) propose an implicit formulation able to address the symmetry issues previously encountered. They explain that using models where variables describe whether an employee is assigned or not to an activity in a slot, exhibits a lot of symmetry in case of homogeneous employees. Indeed, the proposed implicit model uses variables indicating the number of employees assigned to an activity in a slot, and through the grammar defining feasible shifts, they are able to build the explicit individual schedules from the implicit solution.

Recently, Dahmen et al. (2018) show that multi-activity shift scheduling problems can be implicitly formulated using forward and backward constraints on a series of transportation problems. The sources of flexibility considered in defining shift types concern the starting time, the length of the shift, the duration of the meal break and the pre- and post-break work duration. Furthermore, minimum and maximum bounds are imposed on activities length, with the purpose of avoiding productivity losses and important physical or mental effort. The implicit model enumerates all possible profiles that can precede and follow the meal break, and it uses integer variables for each shift type and each profile. A transportation problem is defined for each shift type, where appropriate forward and backward constraints are used for matching pre-break and post-break profiles variables. The authors describe a procedure to build explicit shifts from the solution of the implicit model. Since pre-break and post-break profiles are exhaustively enumerated, they can easily handle different and complex rules on the profile feasibility. For instance, limiting the number of different activities worked at each profile is shown to be beneficial in reducing the total number of variables and the computational time, with a negligible degradation of the cost. Computational experiments are performed and the implicit model proposed in the paper is compared with the implicit one of Côté et al. (2011b). The performance of the two models depends on the characteristics of the instances considered. Indeed, the formulation of Dahmen et al. (2018) outperforms the one of Côté et al. (2011b) on instances where complex rules (i.e., number and length of activities) define feasible pre- and post-break profiles, limiting their number. The opposite behavior is shown on the instances proposed by Côté et al. (2011b), where no restrictions on activities duration are imposed.

MIP models suffer when dealing with large-scale instances, where the employees are heterogeneous and a high number of activities and a high degree of flexibility is considered. For this reason, other exact algorithms have been proposed in the literature based on CG and B&P algorithms.

Côté et al. (2013) present a B&P approach to solve the heterogeneous version of the problem presented in Côté et al. (2011b), where employees have different skills. The master problem is formulated using the classical set covering formulation, while the pricing problems are modeled by using context-free grammar that allows to define the feasible shifts for each employee. The proposed CG is embedded within a B&P algorithm and it is flexible enough to be adapted and solve a variety of problems. Although grammars flexibility enables to capture a large set of shift rules, some limitations are presented concerning the total number of working periods over a long planning horizon, such as one week.

Boyer et al. (2013) solve a shift scheduling problem with heterogeneous employees, where besides considering multiple activities, they also include multiple uninterruptible tasks. They propose a B&P algorithm that extends the one proposed by Côté et al. (2013). The pricing problems are formulated using context-free grammar, and they are able to model complex rules in the construction of feasible shifts for an employee. They give two formulations for task precedence constraints and they propose three different branching strategies. Computational results show that the proposed B&P algorithm can find integer solutions within 2 hours with an optimality gap lower than 5% in the best case.

Besides exacts methods, the literature exhibits different approaches based on relaxations and heuristics techniques, such as local search, large neighborhood search, and more complex matheuristics based on CG and B&P approaches.

Demassey et al. (2005) present a CG algorithm where the regulations defining shifts are modeled using regular language. In order to generate negative reduced cost schedules, a constraint called `cost-regular` is introduced. The authors address three classes of problems, presenting for each one the corresponding model. The first class considers homogeneous employees and the overall cost is given by the sum of the schedules assigned to each employee, while the second class includes also under and over covering with their additional costs. Finally, in the third class, employees are heterogeneous and their preferences are taken into account. Computational experiments are done on a set of generated benchmarks based on realistic data from a retail store falling in the first category. The results provide lower bounds on the optimal value of the original integer problem. An heuristic upper bound is evaluated solving the integer linear program with all the generated columns. The proposed algorithm has been extended later by the same authors in Demassey et al. (2006) where they improve the resolution of the pricing problem and propose a B&P algorithm. The approach is tested both on the already generated benchmarks from the retail store, and on complex real-world instances from a large bank. Even thought the method is efficient to solve the linear relaxation of the first instances, it fails to converge on the more complex weekly real-world ones. Furthermore, integer solutions are found only for small instances, with up to 3 activities.

Meisels and Schaerf (2003) propose a local search method, focusing on then hill climbing technique. The neighborhood structure is based on the type move `Replace`, where an employee who is working in a specific shift, is replaced by another employee who is not working. This move does not change the number of assignments for shift-activity pairs, therefore it preserves the satisfaction of workload constraints. Given a solution s, the new solution is selected according four different selection rules, which explore the neighborhood with different levels of steepness. The cost function is defined as the total number of constraint violations of the current solution, and the initial solution is constructed by a greedy procedure which builds a solution satisfying requirements constraints. In addition, the authors propose an extended generalized local search, which enlarges the neighborhood defining two other move types: `Insert` and `Delete`. The search space includes all possible assignments independently to the fact that the requirements are not satisfied. More complex selection rules are presented, while the cost function is still defined by constraint violations of the current solution. The difference consists in the fact that constraints are split into two categories: violations to minimum requirements and workloads, and violations to all other constraints. Furthermore a third component is taken into account and it measures the possibility for a partial solution to be complete. Therefore, the costs function is composed by three different components whose weights are adaptively modified. Computational experiments are performed on instances from the nurse rostering. The generalized local search show better performance compared to other local search methods. The authors also analyze the role of each single feature of their algorithm.

Quimper and Rousseau (2009) show that formal languages, such as regular and context-free languages, can be used to model shift scheduling regulations. From these languages they derive a large neighborhood search procedure. They proposed two Very Large Neighborhood Search (VLNS) procedures: the first uses regular language while the second uses context-free language. Concerning regular language, they define an automaton that recognizes the feasible shifts satisfying the scheduling rules. They create the associated expanded graph G where paths have a one-to-one correspondence with feasible shifts. The VLNS constructs an initial feasible solution by randomly selecting paths in the graph G. To improve the overall cost of this solution, they proceed as follows: they remove a schedule s in the initial solution and they assign to each edge in G a cost according to schedule s. The shortest path in the expanded graph G corresponds to the new schedule that replaces s. This operation is repeated until a local minimum is reached. Concerning context-free language, they obtain a feasible solution by constructing, for each employee, a random sequence. This is done by traversing the grammar graph from the root to the leaves and randomly branching on a pair of children nodes. As previously, they select a schedule s, and they replace it with a new schedule that improves the total cost. This is done assigning weights to the nodes in the grammar graph. This operation is repeated until a local minimum is reached. Tests on mono-activity instances show that the methods find a solution with an optimality gap of 1% within 25 seconds for every instance considered. Furthermore, they give upper bounds on the unsolved multi-activity instances from Demassey et al. (2006), with one day time horizon and a number of activities from 1 to 10.

Restrepo et al. (2012) propose a heuristic method based on CG to solve the set covering formulation of the multi-activity shift scheduling problem. More in details, the linear relaxation is solved by means of the CG, and the resulting reduced master problem is solved forcing the integrality constraint to obtain an integer solution. The subproblem is modeled as a resource constrained shortest path problem, where most of the rules defining shifts are embedded in the underlying graph. Computational results are performed on real word instances arising in a parking operator. Even though a week planning horizon is considered, the authors do not address the allocation of days-off.

Dahmen and Rekik (2015) solve a multi-activity shift scheduling problem in which, given day-off schedules associated with each employees, their objective is to construct and assign admissible multi-activity shifts to employees in their working days. Different shift types are considered, defined by starting time, shift duration, break duration and break window. Their hybrid heuristic combines the tabu search technique with an exact B&B procedure. The initial solution is built by a constructive algorithm. The classical tabu search is combined with intensification and diversification procedures. The intensification procedure generates new schedules that preserves some properties that often appeared in the solutions previously found. The idea is based on the fact that good properties tend to appear often in the best neighbor solutions. The diversification procedure, instead, considers a new initial feasible solution which includes properties that appear the least in the best neighbor solutions identified in the search process. The B&B procedure is embedded into the tabu search at three stages: the neighborhood exploration, the intensification and the diversification.

Recently, Hernández-Leandro et al. (2018) address the heterogeneous version of the multi-activity shift scheduling. The proposed formulation is a variant of the set partitioning model where both under coverage and over coverage are allowed and minimized. The problem is solved using a matheuristic based on Lagrangian relaxation. In particular, demand constraints are relaxed in the objective function and the subgradient method is employed to solve the Lagrangian dual. The resolution of the Lagrangian subproblems requires the generation of daily shifts, one for each employee, and context-free grammar is used to determine the shifts with minimum cost defined by the multipliers. All generated shifts are integrated into a restricted set partitioning problem, which is solved to obtain a feasible solution. The main idea is to use Lagrangian relaxation to identify only the promising shifts to be considered, reducing in this way the number of variables and the complexity of the problem. Experiments are performed on instances from the literature and the results show that the proposed method substantially decreases the computational time and it improves in many cases the best solution known, obtained by Côté et al. (2013).

1.3.3 Tour scheduling

In the attempt to address more complex and realistic problems in personnel scheduling, many researchers have focus on the tour scheduling, that involves not only the design of the shifts during the working days, but also the assignment of the days-off.

Bailey (1985) proposes one of the first implicit model for the tour scheduling problem with no restriction on shift starting time. Besides this source of flexibility, shifts have a fix duration of eight hours and do not include any break. The proposed model makes use of integer variables associated to shift patterns and days-off patterns. Furthermore, under coverage slack variables are added to demand constraints, and they are penalized in the objective function. The author presents a heuristic to allocate shifts to days-off pattern and obtain complete tours from the implicit solution. Computational results show that the proposed model leads to cost saving when compared to a two phase linear programming formulation, that first solves the days-off problem to determine the workforce available each day, and then it solves seven shift scheduling problems.

When no restriction on the starting time is considered, it may happens that the complete tour does not provide enough rest between consecutive shifts. Bailey (1985) suggested to divide the 24-hour day into three overlapping 12-hour periods, and to limit each employee to work the same period throughout the week. However, he does not show how to integrate this extension in the model. In this regard, Jacobs and Brusco (1996) introduce the concept of start-time band, defined as a block of shift starting times, and they propose an implicit model that makes use of shift and days-on variables. The model is used on real instances to schedule tool collectors, and the results reveal the benefit of start-time bands in reducing the workforce size compared with fixed shift starting time.

Later, Brusco and Jacobs (2000) consider a higher degree of flexibility by introducing meal break windows, in addition to start-time bands. The implicit model proposed makes use of shift, working days and break variables, where the shift and break variables are linked by the forward and backward constraints introduced in Bechtold and Jacobs (1990). Computational results performed on real instances show that the different sources of flexibility considered interact with each other, and the effect of one source on the optimal workforce size may vary depending on the level of the other sources.

Besides implicit formulations, heuristic methods have been proposed for the tour scheduling problem. Some of them are based on CG to solve the explicit set covering or set partitioning formulations, where each feasible tour is represented with an integer variable.

Al-Yakoob and Sherali (2008) propose a heuristic CG for assigning heterogeneous employees to gas stations, which can be seen as activities. Three different shifts are considered and they cover the complete day. The problem is formulated as a set partitioning model, whose linear relaxation is solved by means of CG. To achieve integrality, a variable fixing procedure generates feasible schedules for each employee. The CG subproblem consists of a MIP constraining the shifts that can be assigned in two successive days, together with the number of total and consecutive weekly working days. Although the limited number of shifts considered, computational results show that the method is able to generate a solution for instances with up to 90 stations and 336 employees.

Easton and Rossin (1991) consider a tour scheduling problem where a weekly tour needs to be defined for a heterogeneous workforce. Employees may have a full-time contract that imposes five consecutive working days with shifts of nine hours (four hours working followed by one hour meal break and four hours of working) starting in every moment of the day. In addition, employees can have a part-time contract where shifts have a fix duration of four hours without any break. The proposed model is based on a generalized set covering formulation where, besides the classical demand constraints, the authors introduce constraints restricting the number of employees in a particular class to a fraction of the total workforce. They propose a heuristic method based on CG, where the problem is solved as an integer program as soon as all feasible tours needed to prove the optimality of the linear relaxation are generated. Computational results show that the heuristic yields to lower cost solutions in less time when compared to other heuristic algorithms reported in the literature.

Starting from the scheduling environment presented in Easton and Rossin (1991), Brusco and Jacobs (1993) introduce a higher degree of flexibility. Indeed, meal breaks are allowed to start at different times during the shift, the placement of meal breaks and the shift starting time can float from one day to another, and days-off are not imposed to be consecutive. The authors propose a simulated annealing heuristic where a neighbor of a solution is found by dropping a fixed number of employees whose tours cause the highest over coverage. The obtained partial solution is then returned to feasibility by generating new tours with the best coverage rate: for each day, the shift that better covers the demand is selected, and a complete tour is built by combining the five best shifts over the week. Computational results show that the additional flexibility considered allows a considerable reduction in employees requirements.

Brunner and Bard (2013) solve a tour scheduling over one week planning horizon arising in mail processing and distribution centers, where flexibility includes different shift starting time, different shifts lengths, lunch break allocation and different days-off assignments. Furthermore, two groups of employees are considered: the first group consists of full-time or regular employees, while the second group consists of part-time or flexible employees. The authors developed a set covering formulation to solve the linear relaxation of the problem and an exact B&P algorithm for obtaining integer solutions. Two different constraint-based formulations are presented to solve the two subproblems, one for regular and one for flexible employees. In the first, shifts are explicitly enumerated, while in the second, the idea of implicit shift construction is exploited. Computational results analyze the impact of the different sources of flexibility on the size of the workforce and the utilization rate (i.e., the percentage of time that the workforce is active). They show that considerable cost savings are possible when a high degree of flexibility is considered in the problem.

1.3.3.1 Multi-activity tour scheduling

When more than one activity is considered, the tour scheduling problem is called multi-activity tour scheduling. While the multi-activity context has been intensively studied for the shift scheduling problem, to the best of our knowledge, it has been considered for the tour scheduling problem only recently. Both exact and heuristic methods have been proposed.

Gérard et al. (2016) consider a multi-activity tour scheduling problem with heterogeneous employees. The objective is to compute employees schedules in order to minimize under and over coverage. The authors develop four methods: a compact MILP model; a B&P based approach with a nested dynamic programming to solve heuristically the subproblems; a diving heuristic; a greedy heuristic based on their subproblem solver. Concerning the second approach, in order to solve the subproblem they do not use constrained resource shortest path formulation since they state that it is not efficient when both lower and upper bounds are considered. Moreover, they do not employ grammar-based formulations since the considerable number of bound constraints and a long time horizon result in a huge hypergraph. The proposed nested dynamic programming algorithm decomposes the individual planning in different levels: task, timeslot, shift and individual planning (schedule). They solve the problem starting from the inner level, and going up to the outer level. In the branching step, they choose the most fractional triplet employee-activity-slot and they use a depth-first strategy to explore the search tree. They heuristically accelerate the algorithm by simplifying the assignment of breaks and pauses and by restricting the state space. In the third approach, they fix a complete schedule for one selected employee at each node of the search tree. The subproblem associated to this employee is no more called in descendant nodes. In the last approach, the subproblems are still solved with their nested dynamic program, but planning are iteratively removed and generated one by one and added to the solution. In this approach, the dual variables are estimated considering the residual work demand. The objective function of the subproblems is based on the residual work demand which takes into account the satisfied demand by the schedules already in the partial solution. Computational results performed both on random and mini-marts[2] instances show that the greedy heuristic is very fast but it can be far from the optimal solution; the compact method gives optimal solution only for small instances; the heuristic B&P terminates within 24 hours time limit only for half of the instances, but it is able to find the optimal solution for these instances; the diving heuristic is more effective and outperforms the MIP solver both in terms of execution time and quality solution.

In parallel with Gérard et al. (2016), Restrepo et al. (2016) propose two B&P approaches with two different formulations of the multi-activity tour scheduling problem, where heterogeneous employees are considered. Both formulations are based on Dantzig-Wolfe decomposition and make use of context-free grammar to design feasible daily shift. The difference consists in the way these daily shifts are combined into a complete schedule. Indeed, the first one, called daily-based formulation, links them in the master problem by means of extra constraints that assure all regulations imposed on a schedule, such as the minimum and the maximum weekly working hours, the minimum and the maximum number of working days and the minimum rest time between consecutive daily shifts. The second one, called tour-based formulation, assembles the daily shifts in the pricing problem using a resource constrained shortest path problem. For both formulations branching strategies are proposed. However, an intense variable fixing strategy is adopted for the tour-based formulation, making the proposed B&P heuristic. Indeed, besides branching, at each node of the search tree, the employee with the largest fractional variable is identified, and its schedule is fixed to one. Computational results on random instances show that the daily-based formulation do not show good performance since it is not able to improve the integer solution found at the root node within one hour, while the tour-based formulation finds an integer solution with an optimality gap of 1% for all instances considered.

Recently, Restrepo et al. (2018) present an approach that combines Benders decomposition and CG. However, differently from the previous work, employees are supposed to be homogeneous. The model is decomposable by days, and it consists of a Benders master problem and a set of Benders daily subproblems. The Benders master problem includes the variables associated with employee schedules and with daily shifts shells, where activities and breaks are not assigned. The Benders subproblems include the variables related to the allocation of work activities and breaks to daily shifts and to the under and over coverage of the demand. Since the number of feasible schedule can be too large to be completely enumerated, the authors solve the Benders master problem by means of CG. The Benders subproblems are modeled with the implicit grammar based integer programming model presented in Côté et al. (2011b).

[2]A mini-mart is a small retail business that stocks a range of everyday items. Mini-marts usually charge significantly higher prices than conventional supermarkets, and have longer open hours.

1.4 Instances

This section presents the instances of multi-activity tour scheduling used to evaluate the performance of the solving methods developed in this thesis. To the best of our knowledge, there are two works in the literature that consider a problem which is very close to ours: Restrepo et al. (2016) and Gérard et al. (2016). We were able to obtain only the instances considered by the first authors. However, we will see that the degree of flexibility in these instances is reduced compared to the one faced by the company Horizontal Software. Due to the fact that this thesis is conducted in an industrial context, it is crucial to verify the behavior of the solution methods on instances that correspond to the reality of the business. For this reason, both generated and real-world instances provided by Horizontal Software have been considered.

Three different sets of multi-activity tour scheduling instances are used for the computational experiments. The first set, called **RGR**, comes from the work of Restrepo et al. (2016). The main feature of these instances is that only three types of daily shift are allowed. The second set of instances is called **RGR flexible**, and it consists of a more flexible variant of RGR where no particular structure in imposed on the daily shift. As a consequence, these instances present a higher degree of flexibility when compared to RGR. Finally, the third set is called **Real** and it has the highest degree of flexibility. It consists of instances coming from a fast food restaurant chain, one of the business to which Horizontal Software provides its solution.

The details of three sets are given in the following paragraphs. However, all instances share some common features: they are all defined over a one week planning horizon and the quality of the planning is evaluated in the same way. Indeed, the cost of performing an activity in one slot is fixed to 15, such as the cost for every transition between activities, and under and over assignment costs are set respectively to 100 and 10.

RGR instances. This set of instances from the work of Restrepo et al. (2016) considers three types of daily shifts, that is four-hour shift, six-hour shift and eight-hour shift with one-hour break in the middle. The time unit is fixed to 15 minutes. Furthermore, the following bounds are imposed:

- the activities duration goes from 30 minutes to 6 hours, depending on the instance (L1);
- the weekly working hours fall between 35 and 40 hours (L7);
- the number of working days falls between 5 and 6 days (L9);
- the rest between consecutive daily shifts goes from 12 to 60 hours (L10).

Due to the predefined structure of the daily shifts, the bounds of some legal constraints are implicitly imposed. For instance, the consecutive working time is either 4 or 6 hours (L2), only one break can be assigned with a fixed duration of 1 hour and no interruption is allowed (L3), the daily working time goes from 4 to 8 hours (L4), and the amplitude of the working day goes from 4 to 9 hours (L5). Finally, no bounds are imposed on the consecutive working days (L8).

RGR instances are further divided in three different groups G1, G2 and G3, which differ in the type of daily shifts considered and in the slots in which they are allowed to start (L6).

- Group G1: it considers the three types of daily shifts, and each of them can start at any slot, taking care that no daily shift starts before the first slot with workload. As consequence, the number of feasible starting slots depends on the demand.
- Group G2: it considers two types of daily shifts, that is four-hour shift and eight-hour shift with one-hour break in the middle. The starting slots are the same of group G1.
- Group G3: it considers all three types of daily shifts, and each of them can start only at five predefined starting slots, that is 12am, 4am, 8am, 12pm and 3pm.

As shown in Table 1.1, instances are labeled with the format `E_A_D_V_G`, where `E`, `A`, `D`, `V`, and `G` represent the number of employees, number of activities, length of the planning horizon in days, version of the instance and group, respectively.

G1	G2	G3	
20_1_7_v1_G1	20_3_7_v1_G2	20_1_7_v1_G3	20_3_7_v1_G3
20_1_7_v2_G1	20_3_7_v2_G2	20_1_7_v2_G3	20_3_7_v2_G3
25_1_7_v1_G1	20_3_7_v3_G2	25_1_7_v1_G3	20_3_7_v3_G3
25_1_7_v2_G1		25_1_7_v2_G3	20_5_7_v1_G3
40_1_7_v1_G1		40_1_7_v1_G3	20_5_7_v2_G3
40_1_7_v2_G1		40_1_7_v2_G3	

Table 1.1: RGR instances names.

RGR flexible instances. This set considers the RGR instances of groups G1 and G2, without imposing the daily shifts to be either a four-hour, or a six-hour or an eight-hour shifts. The bounds of the constraints concerning the activities duration (L1), the weekly working hours (L7), the number of working days (L9), and the rest between consecutive daily shifts (L10) are imposed as in RGR. In addition, the daily shifts can start at any slot (L6), such as groups G1 and G2 of RGR. Due to the fact that the daily shifts do not have a predefined structure, RGR flexible instances impose also the following bounds:

- the consecutive working hours goes from 4 to 6 hours (L2);
- one break and one interruption are allowed each day; the first has a fixed duration of 1 hour, while the second has a duration that goes from 2 to 5 hours (L3);
- the daily working hours goes from 4 to 8 hours (L4);
- the amplitude of the working day goes from 4 to 12 hours (L5).

Instances are labeled with the format `E_A_D_V_G` used for RGR in order to easily identify which instance has been considered and transformed into a RGR flexible one. We recall that `E`, `A`, `D`, `V`, and `G` represent respectively the number of employees, number of activities, length of the planning horizon in days, version of the instance and group. All instances RGR flexible are identified with group G4.

G4
20_1_7_v1_G4
20_1_7_v2_G4
20_3_7_v1_G4
20_3_7_v2_G4
20_3_7_v3_G4

TABLE 1.2: RGR flexible instances names.

Real instances. This set contains real instances provided by the company Horizontal Software, and they come from a fast food restaurant chain. Their main characteristic is their high degree of flexibility, which results in a high number of feasible daily shifts and, therefore, feasible schedules. Indeed, unlike in Restrepo et al. (2016), daily shifts are not limited to be either four-hour, six-hour or eight-hour shifts. In addition, daily shifts with up to three timeslots are considered that can be divided by either a break or an interruption. The latter is typical in fast food restaurant chains and it has the goal to better handle rush periods where the demand for employees is higher, such as lunchtime and dinner.

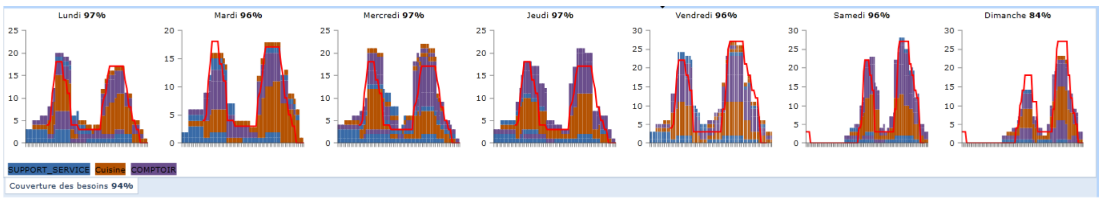


FIGURE 1.1: Demand profile example with rush periods for Real instances from E-Optim.

Instances are labeled with the format `E_A_T_V`, where `E`, `A`, `T` and `V` represent respectively the number of employees, number of activities, time unit and some informations on the version of the instance.

60 minutes	30 minutes	15 minutes
57_11_60_le	57_11_30_le	57_11_15_le
75_14_60_ll	75_14_30_ll	75_14_15_ll
43_15_60_lb	43_15_30_lb	43_15_15_lb
38_05_60_nh	38_05_30_nh	38_05_15_nh
23_05_60_nn	23_05_30_nn	23_05_15_nn

TABLE 1.3: Real instances names.

The constraint bounds imposed in these instances are summarized in Table 1.4. All values are reported in hours. The first column presents the name of the instance. The second and third columns show respectively the consecutive working hours (c_wkh) (L2), and the daily working hours (d_wkh) (L4). The amplitude of the working day ($ampl$) (L5) is reported in the fourth column. The duration of the break (dur_b) and the duration of the interruption (dur_i) (L3) are shown in the fifth and sixth columns, while the rest after a daily shift (*rest_d*) and the rest after a night shift (*rest_n*) (L10) are presented in the last two columns. For these instances, a shift is considered a night shift if it ends in the late evening, i.e. after 10pm.

Instance	c_wkh	d_wkh	ampl	dur_b	dur_i	rest_d	rest_n
57_11_60_le	[2,5]	≤ 10	≤ 12	1	[2,5]	≥ 11	≥ 12
75_14_60_ll	[2,4]	≤ 10	≤ 12	1	[2,5]	≥ 11	≥ 12
43_15_60_lb	[2,5]	≤ 10	≤ 12	1	[2,5]	≥ 11	≥ 12
38_05_60_nh	[2,5]	≤ 9	≤ 10	1	[2,6]	≥ 11	≥ 12
23_05_60_nn	[2,5]	≤ 9	≤ 10	1	[2,5]	≥ 11	≥ 12
57_11_30_le	[2,5]	≤ 10	≤ 12	0.5	[2,5]	≥ 11	≥ 12
75_14_30_ll	[2,4]	≤ 10	≤ 12	0.5	[2,5]	≥ 11	≥ 12
43_15_30_lb	[2,5]	≤ 10	≤ 12	0.5	[2,5]	≥ 11	≥ 12
38_05_30_nh	[2,5.5]	≤ 9	≤ 10.5	1	[2,6]	≥ 11.5	≥ 12.5
23_05_30_nn	[2,5.5]	≤ 9	≤ 10	1	[2,5]	≥ 11.5	≥ 12.5
57_11_15_le	[2,5]	≤ 10	≤ 12	0.5	[2,5]	≥ 11	≥ 12
75_14_15_ll	[2,4]	≤ 10	≤ 12	0.5	[2,5]	≥ 11	≥ 12
43_15_15_lb	[2,5]	≤ 10	≤ 12	0.5	[2,5]	≥ 11	≥ 12
38_05_15_nh	[2,5.75]	≤ 9	≤ 10.5	0.75	[2,5.75]	≥ 11.5	≥ 12.5
23_05_15_nn	[2,5.75]	≤ 9	≤ 10	0.75	[2,5]	≥ 11.5	≥ 12.5

TABLE 1.4: Real instances description.

In addition to the constraint bounds reported in Table 1.4, these instances consider also the following bounds:

- the activities duration goes from 1 to 5 hours, depending on the instance (L1);
- the daily and night shifts have to start after the restaurant has opened (7am) and have to end before the restaurant has closed time (2am) (L6);
- the weekly working hours are 24 or 35 depending if the employee has part-time or a full time contract; in addition, the availabilities of the employee may limit the working hours that he can perform (L7);
- the number of working days does not exceed 5 (L9), such as the number of consecutive working days (L8); note that the two bounds coincide, and there is not a real restriction on the consecutive working days.

We point out the high degree of flexibility that we have in defining a feasible daily shift. This is mostly due to the large interval for the consecutive working hours and the interruption duration, along with the high upper bound for the amplitude of the working day. As consequence, we have a high number of feasible daily shifts and, therefore, feasible schedules.

1.5 Conclusions

In this chapter we have described the multi-activity tour scheduling problem addressed in this thesis. Besides the various constraints defining the feasibility of the schedules, the problem is characterized by high levels of heterogeneity and flexibility. The first comes from employees skills, availabilities and contract types, while the second is determined by the legal regulations and by the introduction of the interruption pause. The literature review shows the variety of the problems addressed and the solution methods proposed. Even within the same class, problems may differ in degree of flexibility considered, employees homogeneity and constraints considered. As a result, in many cases the proposed methods are not well suited to be adapted when other problems are considered.

In the next chapter we formalize the multi-activity tour scheduling problem throughout a compact MILP model. In addition, we show that the problem is well-suited for the Dantzig-Wolfe decomposition.

Chapter 2

Mathematical Model

This chapter presents two mathematical formulations of the multi-activity tour scheduling problem addressed in this thesis: a compact MILP model and a Dantzig-Wolfe decomposition. The compact MILP model described has the goal of minimizing the total cost of the planning, given by under and over coverage penalizations, activities costs and activities transition costs. The various complex constraints introduced in Section 1.2.3 are formulated by means of linear constraints. The presented model is inspired by the one proposed in Gérard et al. (2016). However, some extensions need to be carried out in order to consider the following elements:

- different types of pauses, namely breaks and interruptions;
- different rest durations between two consecutive daily shifts;
- predefined length of the timeslots;
- predefined starting slots of the daily shifts;
- transition between activities in the timeslots.

When building a schedule for an employee, the minimal element that can be assigned is the *activity* in a slot. Moreover, a sequence of the same activity performed consecutively is called *task*, while a sequence of tasks without any pause is called *timeslot*. Finally, a sequence of timeslots and pauses is called *daily shift*, and a sequence of daily shifts and days-off, covering the whole planning horizon, is called *schedule*. We will see that the definition of the variables reflects the layered structure of the schedule.

The chapter is organized as follows: Section 2.1 formulates the problem in the form of a compact MILP, while Section 2.2 presents the Dantzig-Wolfe decomposition. Conclusions in Section 2.3 close the chapter.

2.1 Compact MILP model

2.1.1 Notations

This section introduces the notations used in the mathematical formulation of our multi-activity tour scheduling problem. Firstly, we define the sets to which the variables indices belong. For instance, we find the set of employees, the set of activities and the set of slots. Then, we introduce the input data, that mainly consist in the bounds imposed by the different constraints. Finally, we describe the variables used in the model.

2.1.1.1 Sets

The following list reports the sets to which the variables indices belong.

- I: set of employees;
- J: set of slots;
- A: set of activities;
- A_i: set of activities for which employee i has the skills;
- D: set of days covering the planning horizon;
- J_d: set of slots in day d ($J_d \subseteq J$);
- N: set of night slots ($N \subseteq J$);
- S: set of daily shifts;
- S_i: set of daily shifts of employee i ($S_i \subseteq S$);
- S_d: set of daily shifts in day d ($S_d \subseteq S$);
- T: set of timeslots;
- T_s: set of timeslots in daily shift s ($T_s \subseteq T$);
- B: set of breaks;
- B_s: set of breaks in daily shift s ($B_s \subseteq B$);
- C: set of interruptions;
- C_s: set of interruptions in daily shift s ($C_s \subseteq C$);
- K: set of tasks;
- K_t: set of tasks in timeslot t ($K_t \subseteq K$);
- K_{ta}: set of tasks in timeslot t where activity a is performed ($K_{ta} \subseteq K$);
- R_i: pre-assignments of employee i ($R_i \subseteq J \times A$);
- U_i: unavailable slots of employee i ($U_i \subseteq J$)
- L_t: set of predefined lenghts for timeslot t;
- J_s: set of predefined starting slots for daily shift s ($J_s \subseteq J$);
- A_f: set of forbidden successions of activities ($A_f \subseteq A \times A$).

2.1.1.2 Input data

The following list reports the input data of the problem.

- b_{ja}: demand for activity a in slot j;
- c^i_{ja}: cost for the assignment of activity a in slot j to employee i;
- $\underline{c}_{ja}$: under assignment cost for activity a in slot j;
- $\bar{c}_{ja}$: over assignment cost for activity a in slot j;
- c_t: cost for each transition between activities in timeslot t;
- $[l_a, u_a]$: bounds for the duration of activity a;
- $[l^c_w, u^c_w]$: bounds for the consecutive working hours;
- $[l^d_w, u^d_w]$: bounds for the daily working hours;
- $[l^w_w, u^w_w]$: bounds for the weekly working hours;
- $[l_m, u_m]$: bounds amplitude daily shifts;
- $[l_b, u_b]$: bounds for the duration of breaks;
- $[l_c, u_c]$: bounds for the duration of interruptions;
- l^d_r: minimum rest after daily shift;
- l^n_r: minimum rest after night shift;
- $[l^n_d, u^n_d]$: bounds on the number of working days;
- $[l^c_d, u^c_d]$: bounds on the number of consecutive working days;
- $[l^s_d, u^s_d]$: bounds on the starting and finishing time for daily shift s;
- $[l^a_d, u^a_d]$: bounds on the total number of slots spent performing activity a on day d;
- $[l^a_n, u^a_n]$: bounds on the number of times activity a is performed.

2.1.1.3 Variables

The variables appearing in the model can be divided into different groups concerning activities, tasks, timeslots, breaks, interruptions and daily shifts. In the following we define the variables belonging to each group.

Activities. The variables concerning activities are the following:

- $x^i_{ja} \in \{0,1\}$: 1 if employee $i \in I$ performs activity $a \in A$ in slot $j \in J$, 0 otherwise;
- $\underline{y}_{ja} \in \mathbb{R}^+$: under assignment for activity $a \in A$ in slot $j \in J$;
- $\bar{y}_{ja} \in \mathbb{R}^+$: over assignment for activity $a \in A$ in slot $j \in J$.

Tasks. The variables concerning tasks are the following:

- $x^K_k \in \{0,1\}$: 1 if task $k \in K$ is performed, 0 otherwise;
- $y^K_{kj} \in \{0,1\}$: 1 if task $k \in K$ has already started in slot $j \in J$, 0 otherwise;
- $z^K_{kj} \in \{0,1\}$: 1 if task $k \in K$ has already finished in slot $j \in J$, 0 otherwise.

Timeslots. The variables concerning timeslots are the following:

- $x^T_t \in \{0,1\}$: 1 if timeslot $t \in T$ is performed, 0 otherwise;
- $y^T_{tj} \in \{0,1\}$: 1 if timeslot $t \in T$ has already started in slot $j \in J$, 0 otherwise;

- $z^T_{tj} \in \{0,1\}$: 1 if timeslot $t \in T$ has already finished in slot $j \in J$, 0 otherwise;
- $l^T_t \in \mathbb{R}^+$: length, or duration, of timeslot $t \in T$;
- $n^T_t \in \mathbb{R}^+$: number of transitions between activities in timeslot $t \in T$;
- $v^T_{tl} \in \{0,1\}$: 1 if timeslot $t \in T$ has length $l \in L_t$, 0 otherwise.

Breaks and interruptions. The variables concerning breaks and interruptions are the following:

- $x^B_b \in \{0,1\}$: 1 if break $b \in B$ is performed, 0 otherwise;
- $y^B_{bj} \in \{0,1\}$: 1 if break $b \in B$ has already started in slot $j \in J$, 0 otherwise;
- $z^B_{bj} \in \{0,1\}$: 1 if break $b \in B$ has already finished in slot $j \in J$, 0 otherwise;
- $x^C_c \in \{0,1\}$: 1 if interruption $c \in C$ is performed, 0 otherwise;
- $y^C_{cj} \in \{0,1\}$: 1 if interruption $c \in C$ has already started in slot $j \in J$, 0 otherwise;
- $z^C_{cj} \in \{0,1\}$: 1 if interruption $c \in C$ has already finished in slot $j \in J$, 0 otherwise.

Daily shifts. The variables concerning daily shifts are the following:

- $x^S_s \in \{0,1\}$: 1 if daily shifts $s \in S$ is performed, 0 otherwise;
- $y^S_{sj} \in \{0,1\}$: 1 if daily shifts $s \in S$ has already started in slot $j \in J$, 0 otherwise;
- $z^S_{sj} \in \{0,1\}$: 1 if daily shifts $s \in S$ has already finished in slot $j \in J$, 0 otherwise;
- $l^S_s \in \mathbb{R}^+$: length, or duration, of daily shifts $s \in S$, 0 otherwise;
- $b^S_s \in \mathbb{R}^+$: starting time of daily shifts $s \in S$;
- $f^S_s \in \mathbb{R}^+$: finishing time of daily shifts $s \in S$;
- $v^S_{sj} \in \{0,1\}$: 1 if daily shifts $s \in S$ starts in slot $j \in J_s$, 0 otherwise.

Using the variables previously introduced, we are able to give the mathematical model of the problem presented in Section 1.2. In particular, we first formulate the objective function, followed by the constraints, which are divided according to their category. We find *workload constraints* that impose the coverage of the demand; *legal constraints* that impose the satisfaction of the contract regulations; *activities cardinality constraints* that impose a single activity to be performed in each slot; *skills constraints* avoid that employees are assigned to activities for which they do not have the competencies; *pre-assignment constraints* that specify when and to which activity each employee is already assigned; *unavailability constraints*, that identify the slots where each employee cannot be assigned to any activity; *succession constraints*, that tell the sequences of activities that are not feasible; *distribution constraints*, that specify the duration and the number of times that an activity can be performed in a period or in a set of periods.

2.1.2 Objective function

The objective function (2.1) aims at minimizing three elements: the first one consists in the sum of the costs c^i_{ja} of assigning employee $i \in I$ to activity $a \in A$ in slot $j \in J$; the second one consists in the total costs $\underline{c}_{ja}$ and $\bar{c}_{ja}$ of respectively under assignment and over assignment, for each slot $j \in J$ and activity $a \in A$; the third one minimizes the number of times activities change in each timeslot $t \in T$.

$$\min \sum_{i\in I}\sum_{j\in J}\sum_{a\in A} c^i_{ja}x^i_{ja} + \sum_{j\in J}\sum_{a\in A}\left(\underline{c}_{ja}\underline{y}_{ja} + \bar{c}_{ja}\bar{y}_{ja}\right) + \sum_{t\in T} c_t n^T_t. \tag{2.1}$$

2.1.3 Workload constraints

The demand needs to be satisfied for each activity a and in each slot j. This is imposed by the following equations:

$$\sum_{i\in I} x^i_{ja} + \underline{y}_{ja} - \bar{y}_{ja} = b_{ja}, \qquad \forall j \in J, \forall a \in A. \tag{2.2}$$

2.1.4 Legal constraints

Legal constraints are imposed by the contract regulations and mainly define the feasibility of a schedule. In the following we present their mathematical formulation, together with the constraints needed to link the different variables that describe activities, tasks, timeslots, breaks, interruptions and daily shifts.

Tasks. The set of constraints presented in this paragraph allows to impose the legal regulation (L1) on activities duration.

Variables x^i_{ja} are linked to task variables y^K_{kj} and z^K_{kj} by means of constraints (2.3). The difference between y^K_{kj} and z^K_{kj} is equal to 1 only if task k is performed during slot j. Therefore, if in task k activity a is performed, then variable x^i_{ja} need to be equal to 1.

$$x^i_{ja} = \sum_{s\in S_i}\sum_{t\in T_s}\sum_{k\in K_{ta}} (y^K_{kj} - z^K_{kj}), \qquad \forall i \in I, \forall j \in J, \forall a \in A. \tag{2.3}$$

Constraints (2.4)-(2.8) define the relation between variables that concern the starting, the ending and the performing of task k. Indeed, if task k starts in slot j, then all variables $y^K_{kj'}$ have value 1 for slots j' following j according to (2.4). Analogously, if task k ends in slot j, constraints (2.5) impose that all variables $z^K_{kj'}$ have value 1 for slots j' following j. Constraints (2.6) assure that task k ends after it starts and its duration is at least 1 slot, while constraints (2.7) and (2.8) impose x^K_k equal to 1 only if task k is performed.

$$y^K_{kj} \le y^K_{k(j+1)}, \qquad \forall j \in J, \forall k \in K, \tag{2.4}$$

$$z^K_{kj} \le z^K_{k(j+1)}, \qquad \forall j \in J, \forall k \in K, \tag{2.5}$$

$$z^K_{k(j+1)} \le y^K_{kj}, \qquad \forall j \in J, \forall k \in K, \tag{2.6}$$

$$y^K_{kj} \le x^K_k, \qquad \forall j \in J, \forall k \in K, \tag{2.7}$$

$$x^K_k \le \sum_{j\in J}(y^K_{kj} - z^K_{kj}), \qquad \forall k \in K. \tag{2.8}$$

Each activity a can be performed during a number of consecutive slots bounded from above and from below (L1). Therefore, constraints (2.9) impose lower and upper bounds on the length of all tasks k where activity a is performed.

$$l_a x_k^K \leq \sum_{j \in J} (y_{kj}^K - z_{kj}^K) \leq u_a x_k^K, \qquad \forall a \in A, \forall t \in T, \forall k \in K_{ta}. \tag{2.9}$$

To forbid that two tasks k and $k+1$, where the same activity a is performed, are concatenated, we impose constraints (2.10), which avoid the maximum duration of activity a to be violated.

$$y_{(k+1)(j+1)}^K \leq z_{kj}^K, \qquad \forall j \in J, \forall a \in A, \forall t \in T, \forall k, k+1 \in K_{ta}. \tag{2.10}$$

Timeslots. The set of constraints presented in this paragraph allows to impose the legal regulation (L2) on the consecutive working hours.

The link between variables concerning tasks and variables concerning timeslots is defined by constraints (2.11) and (2.12). The first ensure that timeslot t is not performed if none of the related tasks is done. The second impose that timeslot t is worked if one of its tasks is performed. Furthermore, if employee i performs a timeslot t that covers slot j, then the sum of variables x_{ja}^i over all activities needs to be equal to 1 (constraints (2.13)).

$$x_t^T \leq \sum_{k \in K_t} x_k^K, \qquad \forall t \in T, \tag{2.11}$$

$$y_{kj}^K - z_{kj}^K \leq y_{tj}^T - z_{tj}^T, \qquad \forall j \in J, \forall t \in T, \forall k \in K_t, \tag{2.12}$$

$$\sum_{a \in A} x_{ja}^i = \sum_{s \in S_i} \sum_{t \in T_s} (y_{tj}^T - z_{tj}^T), \qquad \forall i \in I, \forall j \in J. \tag{2.13}$$

Similarly to the tasks, constraints (2.14)-(2.18) define the relation between variables that concern the starting, the ending and the performing of timeslot t. Constraints (2.14) and (2.15) impose that if timeslot t starts (ends) in slot j, then all variables $y_{tj'}^T$ ($z_{tj'}^T$) take value 1 for all slots j' following j. Furthermore, timeslot t ends after it starts and its duration is at least 1 slot (constraints (2.16)), while x_t^T takes value 1 only if timeslot t is performed (constraints (2.17) and (2.18)).

$$y_{tj}^T \leq y_{t(j+1)}^T, \qquad \forall j \in J, \forall t \in T, \tag{2.14}$$

$$z_{tj}^T \leq z_{t(j+1)}^T, \qquad \forall j \in J, \forall t \in T, \tag{2.15}$$

$$z_{t(j+1)}^T \leq y_{tj}^T, \qquad \forall j \in J, \forall t \in T, \tag{2.16}$$

$$y_{tj}^T \leq x_t^T, \qquad \forall j \in J, \forall t \in T, \tag{2.17}$$

$$x_t^T \leq \sum_{j \in J} (y_{tj}^T - z_{tj}^T), \qquad \forall t \in T. \tag{2.18}$$

Variable l_t^T defines the length of timeslot t (constraints (2.19)), and it is bounded from below and above by means of constraints (2.20). Bounds l_w^c and u_w^c correspond to the minimum and maximum consecutive working hours bounds (L2).

$$l_t^T = \sum_{j \in J} (y_{tj}^T - z_{tj}^T), \qquad \forall t \in T, \tag{2.19}$$

$$l_w^c x_t^T \leq l_t^T \leq u_w^c x_t^T, \qquad \forall t \in T. \tag{2.20}$$

Constraints (2.21) force variable n_t^T to be equal to the number of times activities change in timeslot t. It corresponds to the number of tasks assigned in t minus one, since a timeslot where only one task is assigned does not change activity.

$$n_t^T \geq \sum_{k \in K_t} x_t^T - 1, \qquad \forall t \in T. \tag{2.21}$$

It may happen that variables l_t^T are imposed to take value in a discrete set L_t. For example, we want the duration l_t^T of timeslot t to be equal either to 4, or to 6 or to 8, that is $L_t = \{4, 6, 8\}$. Constraints (2.20) is no more suitable to deal with this situation, since it forces l_t^T to be bounded from below and from above. For this reason, we use variables v_{tl}^T, which take value 1 if timeslot t has length l. Then, constraints (2.22) and (2.23) impose that, if timeslot t is performed, its duration take one of the values in L_t.

$$x_t^T = \sum_{l \in L_t} v_{tl}^T, \qquad \forall t \in T, \tag{2.22}$$

$$l_t^T = \sum_{l \in L_t} l v_{tl}^T, \qquad \forall t \in T. \tag{2.23}$$

Breaks and interruptions. The set of constraints presented in this paragraph allows to impose the legal regulation (L3) on the duration of breaks and interruptions.

The link between variables concerning timeslots, breaks and interruptions is defined by constraints (2.24)-(2.26). In particular, the first ensure that, if two consecutive timeslots are worked, also a pause must be performed. The pause can be either a break or an interruption. The second impose that the pause starts after the end of the previous timeslot t_h , while the third impose that the following timeslot t_{h+1} starts exactly when the pause ends.

$$x_{b_h}^B + x_{c_h}^C = x_{t_{h+1}}^T, \qquad \forall j \in J, \forall s \in S, \forall h \in \{0, \dots, |T_s| - 1\}, \tag{2.24}$$

$$y_{b_h j}^B + y_{c_h j}^C \leq z_{t_h j}^T, \qquad \forall j \in J, \forall s \in S, \forall h \in \{0, \dots, |T_s|\}, \tag{2.25}$$

$$y_{t_{h+1} j}^T = z_{b_h j}^B + z_{c_h j}^C, \qquad \forall j \in J, \forall s \in S, \forall h \in \{0, \dots, |T_s| - 1\}. \tag{2.26}$$

Analogously to tasks and timeslots, constraints (2.27)-(2.31) (constraints (2.32)-(2.36)) define the relation between variables that concern the starting, the end and the performing of break $b \in B$ (interruption $c \in C$).

$$y^B_{bj} \leq y^B_{b(j+1)}, \qquad \forall j \in J, \forall b \in B, \tag{2.27}$$
$$z^B_{bj} \leq z^B_{b(j+1)}, \qquad \forall j \in J, \forall b \in B, \tag{2.28}$$
$$z^B_{b(j+1)} \leq y^B_{bj}, \qquad \forall j \in J, \forall b \in B, \tag{2.29}$$
$$y^B_{bj} \leq x^B_b, \qquad \forall j \in J, \forall b \in B, \tag{2.30}$$
$$x^B_b \leq \sum_{j\in J}(y^B_{bj} - z^B_{bj}), \qquad \forall b \in B. \tag{2.31}$$

$$y^C_{cj} \leq y^C_{c(j+1)}, \qquad \forall j \in J, \forall c \in C, \tag{2.32}$$
$$z^C_{cj} \leq z^C_{c(j+1)}, \qquad \forall j \in J, \forall c \in C, \tag{2.33}$$
$$z^C_{c(j+1)} \leq y^C_{cj}, \qquad \forall j \in J, \forall c \in C, \tag{2.34}$$
$$y^C_{cj} \leq x^C_c, \qquad \forall j \in J, \forall c \in C, \tag{2.35}$$
$$x^C_c \leq \sum_{j\in J}(y^C_{cj} - z^C_{cj}), \qquad \forall c \in C. \tag{2.36}$$

As for each single activity, the duration of breaks and interruptions is bounded both from below and from above. Constraints (2.37) and (2.38) limit the duration of break b and interruption c, respectively (L3).

$$l_b x^B_b \leq \sum_{j\in J}(y^B_{bj} - z^B_{bj}) \leq u_b x^B_b, \qquad \forall b \in B, \tag{2.37}$$
$$l_c x^C_c \leq \sum_{j\in J}(y^C_{cj} - z^C_{cj}) \leq u_c x^C_c, \qquad \forall c \in C. \tag{2.38}$$

Daily shifts. The set of constraints presented in this paragraph allows to impose the legal regulations (L4) on the daily working hours, (L5) on the amplitude of the working day, and (L6) on the starting and finishing time.

Constraints (2.39)-(2.41) express the link between timeslots, breaks, interruptions and daily shifts. In particular, constraints (2.39) and (2.40) impose that daily shift s is performed, that is x^S_s is equal to 1, only if one of its timeslots is performed. Furthermore, daily shift s is performed in slot j, meaning that the difference $y^S_{sj} - z^S_{sj}$ is equal to 1, if either a timeslot t, a break b or an interruption c is performed in slot j. This is imposed by constraints (2.41).

$$x^S_s \leq \sum_{t\in T_s} x^T_t, \qquad \forall s \in S, \tag{2.39}$$
$$x^T_t \leq x^S_s, \qquad \forall s \in S, \forall t \in T_s, \tag{2.40}$$
$$\sum_{t\in T_s}(y^T_{tj} - z^T_{tj}) + \sum_{b\in B_s}(y^B_{bj} - z^B_{bj}) + \sum_{c\in C_s}(y^C_{cj} - z^C_{cj}) \leq y^S_{sj} - z^S_{sj}, \qquad \forall j \in J, \forall s \in S. \tag{2.41}$$

Relations between variables concerning starting y^S_{sj}, ending z^S_{sj} and performing x^S_s of daily shift s are imposed by constraints (2.42)-(2.46). The use of these variables, especially y^S_{sj} and

z^S_{sj}, is needed to define the slots b^S_s and f^S_s where daily shift s begins and ends, respectively.

$$y^S_{sj} \leq y^S_{s(j+1)}, \qquad \forall j \in J, \forall s \in S, \tag{2.42}$$
$$z^S_{sj} \leq z^S_{s(j+1)}, \qquad \forall j \in J, \forall s \in S, \tag{2.43}$$
$$z^S_{s(j+1)} \leq y^S_{sj}, \qquad \forall j \in J, \forall s \in S, \tag{2.44}$$
$$y^S_{sj} \leq x^S_s, \qquad \forall j \in J, \forall s \in S, \tag{2.45}$$
$$x^S_s \leq \sum_{j\in J}(y^S_{sj} - z^S_{sj}), \qquad \forall s \in S. \tag{2.46}$$

The first and the last slots of daily shift s are equal to variables b^S_s and f^S_s respectively. These variables allow to impose constraint (2.49) on the duration of s. This constraint corresponds to the legal regulation on the amplitude of a working day (L5).

$$b^S_s = |T|x^S_s - \sum_{j\in J} y^S_{sj}, \qquad \forall s \in S, \tag{2.47}$$
$$f^S_s = (|T| - 1)x^S_s - \sum_{j\in J} z^S_{sj}, \qquad \forall s \in S, \tag{2.48}$$
$$l_m x^S_s \leq f^S_s - b^S_s \leq u_m x^S_s, \qquad \forall s \in S. \tag{2.49}$$

Daily shift s can be forced to start after slot l^s_d (constraints (2.50)) and to end before u^s_d (constraints (2.51)) (L6).

$$b^S_s \geq l^s_d x^S_s, \qquad \forall d \in D, \forall s \in S_d, \tag{2.50}$$
$$f^S_s \leq u^s_d x^S_s, \qquad \forall d \in D, \forall s \in S_d. \tag{2.51}$$

However, it may happen that the starting slot of daily shift s is imposed to take value in a discrete set J_s. For instance, s can start either at 8am, or at 12pm, or at 2pm. This restriction cannot be imposed by constraints (2.50), and we make use of variables v^S_{sj}, which take value 1 if daily shift s starts in slot j. Constraints (2.52) and (2.53) impose that, if daily shift s is performed, it has to start in a slot that belongs to J_s.

$$x^S_s = \sum_{j\in J_s} v^S_{sj}, \qquad \forall s \in S, \tag{2.52}$$
$$b^S_s = \sum_{j\in J_s} j v^S_{sj}, \qquad \forall d \in D, \forall s \in S_d. \tag{2.53}$$

Constraints (2.54) bound the total daily working hours both from below and from above (L4).

$$l^d_w x^S_s \leq \sum_{t\in T_s} l^T_t \leq u^d_w x^S_s, \qquad \forall s \in S. \tag{2.54}$$

Schedule. The set of constraints presented in this paragraph allows to impose the legal regulations (L7) on the weekly working hours, (L8) on the consecutive working days, (L9) on the number of working days, and (L10) on the rest beetween consecutive daily shifts.

When combining daily shifts to build a complete schedule, we need to impose a rest period of duration greater than l_r^d between two consecutive shifts s and $s+1$ (constraints (2.55)). However, this rest period is not sufficient when daily shift s covers at least one of the night slots contained in N. For instance, we may have that all slots after 10pm are considered night slots. In this case, if a daily shift is worked after 10pm, a rest period of duration $l_r^n \geq l_r^d$ is required before performing the successive daily shift $s+1$ (constraints (2.56)) (L10).

$$l_r^d - (2 - x_s^S - x_{s+1}^S)M \leq b_{s+1}^S - f_s^S, \qquad \forall s \in S, \tag{2.55}$$

$$l_r^n - (3 - x_s^S - x_{s+1}^S - y_{sj}^S + z_{sj}^S)M \leq b_{s+1}^S - f_s^S, \qquad \forall j \in N, \forall s \in S. \tag{2.56}$$

Constraints (2.57) limit the total weekly working hours (L7), while constraints (2.58) bound the total number of working days (L9).

$$l_w^w \leq \sum_{s \in S_i} \sum_{t \in T_s} l_t^T \leq u_w^w \qquad \forall i \in I, \tag{2.57}$$

$$l_d^n \leq \sum_{s \in S_i} x_s^S \leq u_d^n \qquad \forall i \in I. \tag{2.58}$$

The number of consecutive working days is bounded from above by constraints (2.59), while it is bounded from below by constraints (2.60) (L8).

$$\sum_{s'=s}^{s+u_d^c} x_{s'}^S \leq u_d^c, \qquad \forall i \in I, \forall s \in S_i, \tag{2.59}$$

$$x_s^S + (w - \sum_{s'=s+1}^{s+w} x_{s'}^S) + x_{s+w+1}^S \geq 1, \qquad \forall i \in I, \forall s \in S_i, \forall w \in \{1, \ldots, l_d^c - 1\}. \tag{2.60}$$

The constraints just presented in Section 2.1.4 are usually imposed by contract regulations. However, they are not the only constraints contributing to the feasibility of the schedules. Other constraints concern, for instance, the pre-assignment to activities or the unavailability of employees. In the following sections, we give the formulation of all other constraints needed to define feasible schedules.

2.1.5 Activities cardinality constraints

Activities cardinality constraints (2.61) impose that employee i performs at most one activity in each slot j.

$$\sum_{a \in A} x_{ja}^i \leq 1, \qquad \forall i \in I, \forall j \in J. \tag{2.61}$$

2.1.6 Skills constraints

The problem considers a heterogeneous workforce, that is each employee has specific skills. Constraints (2.62) avoid that employees are assigned to activities for which they do not have competencies.

$$x^i_{ja} = 0, \qquad \forall i \in I, \forall j \in J, \forall a \notin A_i. \tag{2.62}$$

2.1.7 Pre-assignment constraints

Pre-assignment constraints (2.63) force an employee i to perform activity a in slot j. This situation can arise when the manager needs a particular employee to perform a specific activity, who may require peculiar skills. As a consequence, some assignments are fixed before scheduling all the employees and they are imposed to be carried out in the final planning.

$$x^i_{ja} = 1, \qquad \forall i \in I, \forall (j,a) \in R_i. \tag{2.63}$$

2.1.8 Unavailability constraints

Employees may not be available during some time periods. In order to prevent from scheduling employees during their unavailable slots, constraints (2.64) are imposed.

$$x^i_{ja} = 0, \qquad \forall i \in I, \forall a \in A, \forall j \in U_i. \tag{2.64}$$

2.1.9 Succession constraints

Not all sequences of activities are feasible and it may happens that one activity a' cannot be performed just after another activity a. All forbidden couples of sequences of activities are contained in set A_f and constraints (2.65) prevent from assigning forbidden sequences.

$$x^i_{ja} + x_{i(j+1)a'} \leq 1, \qquad \forall i \in I, \forall j \in J, \forall (a,a') \in A_f. \tag{2.65}$$

2.1.10 Distribution constraints

Distribution constraints specify the duration (2.66) and the number of times (2.67) that an activity can be performed each day. For instance, we want that activity a is not worked more than 3 hours each day, or we want that it is not performed more than 1 time per day.

$$l^a_d \leq \sum_{j \in J_d} x^i_{ja} \leq u^a_d, \qquad \forall i \in I, \forall a \in A, \forall d \in D, \tag{2.66}$$

$$l^a_n \leq \sum_{s \in S_d \cap S_i} \sum_{t \in T_s} \sum_{k \in K_{ta}} x^K_k \leq u^a_n, \qquad \forall i \in I, \forall a \in A, \forall d \in D. \tag{2.67}$$

2.2 Dantzig-Wolfe decomposition

The Dantzig-Wolfe decomposition (Dantzig and Wolfe (1960)) is well-known in personnel scheduling. The problem is decomposed in a master problem and multiple disjoint subproblems. The first combines feasible schedules to satisfy workload requirements, while minimizing the overall planning cost, given by under and over coverage penalizations, activities costs and transition costs. The second ones define feasible schedules (columns) taking into account the specific constraints of each set of homogeneous employees. The most common models proposed in literature are based on set covering (Côté et al. (2013)), or on a generalized version of set partitioning (Restrepo et al. (2016) and Gérard et al. (2016)). Our model falls in the last category, since both under and over coverage are allowed. We will refer to this model as a generalized set partitioning with convexity constraints and it will be treated in Chapter 3.

2.2.1 Master problem

Let us denote P^i the index set of all feasible schedules (columns) of employee i. Each column with index $p \in P^i$ is defined as a binary vector $(\delta^i_{pja})_{j\in J, a\in A}$, that satisfies constraints (2.3)-(2.67) concerning employee i. More in details, $\delta^i_{pja} \in \{0,1\}$ is equal to 1 if employee $i \in I$ performs activity $a \in A$ in slot $j \in J$, 0 otherwise. Furthermore, let us define binary variables x^i_p for each $p \in P^i$ and $i \in I$, where x^i_p is equal to 1 if schedule $(\delta^i_{pja})_{j,a}$ is selected for employee i, 0 otherwise. The cost c^i_p of each schedule considers both the cost c^i_{ja} of assigning activity a to employee i in slot j, and the cost c_t of changing activity within timeslot t assigned. Therefore, the cost c^i_p of column c is evaluated as follows:

$$c^i_p = \sum_{j\in J}\sum_{a\in A} c^i_{ja}\delta^i_{pja} + \sum_{s\in S_i}\sum_{t\in T_s} c_t n^T_{tp}, \qquad \forall i \in I, \forall p \in P^i, \tag{2.68}$$

where n^T_{tp} counts the number of transitions between activities in timeslot t, belonging to column p. The *master problem* can be stated as follows:

$$\min \quad \sum_{i\in I}\sum_{p\in P^i} c^i_p x^i_p + \sum_{j\in J}\sum_{a\in A}\left(\underline{c}_{ja}\underline{y}_{ja} + \bar{c}_{ja}\bar{y}_{ja}\right), \tag{2.69}$$

$$\text{s.t.} \quad \sum_{i\in I}\sum_{p\in P^i} \delta^i_{pja}x^i_p + \underline{y}_{ja} - \bar{y}_{ja} = b_{ja}, \qquad \forall j \in J, \forall a \in A, \tag{2.70}$$

$$\sum_{p\in P^i} x^i_p = 1, \qquad \forall i \in I, \tag{2.71}$$

$$x^i_p \in \{0,1\}, \qquad \forall p \in P^i, \forall i \in I, \tag{2.72}$$

$$\underline{y}_{ja}, \bar{y}_{ja} \geq 0, \qquad \forall j \in J, \forall a \in A. \tag{2.73}$$

The objective function (2.69) minimizes the total planning cost together with the under and over coverage costs. We recall that both activities and transitions costs are considered in c^i_p. Constraints (2.70) are called *linking constraints* and they impose the workload to be satisfied for each activity over the whole planning horizon. Each employee i needs to be assigned exactly to one schedule, and this is imposed by the *convexity constraints* (2.71). Finally, (2.72) and (2.73) define the variables domain. Problem (2.69)-(2.73) presents an exponential number of variables which corresponds to the number of feasible schedules. For this reason, techniques such as CG turn out to be suitable for solving this problem. CG is explained in the context of multi-activity tour scheduling in the next section.

2.2.2 Column generation.

CG is a mathematical programming technique that enables to solve a wide class of large linear problems by iteratively adding the variables of the model (Desrosiers and Lübbecke (2005)). Typically, only a reduced number of the variables is needed to prove optimality, which makes the technique interesting for problems with a huge number of variables. We employ CG to solve the linear relaxation of the master problem (2.69)-(2.73), obtained by relaxing the integrality constraints on binary variables x_p^i. CG replaces this linear relaxation by a restricted version, where only a subset of schedules $\tilde{P}^i \subset P^i$ of tractable size ($|\tilde{P}^i| \ll |P^i|$) is considered. The resulting problem is called *restricted master problem* (RMP) and it is stated as follows:

$$
\begin{aligned}
(RMP) \quad \min \quad & \sum_{i\in I}\sum_{p\in\tilde{P}^i} c_p^i x_p^i + \sum_{j\in J}\sum_{a\in A}\left(\underline{c}_{ja}\underline{y}_{ja} + \bar{c}_{ja}\bar{y}_{ja}\right), && && (2.74)\\
\text{s.t.} \quad & \sum_{i\in I}\sum_{p\in\tilde{P}^i} \delta_{pja}^i x_p^i + \underline{y}_{ja} - \bar{y}_{ja} = b_{ja}, && \forall j \in J, \forall a \in A, && (2.75)\\
& \sum_{p\in\tilde{P}^i} x_p^i = 1, && \forall i \in I, && (2.76)\\
& 0 \le x_p^i \le 1, && \forall p \in \tilde{P}^i, \forall i \in I, && (2.77)\\
& \underline{y}_{ja}, \bar{y}_{ja} \ge 0, && \forall j \in J, \forall a \in A. && (2.78)
\end{aligned}
$$

CG algorithm iteratively solves (RMP) on the reduced set of columns $\tilde{P}^i$ and updates $\tilde{P}^i$ with columns in $P^i \setminus \tilde{P}^i$. To explain which columns need to be added, we introduce the dual problem of (RMP):

$$
\begin{aligned}
(D) \quad \max \quad & \sum_{j\in J}\sum_{a\in A} b_{ja}u_{ja} + \sum_{i\in I} v_i, && && (2.79)\\
\text{s.t.} \quad & \sum_{j\in J}\sum_{a\in A} \delta_{pja}^i u_{ja} + v_i \le c_p^i, && \forall i \in I, \forall p \in \tilde{P}^i, && (2.80)\\
& -\underline{c}_{ja} \le u_{ja} \le \bar{c}_{ja}, && \forall j \in J, \forall a \in A, && (2.81)
\end{aligned}
$$

where $\{u_{ja}\}_{j\in J, a\in A}$ are the dual variables associated to the linking constraints (2.75), and $\{v_i\}_{i\in I}$ are the dual variables associated to the convexity constraints (2.76). Columns or variables x_p^i in (RMP) correspond to rows or constraints in the dual problem. The interesting columns $p \in P^i \setminus \tilde{P}^i$ to be added are those for which constraint (2.80) is violated by the current dual solution. Therefore, given the optimal dual solution (u, v), the following *pricing problems* are solved to identify the most violated constraints, or to prove that all these constraints are satisfied and optimality is achieved. For each employee $i \in I$, the corresponding pricing problem is stated as follows:

$$c^{i*} := \min_{p \in P^i} \quad c^i_p - \sum_{j \in J} \sum_{a \in A} \delta^i_{pja} u_{ja} - v_i. \tag{2.82}$$

In the original variables, the problem results in the following formulation:

$$\min \quad \sum_{j \in J} \sum_{a \in A} (c^i_{ja} - u_{ja}) x^i_{ja} - v_i + \sum_{s \in S_i} \sum_{t \in T_s} c_t n^T_t, \tag{2.83}$$

$$\text{s.t.} \quad (2.3) - (2.67). \tag{2.84}$$

The objective function evaluates the so called *reduced cost* of a column. According to the optimal values of problems (2.82), two different cases can be identified: if all columns have non-negative reduced costs, that is $c^{i*} \geq 0$ for all empoyees $i \in I$, then the optimal solution of (RMP) is also optimal for the master problem; otherwise, if there exists at least one column with negative reduced cost, that is $c^{i*} < 0$ for at least one $i \in I$, then the column derived from the optimal pricing solution is added to (RMP), which is solved again to obtain new dual variables. A well-known result about CG states that the algorithm is exact and terminates after a finite number of iterations, when $\bigcup_i P^i$ is a finite set as in our case. Indeed, since no variable in (RMP) has negative reduced cost, we have that each column can be generated and added at most once.

Unfortunately, CG algorithm suffers from several drawbacks. It may have a slow convergence that leads to the well-known tailing-off effect, the oscillations of the dual variables whose components jump from one extreme value to another and the degeneracy in the master problem that causes multiple optimal solutions in the dual. Finally, we have the fact that the first iterations do not generate useful columns due to the poor quality informations given by the dual variables. The reader is refered to Vanderbeck (2005) for a review on the main issues that arise when implementing a CG method. For dealing with these issues, we developped a dual ascent heuristic, that turns out to speed up CG limiting the dual variable oscillations. This method is described in Chapter 3.

2.3 Conclusion

In this chapter we presented a compact MILP formulation that models all constraints of the multi-activity tour scheduling problem addressed in this thesis. The problem is well suited for the Dantzig-Wolfe decomposition since it consists of disjoint subproblems that are linked by the workload constraints. However, the latter model presents an exponential number of variables, making techniques based on CG interesting for solving it. The next part of the thesis presents the methods used for solving the problem. In particular, we describe a dual ascent heuristic to speed up CG, an algorithm used for solving the pricing problem, and an exact B&P approach to obtain integer solutions. In addition, we present different heuristic methods to deal with real large-scale instances.

Part II

Exact method

Introduction to Part II

Part II of this thesis focuses on an exact B&P algorithm developed for solving the multi-activity tour scheduling problem presented in Chapter 1. The literature exhibits a wide range of models and solution techniques such as integer programming, CG, constraint programming, and decomposition methods. Concerning compact models, implicit and integer programming formulations using formal language have been proposed by Côté et al. (2011a), Côté et al. (2011b) and, more recently, Gérard et al. (2016). CG based on regular language have been employed by Demassey et al. (2005), extended in Demassey et al. (2006) who propose a B&P algorithm for obtaining integer solutions of the multi-activity shift scheduling. Later, a similar problem has been addressed through B&P by Côté et al. (2013) and Boyer et al. (2013). In the context of multi-activity tour scheduling, Restrepo et al. (2016) propose a daily-based Dantzig-Wolfe decomposition which is tackled with B&P. Recently, Restrepo et al. (2018) combines CG and Benders decomposition.

The proposed B&P algorithm differs from the previous either in the problem addressed, or in the Dantzig-Wolfe decomposition used. One of the key elements of every B&P is CG, which obtains a lower bound of the master problem at each node of the search tree. In order to do that, CG iteratively solves a reduced master problem and one or more pricing problems. Therefore, before describing the proposed B&P algorithm, we design our CG generation framework by showing how the reduced master problem and the pricing problems are solved. In particular, we develop a new dual ascent heuristic for the reduced master problem. Furthermore, for solving the pricing problems, we combine constraint and dynamic programming techniques in order to deal with the complex constraints that characterize the problem addressed in this thesis. Part II is organized as follows:

- Chapter 3 focuses on the resolution of the reduced master problem and presents a dual ascent heuristic to estimate dual variables in CG frameworks. In order to prove optimality, the procedure is combined with an LP solver. This algorithm is tested numerically not only on multi-activity tour scheduling, but also on problems from graph theory and random generated instances.
- Chapter 4 focuses on the resolution of the pricing problem and details the model and the algorithm used to generate minimum reduced cost schedules. The proposed method combines constraint programming and dynamic programming techniques.
- Chapter 5 presents a B&P algorithm for obtaining an optimal integer solutions of the problem. It makes use of the dual ascent heuristic and the pricing resolution method proposed in the previous chapters.

Chapter 3

Dual Ascent Heuristic

The Danztig-Wolfe decomposition (2.69)-(2.73) of the multi-activity tour scheduling problem appears in the form of a generalized set partitioning problem with convexity constraints. This model is usually encountered in CG algorithms, where the master problem involves set partitioning, set covering, set packing and additional convexity constraints. The purpose of this chapter is to present a dual ascent (DA) heuristic for evaluating dual variables of the linear relaxation of the generalized set partitioning model with convexity constraints.

Starting from Baldacci et al. (2008), the DA has shown to be very effective on different problems, helping to speed up the CG convergence and to reduce the dual variables oscillations. To the best of our knowledge, the DA has been applied to models involving only some of the variants and combinations of the classical set partitioning, covering and packing, which can be all included in the generalized set partitioning model. The DA proposed in this thesis is capable to deal with this generalized model. It is based on a parametric reformulation and it uses the Lagrangian relaxation and the subgradient method. To prove its validity it has been applied for solving different problems: the multi-activity tour scheduling, the minimum sum coloring, and some new generated instances.

The chapter is organized as follows: after the introduction in Section 3.1, Section 3.2 presents a brief review on the methods used to solve the set partitioning, set covering and set packing problems. Section 3.3 formally defines the generalized set partitioning problem with convexity, while Section 3.4 presents the DA heuristic. Classical Lagrangian relaxation is recalled in Section 3.5 and it is compared with DA. Different applications, together with some computational results are presented in Section 3.6 and Section 3.7. Conclusions in Section 3.8 close the chapter.

This paper has been submitted to *Discrete Optimization* in December 2017.

3.1 Introduction

The set partitioning, set covering and set packing problems are fundamental models in combinatorial optimization and they are concerned with finding an optimal family of subsets of elements from a set. They are formally presented as follows: suppose we are given a finite set $M = \{1, \dots, m\}$, and $\{M_j\}_{j \in N}$ are subsets of M, where $N = \{1, \dots, n\}$. Furthermore, let $A \in \{0,1\}^{m \times n}$ be the matrix of coefficients, $c \in \mathbb{R}^n$ the vector of costs, e the m-vector of all one and $x \in \{0,1\}^n$ the vector of decision variables. The set covering problem is to find a minimum cost cover of M, i.e. a collection $F \subseteq N$ such that $\bigcup_{j \in F} M_j = M$, $\{\min c^T x : Ax \geq e, x \in \{0,1\}^n\}$. The set packing problem is to find a maximum cost packing of M, i.e. a collection $F \subseteq N$ such that $M_j \cap M_k = \emptyset$ for all $j, k \in N$, $j \neq k$, $\{\max c^T x : Ax \leq e, x \in \{0,1\}^n\}$. Finally, the set partitioning problem is to find a minimum cost partition of M, i.e. a collection $F \subseteq N$ which is both a cover and a packing, $\{\min c^T x : Ax = e, x \in \{0,1\}^n\}$. A comprehensive survey on theory and applications of these three models is presented for example in Balas and Padberg (1976) and Vemuganti (1998). The set partitioning, covering and packing models are strictly related. Indeed, set partitioning can be brought to set covering and packing. Furthermore, set packing can be restated as a set partitioning (see Balas and Padberg (1976)). The three problems can be combined in a unified model, which aims to allow under (over) coverage, yielding set packing (covering). This model was first proposed by Darby-Dowman and Mitra (1985) and more recently by Rasmussen and Larsen (2011).

This chapter addresses the generalized version of the *unified set partitioning problem* with convexity constraints (i.e., constraints (3.3)), where the right hand side of coverage constraints (3.2) is allowed to be a positive integer vector $b \in \mathbb{Z}^m_+$. The problem is stated as follows:

$$\begin{aligned}
\min \quad & c^\top x + \underline{c}^\top \underline{y} + \bar{c}^\top \bar{y}, && (3.1)\\
\text{s.t.} \quad & Ax + \underline{y} - \bar{y} = b, && (3.2)\\
& Ex = e, && (3.3)\\
& \underline{y}, \bar{y} \geq 0, && (3.4)\\
& x \in \{0,1\}^n, && (3.5)
\end{aligned}$$

where $E \in \{0,1\}^{p \times n}$ is the coefficient matrix of the convexity constraints, $\underline{c}$ and $\bar{c}$ are the m-vectors of under and over coverage costs, and $\underline{y}$ and $\bar{y}$ are the m-vectors of decision variables controlling whether or not constraints (3.2) are respectively under or over covered. It is not difficult to see that model (3.1)-(3.5) catches set partitioning, covering and packing problems. In order to solve the set partitioning problem, we only need to set all components of vectors $\underline{c}$ and $\bar{c}$ equal to a sufficiently large positive number, thereby preventing both under and over coverage. In this case, if a feasible solution exists for set partitioning, the unified model will have the same optimal solution. Similarly, in order to solve set covering (resp. packing), we need to set all components of vector $\underline{c}$ ($\bar{c}$) equal to a large positive number, and $\bar{c}$ ($\underline{c}$) equal to 0. When set partitioning is used as a model, an exact cover may not exist or a solution with under and over coverage could be more interesting. However, this situation cannot be captured in the classical set partitioning, while the flexibility of the unified model allows deviations from an exact cover at a cost defined by the individual penalties, leading in some cases to better solutions. Model (3.1)-(3.5) is easily transformed in the unified set partitioning problem by removing constraints (3.3) and by setting vector b equal to 1.

The proposed method for solving the generalized set partitioning problem with convexity constraints (i.e., model (3.1)-(3.5)), usually encountered in CG algorithms, is a generalization of the DA procedure. The DA is based on a parametric reformulation and it uses Lagrangian relaxation and subgradient method. Its novelty consists in managing the generalized set partitioning constraints, where the right hand side can be different from the unit vector, and under-over coverage is allowed.

Two different applications are used to prove the validity of the proposed method. The first one is the multi-activity tour scheduling problem. Gérard et al. (2016) and Restrepo et al. (2016) propose two formulations based on the generalized set partitioning in order to take into account under and over coverage of the demand. The second application is the minimum sum coloring problem, which is a variant of the vertex coloring problem. A review of recent algorithms to solve the minimum sum coloring problem can be found in Jin et al. (2017). Recently, Furini et al. (2018) have proposed a set covering based formulation for this problem. Since none of these two applications consider together constraints of set partitioning, covering, packing and generalized set partitioning, the approach has been used also to solve new generated instances.

3.2 Literature review

Set partitioning (SPT), covering (SC) and packing (SP) have been used to model a great variety of problems in the literature. These models and their variants have been used to formulate many practical problems in different areas such as crew scheduling, cutting stock, facilities location, graphs coloring, personnel scheduling, vehicle routing problem, timetabling and many others. Below we report some examples for each type of problem. The list is limited since it is not our intent being exhaustive about the applications.

Set covering Problems where every customer is served by some location, vehicle or person often requires the set covering structure. Balas and Carrera (1996) formulate airline crew scheduling and bus driver scheduling using a SC model. Ceria et al. (1998a) propose a large-scale SC model for railways crew scheduling. Muter et al. (2010) make use of SC to model vehicle routing problem with time windows, while Malaguti et al. (2011) address the vertex coloring problem.

Set partitioning When every customer must be served exactly once, the problem takes the set partitioning structure. The vehicle routing problem and its variants widely use formulations based on the SPT model, originally proposed by Balinski and Quandt (1964). Among many different papers, we cite Baldacci et al. (2008), which address the capacitated vehicle routing problem. Desaulniers et al. (1997) use a SPT model to solve a crew scheduling problem for Air France. In Rezanova and Ryan (2010), a recovery problem for train driver duties is modeled as SPT. Brønmo et al. (2010) use SPT for a ship scheduling problem.

Set packing The goal of satisfying as much demand as possible, without creating conflicts, generally requires the set packing format. Rönnqvist (1995) propose a SP model for a cutting stock problem. Zwaneveld et al. (1996) formulate a railway feasibility problem as a SP. Mingozzi et al. (1998) used a SP formulation for a resource constrained project scheduling problem. Rossi and Smriglio (2001) considered a SP formulation for a ground holding problem. In Lusby et al. (2011) is given a survey of models and methods for railway track allocation, including formulations that rely on the SP model.

In the literature there are also papers addressing problems whose formulation combines partitioning, covering and packing constraints. Below we report a short and non exhaustive list of examples. Boschetti and Maniezzo (2015) use an extended covering formulation to model a city logistics problem, where covering constraints impose all clients to be served at least once, while the set packing-like constraints limit the number of vehicles available in each work shift. Baldacci et al. (2016) use a set partitioning based model for the vehicle routing problem with transhipment facilities, where clients have to be served exactly once, while facilities may be used or not. A very similar model combining partitioning and packing constraints is used by Baldacci et al. (2017) for the capacitated ring-star problem. Cacchiani et al. (2014) present a set covering based formulation for the periodic vehicle routing, where packing constraints limit the daily fleet size, while covering constraints define the relation between combinations and routes and ensure that at least one combination is selected for each client. All the examples reported above deal with routing problems, while, as far as we know, the generalized set partitioning problem has been addressed only in personnel scheduling problem (see Gérard et al. (2016) and Restrepo et al. (2016)).

Most of the models cited above have an exponential number of variables, since they are based on a Dantzig-Wolfe decomposition approach. Typically, only a small fraction of them is needed to prove optimality and this aspect makes CG an interesting technique. CG is an iterative process that solves a restricted master problem and one or several subproblems (cf. Section 2.2.2).

Primal or dual simplex methods are commonly used to solve the reduced master problem. Despite all the progress in linear programming, solving these linear programs can be a challenge. Various heuristics to obtain optimal and near optimal dual solutions have been proposed. Fisher and Kedia (1990) solve a mixed set covering-partitioning model using dual heuristics that include greedy and 3-opt heuristics and, in some cases, the subgradient method. It is applied to the dual of the linear relaxation to provide lower bounds for a B&B algorithm. Ceria et al. (1998b) propose a primal-dual Lagrangian heuristic that solves both the Lagrangian relaxations of the primal and dual problems simultaneously of the set covering problem. Then, primal and dual multipliers are used for fixing variables and reducing the problem. An extension of the subgradient method, called the volume algorithm, has been proposed by Barahona and Anbil (2002) to produce a valid lower bound as well as an approximation of the primal solution. More recently, Boschetti et al. (2008) presented both a dual ascent heuristic and an exact method for the set partitioning problem. The dual ascent heuristic makes use of parametric and Lagrangian relaxations to produce feasible dual solutions of the linear relaxation of the set partitioning problem. The exact method described uses the dual solution found by the heuristic to define a reduced problem with a limited subset of variables that is solved by an integer programming solver. The reduced problem is augmented until optimality can be proven. A similar heuristic approach has been later proposed by Boschetti and Maniezzo (2015) to solve a real-world city logistic problem, for which the reduced master problem consists of an extended set covering problem. An exact solution framework that employs dual ascent procedures was proposed by Baldacci et al. (2008). The method is used for the capacitated vehicle routing problem, but it can been tailored to solve several variants of the vehicle routing problem, as shown in Baldacci et al. (2010). Indeed, Baldacci et al. (2011a) address the pickup and delivery problem with time windows, Baldacci et al. (2011b) consider the periodic routing problem, while Baldacci et al. (2016) recently solve the vehicle routing problem with transhipment facilities. In the following we explain how to extend the dual ascent heuristic for solving the generalized set partitioning problem with convexity constraints.

3.3 Problem description

Let us consider the following *compact* mixed-integer linear program formulation representing the problem that we want to solve:

$$
\begin{aligned}
(P) \quad \min \quad & c^\top \tilde{x} + \underline{c}^\top \underline{y} + \bar{c}^\top \bar{y}, && (3.6)\\
\text{s.t.} \quad & \tilde{A}\tilde{x} + \underline{y} - \bar{y} = b, && (3.7)\\
& D\tilde{x} = d, && (3.8)\\
& \tilde{x} \in \{0,1\}^{\tilde{n}}, && (3.9)\\
& \underline{y}, \bar{y} \geq 0, && (3.10)
\end{aligned}
$$

where $c \in \mathbb{R}^{\tilde{n}}$, $\underline{c} \in \mathbb{R}^m$ and $\bar{c} \in \mathbb{R}^m$ are the vectors of costs, $\tilde{A} \in \{0,1\}^{m\times\tilde{n}}$ and $D \in \mathbb{R}^{l\times\tilde{n}}$ are the matrices of coefficients, $b \in \mathbb{Z}_+^m$ and $d \in \mathbb{R}^l$ are the right hand side vectors, $\tilde{x} \in \{0,1\}^{\tilde{n}}$, $\underline{y} \in \mathbb{R}_+^m$ and $\bar{y} \in \mathbb{R}_+^m$ are the decision variables. Problem (P) can be, for instance, the compact MILP model (2.1)-(2.67) presented in Section 2.1. Let

$$X = \{\tilde{x} : D\tilde{x} = d, \tilde{x} \in \{0,1\}^{\tilde{n}}\}$$

be the finite set defined by constraints (3.8) and (3.9). The model (3.6)-(3.10) can be reformulated by applying the classical Dantzig-Wolfe decomposition (DW), and the following extensive formulation is obtained:

$$
\begin{aligned}
(EP) \quad \min \quad & \sum_{p \in \tilde{P}} c_p x_p + \underline{c}^\top \underline{y} + \bar{c}^\top \bar{y}, && (3.11)\\
\text{s.t.} \quad & \sum_{p \in \tilde{P}} \delta_p x_p + \underline{y} - \bar{y} = b, && (3.12)\\
& \sum_{p \in \tilde{P}} x_p = 1, && (3.13)\\
& x \in \{0,1\}^n, && (3.14)\\
& \underline{y}, \bar{y} \geq 0, && (3.15)
\end{aligned}
$$

where $c_p = c^\top \xi_p$, $\delta_p = \tilde{A}\xi_p$ and ξ_p are the extreme points of X. Moreover, it is clear that if the constraints (3.8) have a block diagonal structure, X can be decomposed in k finite sets $X^1, \ldots, X^k$, i.e. $X = \bigcup_{1 \leq i \leq k} X^i$, where $X^i = \{\xi_p^i\}_{1 \leq p \leq \tilde{n}_i}$. We denote $I = \{1, \ldots, k\}$ the set of indices in the partition, $\tilde{P}^i = \{1, \ldots, \tilde{n}_i\}$ the set of indices of extreme points in each subset of the partition, and $M = \{1, \ldots, m\}$ is the index set of constraints in (3.7). The DW reformulation of problem (P) takes the following form:

$$
\begin{aligned}
(EP) \quad \min \quad & \sum_{i \in I} \sum_{p \in \tilde{P}^i} c_p^i x_p^i + \sum_{j \in M} \left(\underline{c}_j \underline{y}_j + \bar{c}_j \bar{y}_j \right), && && (3.16)\\
\text{s.t.} \quad & \sum_{i \in I} \sum_{p \in \tilde{P}^i} \delta_{pj}^i x_p^i + \underline{y}_j - \bar{y}_j = b_j, && \forall j \in M, && (3.17)\\
& \sum_{p \in \tilde{P}^i} x_p^i = 1, && \forall i \in I, && (3.18)\\
& x_p^i \in \{0,1\}, && \forall p \in \tilde{P}^i, \forall i \in I, && (3.19)\\
& \underline{y}_j, \bar{y}_j \geq 0, && \forall j \in M. && (3.20)
\end{aligned}
$$

We remark that model (3.16)-(3.20) corresponds to the DW decomposition (2.69)-(2.73) presented in Section 2.2. We assume that the decomposition results in zero-one coefficient matrix, i.e. $\delta_{pj}^i \in \{0,1\}$. The DW decomposition results in an extended formulation with an exponential number of variables that is often tackled by means of CG algorithms. CG is an iterative process that solves a restricted master problem (RMP) for obtaining each time dual variables. These latter are passed as objective function to the subproblems, which are solved looking for new variables to be added to the model when and if it is necessary. Therefore, the RMP must be solved several times quickly in order to update the dual variables passed to the subproblems.

Let us consider RMP, made up of subsets of columns $P^i \subseteq \tilde{P}^i$ for all i. The restricted master problem (RMP) appears as follows:

$$
\begin{aligned}
(RMP) \quad \min \quad & \sum_{i \in I} \sum_{p \in P^i} c_p^i x_p^i + \sum_{j \in M} \left(\underline{c}_j \underline{y}_j + \bar{c}_j \bar{y}_j \right), && && (3.21)\\
\text{s.t.} \quad & \sum_{i \in I} \sum_{p \in P_j^i} x_p^i + \underline{y}_j - \bar{y}_j = b_j, && \forall j \in M, && (3.22)\\
& \sum_{p \in P^i} x_p^i = 1, && \forall i \in I, && (3.23)\\
& x_p^i \geq 0, && \forall p \in P^i, \forall i \in I, && (3.24)\\
& \underline{y}_j, \bar{y}_j \geq 0, && \forall j \in M, && (3.25)
\end{aligned}
$$

where $P_j^i = \{p \in P^i : (\delta_p^i)_j = 1\}$ denotes the set of columns $p \in P^i$ that cover row $j \in M$. Problem (RMP) is the linear relaxation of the generalized set partitioning. It consists in selecting columns that satisfy coverage (3.22), minimizing the total cost (3.21) given by the columns and the under-over coverage. Constraints (3.23) state that each row i is covered by a convex combination of columns. Finally, constraints (3.24) and (3.25) define the domain of the variables. Let us denote $u = (u_1, \ldots, u_m)$ and $v = (v_1, \ldots, v_k)$ the vectors of the dual variables associated respectively with constraints (3.22) and (3.23). The k subproblems, solved to generate new variables, have the following formulation:

$$
\begin{aligned}
(SP^i) \quad c^{i*} &= \min c^{i\top} \tilde{x}^i - u^\top \tilde{A}^i \tilde{x}^i - v_i \\
\text{s.t.} \quad & \tilde{x}^i \in X^i. \qquad (3.26)
\end{aligned}
$$

The role of the subproblems is to provide columns that price out profitably or to prove that none of them exists and, therefore, optimality has been achieved.

3.4 A dual ascent heuristic

In this section we describe a dual ascent heuristic to compute efficient dual solutions of problem (RMP). The dual problem of (RMP) has the following formulation:

$$
\begin{aligned}
(D) \quad \max z_D = & \sum_{j \in M} b_j u_j + \sum_{i \in I} v_i, && && (3.27)\\
\text{s.t.} \quad & \sum_{j \in R_p^i} u_j + v_i \leq c_p^i, && \forall p \in P^i, \forall i \in I, && (3.28)\\
& -\bar{c}_j \leq u_j \leq \underline{c}_j, && \forall j \in M, && (3.29)
\end{aligned}
$$

where $R_p^i = \{j \in M : (\delta_p^i)_j = 1\}$ is the set of rows $j \in M$ covered by column $p \in P^i$. The dual ascent heuristic is based on a parametric reformulation of (RMP). Then, coverage and convexity constraints are relaxed by means of penalty vectors, to derive the Lagrangian relaxation. We explain in details these techniques respectively in Section 3.4.1 and Section 3.4.2.

3.4.1 Parametric reformulation

This section describes a parametric reformulation of problem (RMP). We associate to each variable x_p^i a set of $|R_p^i|+1$ binary variables z_p^i, z_p^h for all $h \in R_p^i$, according to the following expression:

$$x_p^i = \frac{1}{|R_p^i|+1}\Big(\sum_{h\in R_p^i} z_p^h + z_p^i\Big) \qquad \forall i \in I, p \in P^i. \tag{3.30}$$

Then, the problem obtained by applying expression (3.30) has the following formulation:

$$\min \sum_{i\in I}\sum_{p\in P^i} \frac{c_p^i}{|R_p^i|+1}\Big(\sum_{h\in R_p^i} z_p^h + z_p^i\Big) + \sum_{j\in M}\Big(\underline{c}_j\underline{y}_j + \bar{c}_j\bar{y}_j\Big), \tag{3.31}$$

$$\text{s.t.} \sum_{i\in I}\sum_{p\in P_j^i} \frac{1}{|R_p^i|+1}\Big(\sum_{h\in R_p^i} z_p^h + z_p^i\Big) + \underline{y}_j - \bar{y}_j = b_j, \qquad \forall j \in M, \tag{3.32}$$

$$\sum_{p\in P^i} \frac{1}{|R_p^i|+1}\Big(\sum_{h\in R_p^i} z_p^h + z_p^i\Big) = 1, \qquad \forall i \in I, \tag{3.33}$$

$$\underline{y}_j, \bar{y}_j \geq 0, \qquad \forall j \in M, \tag{3.34}$$

$$z_p^h, z_p^i \in \{0,1\}, \qquad \forall p \in P^i, \forall h \in R_p^i, \forall i \in I. \tag{3.35}$$

3.4.2 Lagrangian relaxation

Problem (3.31)-(3.35) is relaxed dualizing the linking constraints (3.32) and the convexity constraints (3.33) by means of penalty vectors $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^k$ respectively. Then, constraints (3.37) and (3.38) are added to improve the Lagrangian lower bound for given penalty vectors λ and μ. The Lagrangian subproblem $(LRP(\lambda,\mu))$ has the following formulation:

$$z_{LRP}(\lambda,\mu) = \min \sum_{j\in M}\Big(\sum_{i\in I}\sum_{p\in P_j^i} c_p^i(\lambda,\mu) z_p^j + \underline{c}_j(\lambda)\underline{y}_j + \bar{c}_j(\lambda)\bar{y}_j + b_j\lambda_j\Big) +$$
$$+ \sum_{i\in I}\Big(\sum_{p\in P^i} c_p^i(\lambda,\mu) z_p^i + \mu_i\Big) \tag{3.36}$$

$$\text{s.t.} \sum_{i\in I}\sum_{p\in P_j^i} z_p^j + \underline{y}_j - \bar{y}_j = b_j, \qquad \forall j \in M, \tag{3.37}$$

$$\sum_{p\in P^i} z_p^i = 1, \qquad \forall i \in I, \tag{3.38}$$

$$\underline{y}_j, \bar{y}_j \geq 0, \qquad \forall j \in M, \tag{3.39}$$

$$z_p^j, z_p^i \in \{0,1\}, \qquad \forall p \in P^i, \forall j \in R_p^i, \forall i \in I. \tag{3.40}$$

Note that the sum $\sum_{j\in M}\sum_{i\in I}\sum_{p\in P_j^i} z_p^j$ is obtained by rearranging the indices of the sum $\sum_{i\in I}\sum_{p\in P^i}\sum_{h\in R_p^i} z_p^h$. The Lagrangian costs $c_p^i(\lambda,\mu)$ are defined as

$$c_p^i(\lambda,\mu) := \frac{c_p^i - \sum_{j\in R_p^i}\lambda_j - \mu_i}{|R_p^i|+1},$$

while the coefficients $\underline{c}_j(\lambda)$ and $\bar{c}_j(\lambda)$ are respectively $\underline{c}_j - \lambda_j$ and $\bar{c}_j + \lambda_j$.

Problem $(LRP(\lambda,\mu))$ is decomposable into $m+k$ subproblems, one for each row $j \in M$, and one for each row $i \in I$. In the following, we show how a feasible solution of the subproblems can be defined. We first consider an index $i \in I$ and the corresponding subproblem $(LRP^i(\lambda,\mu))$,

$$z^i_{LRP}(\lambda,\mu) = \min \sum_{p\in P^i} c^i_p(\lambda,\mu) z^i_p + \mu_i \tag{3.41}$$

$$\text{s.t.} \sum_{p\in P^i} z^i_p = 1, \tag{3.42}$$

$$z^i_p \in \{0,1\}, \qquad \forall p \in P^i. \tag{3.43}$$

Let $p_i \in P$ be the column covering row i such that $p_i = \arg\min_{p\in P^i} c^i_p(\lambda,\mu)$. An optimal solution of problem $(LRP^i(\lambda,\mu))$ can be obtained setting $z^i_{p_i} = 1$ and $z^i_p = 0$ for all $p \in P^i \setminus \{p_i\}$. We now consider the subproblems $(LRP^j(\lambda,\mu))$ concerning index $j \in M$, which has the following formulation:

$$z^j_{LRP}(\lambda,\mu) = \min \sum_{i\in I}\sum_{p\in P^i_j} c^i_p(\lambda,\mu) z^j_p + \underline{c}_j(\lambda)\underline{y}_j + \bar{c}_j(\lambda)\bar{y}_j + b_j\lambda_j \tag{3.44}$$

$$\text{s.t.} \sum_{i\in I}\sum_{p\in P^i_j} z^j_p + \underline{y}_j - \bar{y}_j = b_j, \tag{3.45}$$

$$\underline{y}_j, \bar{y}_j \geq 0, \tag{3.46}$$

$$z^j_p \in \{0,1\}, \qquad \forall p \in P^i_j, \forall i \in I. \tag{3.47}$$

In order to find an optimal solution of problem $(LRP^j(\lambda,\mu))$, we consider the Lagrangian costs $c^i_p(\lambda,\mu)$ in ascending order $(c^i_{p_1}(\lambda,\mu) \leq \cdots \leq c^i_{p_{b_j}}(\lambda,\mu) \leq \dots)$ and we compare them with $\underline{c}_j(\lambda)$ and $-\bar{c}_j(\lambda)$. The following three different cases can be identified:

C1 : $-\bar{c}_j(\lambda) \leq c^i_{p_{b_j}}(\lambda,\mu) \leq \underline{c}_j(\lambda)$, an optimal solution is given by setting $z^j_p = 1$ for all indices $p = p_1, \dots, p_{b_j}$, while all under and over coverage variables are equal to 0:

$$\underline{y}_j = 0, \quad \bar{y}_j = 0, \quad z^j_p = \begin{cases} 1, & p = p_1, \dots, p_{b_j}, \\ 0, & \text{otherwise.} \end{cases}$$

C2 : $c^i_{p_l}(\lambda,\mu) \leq \underline{c}_j(\lambda) \leq c^i_{p_{b_j}}(\lambda,\mu)$, an optimal solution is given by setting $z^j_p = 1$ for all indices $p = p_1, \dots, p_l$. Furthermore we set under coverage variable $\underline{y}_j = b_j - l$, while over coverage variable equal to 0:

$$\underline{y}_j = b_j - l, \quad \bar{y}_j = 0, \quad z^j_p = \begin{cases} 1, & p = p_1, \dots, p_l, \\ 0, & \text{otherwise.} \end{cases}$$

C3 : $c^i_{p_{b_j}}(\lambda,\mu) \leq c^i_{p_l}(\lambda,\mu) \leq -\bar{c}_j(\lambda)$, an optimal solution is given by setting $z^j_p = 1$ for all indices $p = p_1, \dots, p_l$. Furthermore we set over coverage variable $\bar{y}_j = l - b_j$, while under coverage variable equal to 0:

$$\underline{y}_j = 0, \quad \bar{y}_j = l - b_j, \quad z_p^j = \begin{cases} 1, & p = p_1, \ldots, p_l, \\ 0, & \text{otherwise.} \end{cases}$$

The following Theorem 3.1 proves how a dual feasible solution (u, v) of cost $z_D \geq z_{LRP}(\lambda, \mu)$ can be obtained given any optimal solution of the Lagrangian subproblem $(LRP(\lambda, \mu))$.

Theorem 3.1. *Let us consider two vectors $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^k$. A feasible dual solution (u, v) of the dual problem (D) is given by the following expression:*

$$\begin{aligned} u_j &= \tilde{u}_j + \lambda_j, && \forall j \in M, \\ v_i &= \tilde{v}_i + \mu_i, && \forall i \in I. \end{aligned} \tag{3.48}$$

where

$$\tilde{u}_j = \max\Big\{-\bar{c}_j(\lambda), \min\{c^i_{p_{b_j}}(\lambda, \mu), \underline{c}_j(\lambda)\}\Big\},$$

and

$$\tilde{v}_i = c^i_{p_i}(\lambda, \mu) - \max_{p \in P^i}\Big\{\sum_{j \in R^i_p} (\tilde{u}_j - c^i_p(\lambda, \mu))^+\Big\}.$$

Furthermore, the cost z_D of this dual solution is greater than or equal to $z_{LRP}(\lambda, \mu)$.

Proof. We need to prove that (u, v), defined as in (3.48), is a feasible solution of problem (D). We can easily see that u satisfies the bound constraints (3.29). We now consider constraints (3.28): for each $i \in I$ and $p \in P^i$,

$$\begin{aligned} \sum_{j \in R^i_p} u_j + v_i &= \sum_{j \in R^i_p} (\tilde{u}_j + \lambda_j) + \tilde{v}_i + \mu_i \\ &= \sum_{i \in R^i_p} \tilde{u}_j + c^i_{p_i}(\lambda, \mu) - \max_{p' \in P^i}\Big\{\sum_{j \in R^i_{p'}} (\tilde{u}_j - c^i_{p'}(\lambda, \mu))^+\Big\} + \sum_{j \in R^i_p} \lambda_j + \mu_i \\ &\leq \sum_{j \in R^i_p} \tilde{u}_j + c^i_{p_i}(\lambda, \mu) - \sum_{j \in R^i_p} (\tilde{u}_j - c^i_p(\lambda, \mu))^+ + \sum_{j \in R^i_p} \lambda_j + \mu_i \\ &\leq \sum_{j \in R^i_p} \tilde{u}_j + c^i_{p_i}(\lambda, \mu) - \sum_{j \in R^i_p} (\tilde{u}_j - c^i_p(\lambda, \mu)) + \sum_{j \in R^i_p} \lambda_j + \mu_i \\ &\leq c^i_p(\lambda, \mu) + |R^i_p| c^i_p(\lambda, \mu) + \sum_{j \in R^i_p} \lambda_j + \mu_i \\ &= (|R^i_p| + 1) c^i_p(\lambda, \mu) + \sum_{j \in R^i_p} \lambda_j + \mu_i = c^i_p. \end{aligned}$$

We now show that the cost z_D of the dual solution defined with vectors λ and μ is greater than or equal to the Lagrangian cost $z_{LRP}(\lambda,\mu)$:

$$\begin{aligned}
z_D &= \sum_{j\in M} b_j u_j + \sum_{i\in I} v_i \\
&= \sum_{j\in M} b_j(\tilde{u}_j+\lambda_j) + \sum_{i\in I}\Big(c^i_{p_i}(\lambda,\mu) - \max_{p\in P^i}\Big\{\sum_{j\in R^i_p}(\tilde{u}_j - c^i_p(\lambda,\mu))^+\Big\} + \mu_i\Big) \\
&\geq \sum_{j\in M} b_j(\tilde{u}_j+\lambda_j) + \sum_{i\in I}(c^i_{p_i}(\lambda,\mu)+\mu_i) - \sum_{i\in I}\sum_{p\in P^i}\sum_{j\in R^i_p}(\tilde{u}_j - c^i_p(\lambda,\mu))^+ \\
&= \sum_{j\in M}\Big(b_j\tilde{u}_j - \sum_{i\in I}\sum_{p\in P^i_j}(\tilde{u}_j - c^i_p(\lambda,\mu))^+ + b_j\lambda_j\Big) + \sum_{i\in I}(c^i_{p_i}(\lambda,\mu)+\mu_i) \\
&= \sum_{j\in M} z^j_D + \sum_{i\in I} z^i_D.
\end{aligned}$$

It is easy to see that $z^i_D = z^i_{LRP}(\lambda,\mu)$. We now show that $z^j_D = z^j_{LRP}(\lambda,\mu)\ \forall j\in M$. We have three different cases, one for each primal solution defined in **C1**, **C2** or **C3**.

If the solution is the one defined in case **C1**:

$$\begin{aligned}
z^j_D &= b_j c^i_{p_{b_j}}(\lambda,\mu) - \sum_{p\in\{p_1,\ldots,p_{b_j}\}}(c^i_{p_{b_j}}(\lambda,\mu) - c^i_p(\lambda,\mu))^+ + b_j\lambda_j \\
&= \sum_{p\in\{p_1,\ldots,p_{b_j}\}} c^i_p(\lambda,\mu) + b_j\lambda_j = z^j_{LRP}(\lambda,\mu).
\end{aligned}$$

If the solution is the one defined in case **C2**:

$$\begin{aligned}
z^j_D &= b_j\underline{c}_j(\lambda) - \sum_{j\in\{p_1,\ldots,p_l\}}(\underline{c}_j(\lambda) - c^i_p(\lambda,\mu)) + b_j\lambda_j \\
&= \sum_{p\in\{p_1,\ldots,p_l\}} c^i_p(\lambda,\mu) + \underline{c}_j(\lambda)(b_j - l) + b_j\lambda_j = z^j_{LRP}(\lambda,\mu).
\end{aligned}$$

If the solution is the one defined in **C3**:

$$\begin{aligned}
z^j_D &= -b_j\bar{c}_j(\lambda) - \sum_{p\in\{p_1,\ldots,p_l\}}(-\bar{c}_j(\lambda) - c^i_p(\lambda,\mu)) + b_j\lambda_j \\
&= \sum_{p\in\{p_1,\ldots,p_l\}} c^i_p(\lambda,\mu) + \bar{c}_j(\lambda)(l - b_j) + b_j\lambda_j = z^j_{LRP}(\lambda,\mu).
\end{aligned}$$

We can conclude that $z_D \geq z_{LRP}(\lambda,\mu)$. □

The following Corollary states that maximizing the function $z_{LRP}(\lambda,\mu)$ with respect to λ and μ, we achieve the optimal dual value z^*_D.

Corollary 3.2. *The following equality holds:*

$$\max_{\lambda,\mu} z_{LRP}(\lambda,\mu) = z^*_D. \tag{3.49}$$

Proof. Let us consider the Lagrangian relaxation of problem (RMP), dualizing constraints (3.22) and (3.23) by means of penalty vectors $\lambda \in \mathbb{R}^m$ and $\mu \in \mathbb{R}^k$ respectively. We call the resulting problem $LR(\lambda, \mu)$,

$$\begin{aligned} z_{LR}(\lambda,\mu) = \min & \sum_{i\in I}\sum_{p\in P^i} \hat{c}_p^i(\lambda,\mu)x_p^i + \sum_{j\in M}\Big(\underline{c}_j(\lambda)\underline{y}_j + \bar{c}_j(\lambda)\bar{y}_j + b_j\lambda_j\Big) + \sum_{i\in I}\mu_i, && (3.50)\\ \text{s.t. } & \underline{y}_j, \bar{y}_j \geq 0, \qquad \forall j \in M, && (3.51)\\ & 0 \leq x_p \leq 1, \qquad \forall p \in P, && (3.52)\end{aligned}$$

where $\hat{c}_p^i(\lambda,\mu) := c_p^i - \sum_{i\in R_p^i}\lambda_j - \mu_i$. The goal is to show that the following inequality

$$z_{LR}(\lambda,\mu) \leq z_{LRP}(\lambda,\mu) \tag{3.53}$$

holds for each vectors λ and μ. It is easy to see that $z_{LR}(\lambda,\mu) = -\infty$ if $\underline{c}_j(\lambda)$ or $\bar{c}_j(\lambda)$ is negative for some index i. Since $z_{LRP}(\lambda,\mu)$ is always finite, inequality (3.53) holds.

We assume that both $\underline{c}_j(\lambda)$ and $\bar{c}_j(\lambda)$ are non negative. Furthermore, let us define the index set $\bar{P}^i = \{p \in P^i : \hat{c}_p^i(\lambda,\mu) < 0\}$ for each $i \in I$, then we have

$$z_{LR}(\lambda,\mu) = \sum_{p\in\bar{P}^i}\hat{c}_p^i(\lambda,\mu) + \sum_{j\in M} b_j\lambda_j + \sum_{i\in I}\mu_i.$$

Using the solution z of the problem $LRP(\lambda,\mu)$, we define the variable x_p according to expression (3.30). Finally, let $J^i = \{p \in P^i : x_p^i > 0\}$ and $\tilde{P}^i = \{p \in J^i : \hat{c}_p^i(\lambda,\mu) < 0\}$. Then we have:

$$\begin{aligned} z_{LRP}(\lambda,\mu) &= \sum_{i\in I}\sum_{p\in J^i}\hat{c}_p^i(\lambda,\mu)x_p^i + \sum_{j\in M}\Big(\underline{c}_j(\lambda)\underline{y}_j + \bar{c}_j(\lambda)\bar{y}_j + b_j\lambda_j\Big) + \sum_{i\in I}\mu_i\\ &\geq \sum_{i\in I}\sum_{p\in\tilde{P}^i}\hat{c}_p^i(\lambda,\mu)x_p^i + \sum_{j\in M} b_j\lambda_j + \sum_{i\in I}\mu_i\\ &\geq \sum_{i\in I}\sum_{p\in\bar{P}^i}\hat{c}_p^i(\lambda,\mu) + \sum_{j\in M} b_j\lambda_j + \sum_{i\in I}\mu_i = z_{LR}(\lambda,\mu).\end{aligned}$$

The first inequality comes from the fact that $\underline{c}_j(\lambda) \geq 0$, $\bar{c}_j(\lambda) \geq 0$ and $\hat{c}_p^i(\lambda,\mu)x_p^i \geq 0$ for each $p \in J^i \setminus \tilde{P}^i$. The second inequality comes from the fact that $x_p^i \leq 1$ and $\tilde{P}^i \subseteq \bar{P}^i$. Since the Lagrangian relaxation $LR(\lambda,\mu)$ has the integrality property, we have that the Lagrangian dual is equal to z_P. Therefore

$$\max_{\lambda,\mu} z_{LR}(\lambda,\mu) = \max_{\lambda,\mu} z_{LRP}(\lambda,\mu) = z_D^*.$$

□

From Corollary (3.2), it follows that the Lagrangian dual problem $\max_{\lambda,\mu} z_{LRP}(\lambda,\mu)$ need to be solved in order to find an optimal dual solution.

3.4.3 A column generation method based on dual ascent

In this section we describe a CG method to compute lower bounds of problems (RMP). This method differs from classical CG since it solves the reduced master problem by means of a dual ascent heuristic, instead of using the simplex algorithm (similar to Baldacci et al. (2016) and Baldacci et al. (2017)). However, in order to prove optimality, it needs to be combined with an LP solver. The proposed method is described as follows.

Step 1. *Initialization.* Initialize problem (RMP) with a set of columns P containing a feasible solution. Furthermore, initialize the penalty vectors $(\lambda, \mu) = (0, 0)$, $iter = 1$ and the parameter $\beta = 2$.

Step 2. *Dual ascent heuristic for finding a dual feasible solution* (u, v). Set $\bar{z}_{LRP}(\lambda, \mu) = -\infty$ and $iterDA = 1$. Perform the following steps:

Step 2a. *Solve* $LRP(\lambda, \mu)$. Using the current multipliers λ and μ, solve all subproblems (3.41)-(3.43) and (3.44)-(3.47) and get a solution of $LRP(\lambda, \mu)$. If $z_{LRP}(\lambda, \mu) > \bar{z}_{LRP}(\lambda, \mu)$, then update $\bar{z}_{LRP}(\lambda, \mu) = z_{LRP}(\lambda, \mu)$, update the dual solution (u, v) using expression (3.48), and set $\beta = \min\{2, 1.2 \times \beta\}$.

Step 2b. Update the multipliers (λ, μ) using the subgradient vectors $(g_j)_{j\in M}$ and $(g_i)_{i\in I}$ defined as $g_j = b_j - \sum_{i\in I}\sum_{p\in P^i_j} x^i_p - \underline{y}_j + \bar{y}_j$ and $g_i = 1 - \sum_{p\in P^i} x^i_p$. Modify the multipliers $\lambda_j = \lambda_j + \alpha g_j$ and $\mu_i = \mu_i + \alpha g_i$, where $\alpha = \beta(0.1 \times z_{LRP}(\lambda, \mu))/(\sum_{j\in M} g_j^2 + \sum_{i\in I} g_i^2)$.

Step 2c. Set $iterDA = iterDA + 1$. If after 2 consecutive iterations, $\bar{z}_{LRP}(\lambda, \mu)$ has not improved, halve β (i.e. $\beta = 0.5 \times \beta$). If $iterDA = MaxitDA(= 10)$ or $\beta < \beta min(= 0.0001)$, stop. Otherwise, return to Step 2a.

Step 3. *Generate new columns.* Generate, for each subproblem $i \in I$ (3.26), the column with minimum reduced cost c^{i*}. Define P^* the set of columns such that the reduced cost $c^{i*} < -0.1$.

Step 4. *Stopping criteria.* If $P^* = \emptyset$ or $iter = Maxit$, stop. Otherwise, update $P = P \cup P^*$ and return to Step 2.

We remark that $\bar{z}_{LRP}(\lambda, \mu)$ is a valid lower bound for problem (RMP), but it is not valid for the complete master problem. However, a valid lower bound is given by the Lagrangian dual bound (see Desrosiers and Lübbecke (2005)):

$$\sum_{j\in M} b_j u_j + \sum_{i\in I} (c^{i*} + v_i) \le z^*. \tag{3.54}$$

where we recall that c^{i*} is the minimum reduced costs obtained solving the subproblem (SP^i) during one iteration of CG, and z^* the optimal value of the linear relaxation of the master problem.

3.5 Classical Lagrangian relaxation

In this section we show how classical Lagrangian relaxation can be employed to solve problem (P), and it will be compared to the proposed DA.

The problem (P) can be also solved by applying the Lagrangian relaxation to the compact formulation, by dualizing exactly the coverage constraints (3.7), which are the linking constraints in the DW decomposition. The resulting subproblems are the same, and the columns generated by the Lagrangian subproblems can be added to the reduced master problem (Huisman et al. (2005)). The Lagrangian relaxation has the following formulation:

$$\begin{array}{llll} (CLR(\lambda)) & z_{CLR}(\lambda) = & \min & c^\top \tilde{x} + \underline{c}^\top \underline{y} + \bar{c}^\top \bar{y} + \lambda^\top (b - \tilde{A}\tilde{x} - \underline{y} + \bar{y}) \\ & & \text{s.t.} & \tilde{x} \in X, \\ & & & \underline{y}, \bar{y} \geq 0. \end{array}$$

We remark that problem $(CLR(\lambda))$ is unbounded if the Lagrangian cost vector $\underline{c} - \lambda$ or $\bar{c} + \lambda$ has at least one negative component. Therefore, we assume that both vectors are positive. As a consequence, in the optimal solution, vectors z and t are equal to 0. The resulting Lagrangian problem decomposes into p subproblems, one for each $k \in K$, due to the assumption that X is decomposable:

$$\begin{array}{llll} (CLR^i(\lambda)) & z^i_{CLR}(\lambda) = & \min & c^{i\top} \tilde{x}^i - \lambda^\top \tilde{A}^i \tilde{x}^i \\ & & \text{s.t.} & \tilde{x}^i \in X^i. \end{array} \tag{3.55}$$

We can see that the subproblem (3.26) of the DW decomposition and the subproblem of the Lagrangian relaxation (3.55) are identical, except for the constant term v_i in the objective function. Solution of the Lagrangian dual problem $\bar{z}_{CLR} = \max_\lambda z_{CLR}(\lambda)$ gives the maximum lower bound.

3.6 Applications

In this paragraph we show two problems whose DW decomposition leads to a particular case of (RMP). The first is the multi-activity tour scheduling problem. The second is the minimum sum coloring problem that consists in minimizing the sum of the cardinality of subsets of vertices receiving the same color, weighted with the index of the color, while ensuring that adjacent vertices receive different colors. These two applications do not consider together set partitioning, covering, packing and generalized set partitioning. For this reason, we generated further instances involving all four types of constraints.

3.6.1 Multi-activity tour scheduling

The DW decomposition of the multi-activity tour scheduling problem has been presented in Section 2.2. We can note that problem (2.69)-(2.73) is modeled as generalized set partitioning problem with convexity constraints. Indeed, the linking constraints (2.70) are expressed in a generalized set partitioning form since both under and over coverage are allowed, while (2.71) are the convexity constraints. According to the notation of this chapter, the extended formulation appears as follows:

$$\min \quad \sum_{i\in I}\sum_{p\in P^i} c_p^i x_p^i + \sum_{j\in M} \underline{c}_j \underline{y}_j + \sum_{j\in M} \overline{c}_j \overline{y}_j \tag{3.56}$$

$$\text{s.t.} \quad \sum_{i\in I}\sum_{p\in P_j^i} x_p^i + \underline{y}_j - \overline{y}_j = b_j, \qquad \forall j \in M, \tag{3.57}$$

$$\sum_{p\in P^i} x_p^i = 1, \qquad \forall i \in I, \tag{3.58}$$

$$x_p^i\{0,1\}, \qquad \forall p \in P^i, \forall i \in I, \tag{3.59}$$

$$\underline{y}_j, \overline{y}_j \geq 0, \qquad \forall j \in M, \tag{3.60}$$

where P^i contains feasible schedules of employee i and $P_j^i \subseteq P^i$ is the set of schedules in which employee i works during period j. Problem (3.56)-(3.60) assigns a feasible schedule to each employee (3.58), in order to cover the demand (3.57) while minimizing the total costs of the planning (3.56). By replacing constraints (3.59) with the following ones:

$$x_p^i \geq 0, \qquad \forall p \in P^i, \forall i \in I, \tag{3.61}$$

we obtain the linear relaxation of (3.56)-(3.60). The resulting problem is a particular case of (RMP). The subproblems, one for each employee i, generate new schedules:

$$\min \quad \sum_{j\in M} (c_j^i - u_j)\,\tilde{x}_j^i - v_i \tag{3.62}$$

$$\text{s.t.} \quad \tilde{x}^i \in X^i, \tag{3.63}$$

where X^i is the set of feasible schedules for employee i, and variables u and v denote the duals associated with constraints (3.57) and (3.58).

3.6.2 Minimum sum coloring

In the minimum sum coloring problem, we are given an undirected graph $G = (M, E)$ with $|M| = m$ vertices and $|E|$ edges. A coloring C of G is a partition of M into k stable sets $C = \{M^1, \dots, M^k\}$, where all the vertices in M^i are colored with the same color i. The sum coloring of C is given by the sum $\sum_{i=1,\dots,k}(i \cdot |M^i|)$. The minimum sum coloring problem consists of finding a coloring C that minimizes its sum coloring. Furini et al. (2018) introduce directly an extended formulation for this problem, without using DW decomposition. The model uses binary variables x_p^i associated with each stable set p and each color i and appears as follows:

$$\min \quad \sum_{i\in I}\sum_{p\in P^i} c_p^i x_p^i \tag{3.64}$$

$$\text{s.t.} \quad \sum_{i\in I}\sum_{p\in P_j^i} x_p^i \geq 1, \qquad \forall j \in M, \tag{3.65}$$

$$\sum_{p\in P^i} x_p^i \leq 1, \qquad \forall i \in I, \tag{3.66}$$

$$x_p^i \in \{0,1\}, \qquad \forall p \in P^i, \forall i \in I, \tag{3.67}$$

where P^i contains the stable sets colored with color i, $P_j^i \subseteq P^i$ contains the stable sets covering vertex j, and $c_p^i = i \cdot |M_p^i|$ is the cost of stable set p colored with color i. Constraints (3.65) impose each vertex j to be contained in at least one stable set, while constraints (3.66) impose each color i to be assigned to at most one stable set. The objective function (3.64) aims at finding a solution with the lowest sum coloring. We remark that constraints (3.66) can be rewritten using equalities, if we consider the empty stable set for each color i. By replacing constraints (3.67) with the following ones:

$$x_p^i \geq 0, \qquad \forall p \in P^i, \forall i \in I, \tag{3.68}$$

we obtain the linear relaxation of problem (3.64)-(3.67), which is a particular case of (RMP). It is sufficient to set all components of $\overline{c}$ equal to zero, and all components of $\underline{c}$ equal to a sufficient large positive number. The RMP (3.64)-(3.68) combines stable sets in order to cover all vertices, while the subproblems define as follows, one for each color i, generate new stable sets:

$$\min \quad \sum_{j\in M} i\, \tilde{x}_j^i - \sum_{j\in M} u_j\, \tilde{x}_j^i - v_k \tag{3.69}$$

$$\text{s.t.} \quad \tilde{x}_j^i + \tilde{x}_{j'}^i \leq 1, \qquad \forall (j,j') \in E, \tag{3.70}$$

$$\tilde{x}_j^i \in \{0,1\}, \qquad \forall j \in M, \tag{3.71}$$

where u denotes the dual variables associated with coverage constraints (3.65), while v denotes the dual variables associated with constraints (3.66). By changing the sign of all coefficients and the sense of the objective function, each subproblem becomes a maximum weight stable set problem on graph G, where the weight of each vertex $j \in M$ is defined as $u_j - i$.

3.7 Computational results

We present some computational results to show the performance of the dual ascent heuristic, by solving the linear relaxations of the problems presented above using CG. The LP solver used for the reduced master problem is CPLEX 12.7. Then, we combine it with the dual ascent heuristic previously presented, which is used during the first iterations of column generation. To be more precise, the dual ascent heuristic is used first and it stops as soon as the minimum reduced cost is greater than -0.1. The goal is to exploit the rapid decrease of the lower bound gap of the dual ascent heuristic, in order to speed up the convergence of the column generation. Finally, analogously to the dual ascent heuristic, we combine CPLEX with the classical Lagrangian relaxation. Experiments on the minimum sum coloring instances and on the generated instances presented above have been performed on a Intel Xeon E5-2650 v3 (2,3GHz), 64 GB of RAM (only one core is used), while an Intel Core i7-3770 CPU at 3,40GHz has been used for the experiments on the multi-activity tour scheduling problem.

3.7.1 Instances

The instances that have been used for testing the proposed approach are presented in the following paragraph. In particular, for the two applications, it was easy and possible to identify them. Moreover, we describe and report the algorithm used to generate the new complete instances.

Multi-activity tour scheduling instances. The multi-activity tour scheduling instances have been generated from real ones, given by the company Horizontal Software. The instances used for these tests are not the ones presented in Section 1.4. The reason relies on the fact that the set of instances Real are characterized by a considerable computational effort for solving the subproblems, as it will be shown in the next chapter. In order to evaluate the efficiency of DA, we decided to generate instances where the resolution of the subproblems does not affect strongly the total computational time of CG. The time horizon is fixed to one day and slots have time units of 1 hour or 30 minutes, resulting in instances with 24 and 48 slots. The instances consider 7 different activities, and 33 or 66 employees. They differ also in workload requirements, which are inspired by realistic demand coming from fast food restaurant chains. We consider 6 workload types named in alphabetic order, from A to F, with increasing demand. Instances are labeled with the format `S_E_W`, where `S`, `E` and `W` represent the number of slots, the number of employees and the workload type respectively.

Minimum sum coloring instances. We perform computational experiments on 43 benchmark instances, which are frequently used to evaluate the performance of minimum sum coloring algorithms (Jin et al. (2017)). These instances come from the COLOR 2002-2004 competitions[1].

Generated instances. We aim at further proving the validity of our approach by doing supplementary tests on more complete and diverse instances, involving set partitioning, covering, packing and generalized set partitioning constraints. They have been generated combining one instance from the covering data set ($\{$*rail507*, *rail516*, *rail582*$\}$), with one

[1] `http://mat.gsia.cmu.edu/COLOR02/`

instance from the partitioning data set ({*sppaa01*, *sppaa02*, *sppaa03*, *sppaa05*, *sppaa06*, *sppus03*, *sppus04*}). These data sets are available at the Beasley's OR-Library[2]. More in details, we proceeded as follows:

- we consider one instance I_1 $(m_1 \times n_1)$ from the set covering data set and one I_2 $(m_2 \times n_2)$ from the set partitioning data set;
- for each instance I_1 and I_2, we add a number of convexity constraints corresponding to 22% of the total number of rows. Therefore $i_1 = 0.22 \cdot m_1$, $i_2 = 0.22 \cdot m_2$ and the total number of convexity rows is $|I| = i_1 + i_2$;
- for each instance I_1 and I_2, each column has been duplicated respectively i_1 and i_2 times, and assigned to the different convexity rows;
- half randomly chosen rows M_1^1 from the set covering instance I_1 are defined as generalized set partitioning constraints, while the other half M_1^2 is kept as covering constraints. Analogously, half randomly chosen rows M_2^1 from the set partitioning instance I_2 are defined as packing constraints while the other half M_2^2 is kept as partitioning constraints;
- all columns costs c_j are equal either to 1 or to 2. The under and over assignment costs for the generalized set partitioning constraints are equal to 10. This means that $\underline{c}_j = \bar{c}_j = 10$ for all $j \in M_1^1$. All other under and over assignment costs are defined equal to 0 or to a large positive number (given by the sum of all columns' costs) depending whether they correspond to a partitioning, covering or packing constraints.

3.7.2 Algorithmic details

We provide here some details on the methods used to solve the subproblems, together with the techniques employed to speed up the convergence of the column generation.

Multi-activity tour scheduling problem The subproblems are solved with the method that will be presented in Chapter 4.

Minimum sum coloring problem The subproblems (3.69)-(3.71) are modeled as maximum weight stable set problems and are solved using the open-source implementation[3] of the B&B algorithm described in Held et al. (2012). In order to speed up the convergence of the CG, we apply a technique proposed by Furini et al. (2018): as soon as no negative reduced cost stable set is found for a color i such that the corresponding constraint is not active (< 1), then no subproblem $h > i$ is solved, due to the fact that no color $h > i$ can generate stable sets with negative reduced costs. This technique cannot be applied when the RMP is solved with the dual ascent heuristic, since no primal solution is produced.

[2] `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`
[3] `https://github.com/heldstephan/exactcolors`

Therefore, we apply a different procedure for speeding up the convergence of CG: as soon as no negative reduced cost stable set is found for color i such that the corresponding dual variable $v_i = 0$, we do not solve all the subproblems with color $h > i$ such that $v_h = 0$.

Generated instances Concerning the generated instances, we initialize the RMP selecting only a subset of the whole set of columns. In order to guarantee that a feasible initial solution satisfying all generalized set partitioning constraints exists, further columns with high costs are added. The subproblems consist simply in pricing out the remaining columns and selecting the one with the most negative reduced cost.

3.7.3 Discussion of the results

Tables 3.1, 3.2 and 3.3 compare the two described methods DA and CLR with the standard commercial software CPLEX.

For the minimum sum coloring problem, we report the name of the instance, the number of vertices n, the number of edges m and the density $d = 2m/n(n-1)$ of the graph, which gives an idea of the number of edges compared to the maximum number of edges that the graph can potentially have. The computational results show, for all three methods, the number of iterations (*iter*) of CG, the final number of columns in the RMP (*cols*) and the total computational time (*time*) in seconds. Instances unsolved within one hour time limit are marked with "tl" and "-". The results on the minimum sum coloring reported in Table 3.1, prove that CLR fails in solving 15 instances, while CPLEX fails in two of them. Furthermore, bold time values indicate that 31 instances are solved using DA in lower computational time compared to CPLEX. Finally, the average values on the last row, underline that DA needs lower number of iterations and computational time to converge. It also generates in average one third of the columns generated by CPLEX. The results on the multi-activity tour scheduling problem and on the generated instances in Table 3.2 and Table 3.3, confirm that DA improves the computational time, since for most of the instances the solving time is lower compared to CPLEX.

These experiments show us that the dual ascent heuristic presented in Section 3.4, embedded in a CG framework, improves the computational time for the majority of the instances. Indeed, CPLEX needs more than 55% of the time employed by DA on the minimum sum coloring, more than 35% on the multi-activity tour scheduling, and more than 42% on the generated instances. This is also supported by the fact that lower and upper bound gaps decrease faster when the dual ascent heuristic is employed in the first iterations.

We analyse in Figure 3.1, 3.2 and 3.3 the evolution of the lower bound gap, the upper bound gap and dual variables oscillations during the CG procedure. It is well known that the use of simplex method in CG causes several drawbacks, such as the dual oscillations and the tailing-off effect (Vanderbeck (2005)). Several stabilization techniques have been proposed to deal with these issues. Among them, we find smoothing techniques, where dual solutions used for pricing are corrected and combined with previous duals. Every time the reduced master problem is solved using CPLEX, we apply the smoothing technique

Instances					DA			CPLEX			CLR		
name	n	m	d	LB	iter	cols	time(s)	iter	cols	time(s)	iter	cols	time(s)
2-Insertions_3	37	72	0.11	62	87	273	**0.1**	102	828	0.2	308	936	0.3
3-Insertions_3	56	110	0.07	92	151	487	**0.8**	205	2022	1.1	323	996	1.3
anna	138	493	0.05	276	929	3953	**70.7**	989	10584	103.4	623	5040	**35.5**
david	87	406	0.11	237	169	1200	**2.9**	194	2844	3.6	407	3655	7.3
DSJC125.1	125	736	0.09	314	201	1100	3211.6	320	5601	3390.7	346	1773	**2976.4**
DSJC125.5	125	3891	0.50	978	140	2391	55.0	166	7143	**42.6**	471	7258	151.7
DSJC125.9	125	6961	0.90	2500	143	5627	**8.5**	136	9242	9.0	631	22260	35.6
DSJC250.5	250	15668	0.50	3105	225	6690	**2827.9**	362	29301	2974.0	-	-	tl
DSJC250.9	250	27897	0.90	8235	301	18114	**217.5**	304	38305	265.2	-	-	tl
DSJR500.1c	500	121275	0.97	16234	273	31349	**466.5**	399	82380	877.7	-	-	tl
games120	120	638	0.09	443	104	1087	1434.7	142	4529	**972.3**	-	-	tl
huck	74	301	0.11	243	78	663	**0.7**	111	1836	0.9	504	4281	3.8
jean	80	254	0.08	217	120	868	**0.9**	192	2158	2.0	428	3411	4.1
miles1000	128	3216	0.40	1666	81	2808	**7.8**	103	5574	10.0	562	18002	81.0
miles1500	128	5198	0.64	3354	88	4764	**4.8**	60	5916	5.9	797	38954	63.9
miles250	128	387	0.05	325	258	1950	352.4	336	5700	**242.3**	353	2566	554.7
miles500	128	1170	0.14	705	103	1745	41.3	143	4635	**34.9**	455	7590	213.4
miles750	128	2113	0.26	1173	82	2381	**6.9**	107	4922	9.1	502	12165	73.8
mug100_1	100	166	0.03	202	167	859	**1735.4**	244	4036	2353.2	-	-	tl
mug100_25	100	166	0.03	202	163	761	2971.9	254	4015	**1716.5**	-	-	tl
mug88_1	88	146	0.04	178	130	628	**308.9**	206	3179	389.6	335	1055	1494.2
mug88_25	88	146	0.04	178	156	964	**151.1**	197	3257	199.7	324	1055	632.1
mulsol.i.1	197	3925	0.20	1957	281	7433	**46.9**	2140	23950	669.9	-	-	tl
mulsol.i.2	188	3885	0.22	1191	2708	9417	910.0	1832	24879	**908.1**	-	-	tl
mulsol.i.3	184	3916	0.23	1187	946	5806	**185.4**	1667	23425	725.5	-	-	tl
mulsol.i.4	185	3946	0.23	1189	342	4436	**45.3**	1505	23186	625.1	-	-	tl
mulsol.i.5	186	3973	0.23	1160	1839	9367	**514.1**	1622	23460	709.6	-	-	tl
myciel3	11	20	0.36	21	23	61	**0.0**	19	74	0.0	257	774	0.0
myciel4	23	71	0.28	44	36	127	**0.0**	38	278	0.0	308	1131	0.1
myciel5	47	236	0.22	88	88	382	**0.2**	99	1197	0.4	280	1103	0.6
myciel6	95	755	0.17	176	173	901	**1.7**	265	3932	3.8	345	1442	2.7
myciel7	191	2360	0.13	349	580	3332	47.2	858	13482	115.9	422	2139	**23.2**
queen10_10	100	1470	0.30	550	146	1807	130.2	178	5175	**105.5**	540	5284	515.5
queen11_11	121	1980	0.27	726	157	2155	721.1	199	7366	**641.3**	566	5877	3213.4
queen5_5	25	160	0.53	75	50	283	**0.1**	41	364	0.1	297	1594	0.1
queen6_6	36	290	0.46	138	47	376	**0.1**	46	643	0.2	321	2347	0.6
queen7_7	49	476	0.40	196	62	564	**0.5**	84	1430	0.8	464	3132	2.5
queen8_12	96	1368	0.30	624	90	1333	52.2	107	4140	**45.3**	421	4406	427.4
queen8_8	64	728	0.36	291	75	797	2.2	84	2090	**1.7**	388	3321	12.7
queen9_9	81	1056	0.33	405	115	1271	**13.5**	146	3443	15.7	-	-	tl
zeroin.i.1	211	4100	0.19	1822	272	11617	**47.2**	3535	30763	2091.3	-	-	tl
zeroin.i.2	211	3541	0.16	1004	387	7295	**46.7**	-	-	tl	-	-	tl
zeroin.i.3	206	3540	0.17	998	267	6590	**36.4**	-	-	tl	-	-	tl
Average					**309**	**5953**	**488.9**	**489**	**14375**	**759.6**	**427**	**5840**	**1548.4**

Table 3.1: Results for minimum sum coloring problem with time limit 3600 seconds.

proposed by Neame (1999). The dual variables (u^t, v^t) used to generate new columns are obtained by taking a linear combination of the dual variables (u^{t-1}, v^{t-1}) of the previous iteration and the optimal dual solution (u^{*t}, v^{*t}) of the current restricted master problem. The subproblems are then solved using the smoothed duals. However, solving this modified subproblem might not yield a negative reduced cost column, even when one exists for the optimal duals. This situation is the result of a mis-pricing. In this case, the subproblem is solved again with a convex combination stepping closer to (u^{*t}, v^{*t}). In practice, it would be better to update the smoothed duals such that $(u^t, v^t) = (u^{*t}, v^{*t})$ after a small number of mis-pricing iterations. For this reason, we apply the scheme proposed by Pessoa et al. (2017), where the convex combination parameter is reduced during a mis-pricing sequence.

Figures 3.1, 3.2 and 3.3 present three graphics where: on the left, they show the behavior of the gap between the lower bound (3.54) and the optimal value z^* of the master problem. In the middle, they show the behavior of the gap between the upper bound, given by the

Instances	DA			CPLEX			CLR		
	iter	cols	time(s)	iter	cols	time(s)	iter	cols	time(s)
24_33_A	33	1084	**2.9**	40	1678	3.3	198	6826	14.2
24_33_B	36	980	**3.2**	37	1607	**3.2**	156	5039	11.9
24_33_C	35	1016	**3.2**	39	1709	3.3	167	5636	12.4
24_66_A	29	1772	**2.8**	43	3554	3.9	208	14312	15.0
24_66_B	43	1036	**4.3**	51	4138	4.8	193	13444	13.8
24_66_C	48	1174	4.9	41	3550	**4.1**	202	14044	14.9
24_66_D	25	2174	**2.5**	46	3824	4.3	238	16364	18.9
24_66_E	41	2032	4.1	41	3550	**4.0**	170	11272	14.0
24_66_F	31	2098	3.2	31	2890	**3.0**	80	4672	7.2
48_33_A	56	1557	**19.6**	86	3306	31.2	232	8124	75.8
48_33_B	71	1615	**25.3**	76	3034	27.4	230	5971	80.5
48_33_C	58	1628	**22.0**	78	3080	30.0	183	5753	61.9
48_66_A	48	3378	**19.0**	86	6612	32.1	198	14004	64.1
48_66_B	49	2900	**19.2**	83	6530	32.2	212	14120	70.2
48_66_C	53	3256	**20.7**	81	6358	30.9	232	16324	74.6
48_66_D	51	3460	**20.3**	89	6826	34.5	233	15868	76.6
48_66_E	75	3312	**33.1**	85	6678	35.0	208	12222	76.1
48_66_F	55	3256	**22.2**	71	5698	27.3	198	12100	70.9
Average	**46**	**2096**	**12.9**	**61**	**4145**	**17.5**	**196**	**10894**	**42.9**

Table 3.2: Results for multi-activity tour scheduling problem.

Instances	DA			CPLEX			CLR		
	iter	cols	time(s)	iter	cols	time(s)	iter	cols	time(s)
rail507_sppaa01	1099	268506	1190.1	1276	295902	1652.5	1126	296636	**1177.8**
rail507_sppaa02	780	144495	650.4	733	149499	683.4	654	145064	**52269**
rail507_sppaa03	962	246043	**959.5**	1321	307713	1528.2	1110	313128	1072.5
rail507_sppaa05	1060	254513	**1022.2**	1192	275412	1507.4	1152	277445	1080.0
rail507_sppaa06	934	195915	**756.3**	1017	210849	1171.9	995	227171	850.4
rail507_sppus03	609	67991	431.3	565	63886	541.4	589	66385	**400.1**
rail507_sppus04	649	75780	529.5	598	74178	**510.9**	653	79657	524.9
rail516_sppaa01	1037	249011	**855.2**	1357	311959	1241.8	1294	307295	1112.9
rail516_sppaa02	512	105526	**332.3**	647	140314	460.8	856	156552	493.7
rail516_sppaa03	966	236608	**770.7**	1306	298856	1392.5	1252	295032	954.8
rail516_sppaa05	966	231800	**766.2**	1283	300056	1294.2	1176	283701	913.3
rail516_sppaa06	786	173239	**547.1**	1033	218254	896.5	973	233490	711.4
rail516_sppus03	536	60639	**365.0**	565	65108	399.5	581	66523	391.4
rail516_sppus04	515	61854	**324.5**	509	66783	340.6	597	74838	357.4
rail582_sppaa01	854	245416	**958.0**	1103	306578	1436.3	1347	323769	1372.6
rail582_sppaa02	681	154775	**618.5**	828	199207	734.6	910	203656	822.8
rail582_sppaa03	949	253826	**1004.0**	1299	341777	1616.7	1286	351219	1368.5
rail582_sppaa05	879	238048	**921.9**	1143	295174	1358.1	1222	323971	1266.4
rail582_sppaa06	779	192349	**760.4**	1018	253960	1055.4	1313	273858	1197.3
rail582_sppus03	574	73645	**532.2**	675	87840	685.2	701	90531	642.9
rail582_sppus04	642	85719	**591.8**	761	107934	706.9	764	107239	691.9
Average	**798**	**172176**	**708.9**	**963**	**208154**	**1010.2**	**978**	**214150**	**853.6**

Table 3.3: Results for the generated instances.

optimal value of the reduced master problem, and the optimal value z^*. On the right, they show the distance between the dual variables at an intermediate iteration t, (u^t, v^t), and the final dual solution (u^*, v^*).

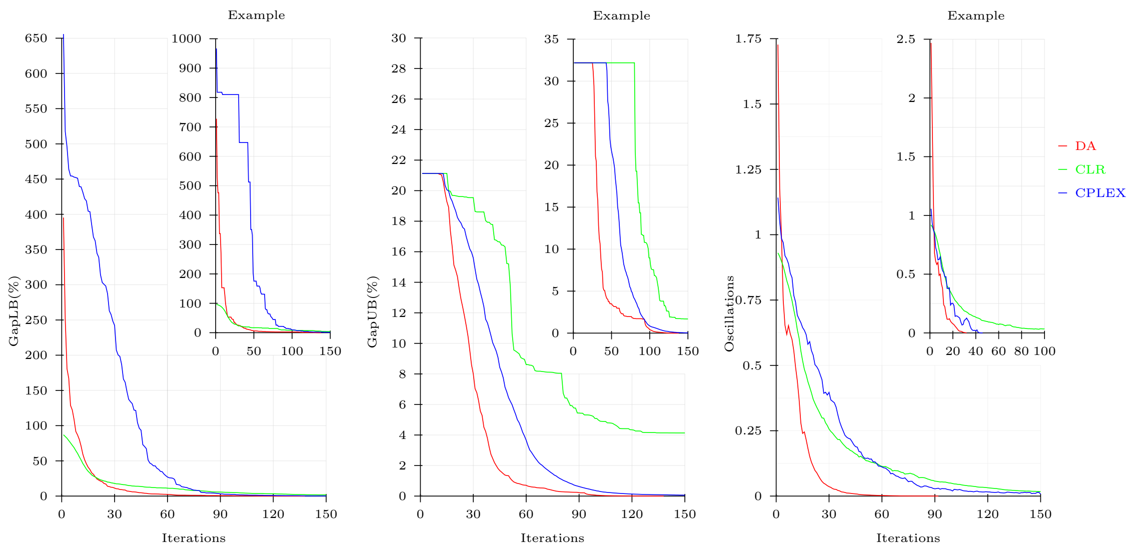


FIGURE 3.1: Lower bound gap, upper bound gap and dual variables oscillations for minimum sum coloring problem

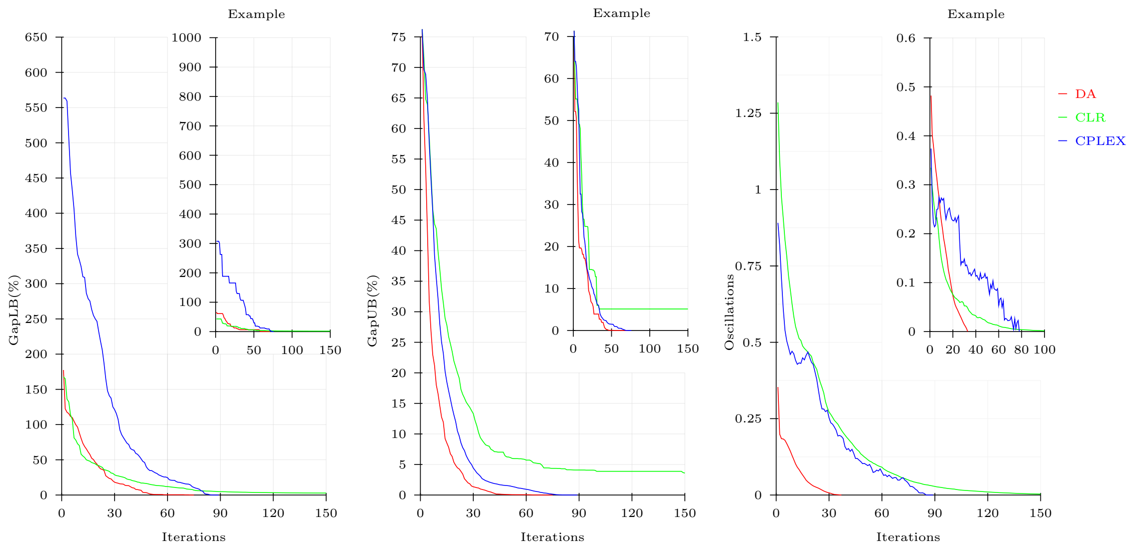


FIGURE 3.2: Lower bound gap, upper bound gap and dual variables oscillations for multi-activity tour scheduling problem

We can see for the three test problems that DA allows to decrease significantly the lower and upper bounds gaps during the first iterations of CG, yielding to a faster convergence compare with CLR and CPLEX. Algorithm CLR is competitive with DA in decreasing the lower bound gap, while the upper bound gap improves slowly. This behavior can be explained by the quality of the generated columns. Indeed, the dual variables estimated by DA allow the generation of good quality columns and a faster decrease of the upper bound gap. Furthermore, the dual variables generated are more stable compared to CPLEX. This

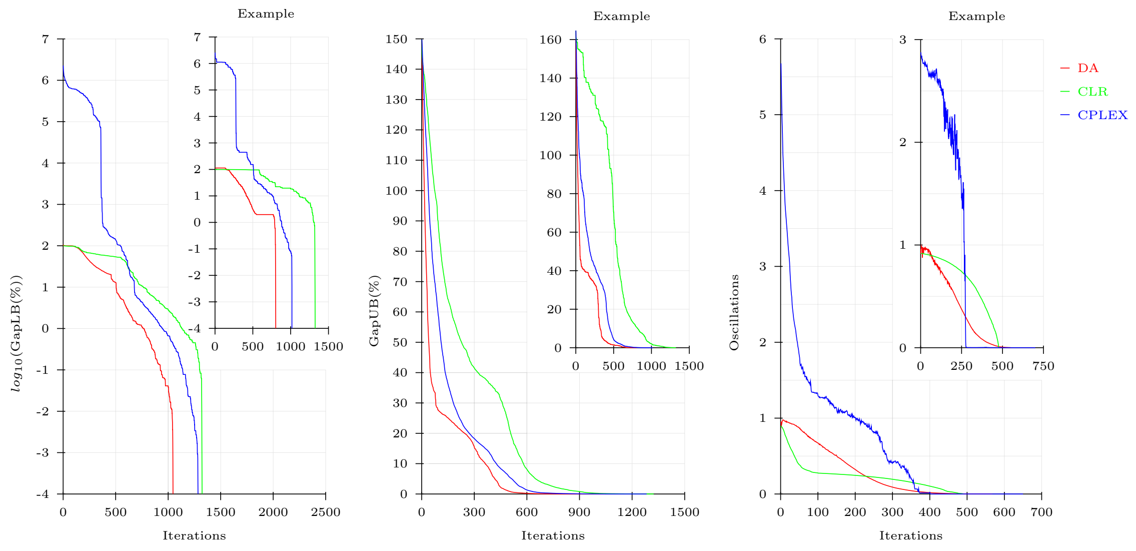


FIGURE 3.3: Lower bound gap, upper bound gap and dual variables oscillations for the generated instances

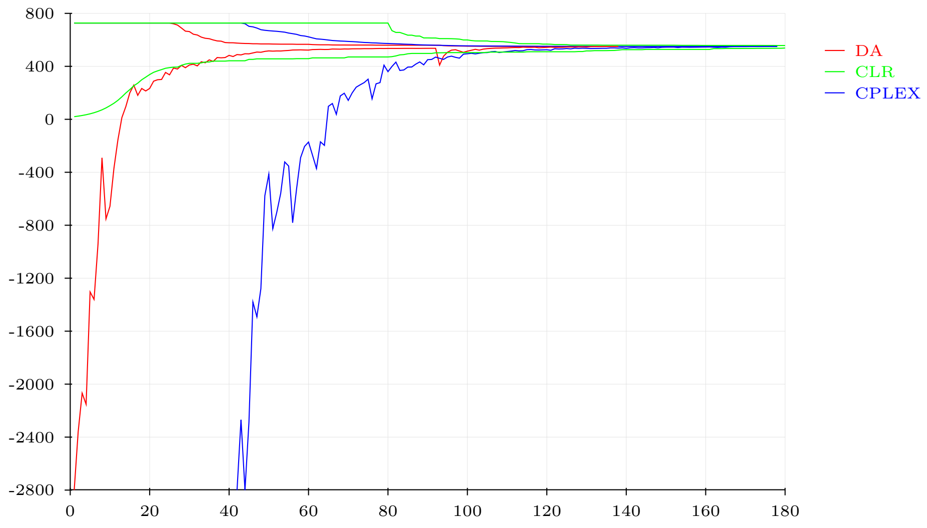


FIGURE 3.4: Lower and upper bounds for instance *queen10_10*.

behavior is marked in Figure 3.4, where we select one of the minimum sum coloring instance (*queen10_10*) and we show the evolution of the lower and upper bounds for DA, CPLEX and CLR. It is evident that the dual ascent heuristic helps to close faster the gap between the bounds. Moreover, it reduce the oscillations of the lower bounds.

3.8 Conclusions

This chapter describes a new dual ascent heuristic, based on parametric and Lagrangian relaxations, to obtain efficient dual feasible solutions of the generalized set partitioning problem with convexity constraints. The proposed method is able to deal with partitioning, covering, packing constraints and more general versions with right hand side different from one, together with under and over coverage variables. The computational experiments have been carried out on three different problem sets: the multi-activity tour scheduling, the minimum sum coloring, and some generated instances. The results indicate that the dual ascent heuristic is efficient and it can be integrated with methods based on simplex LP solver to speed up the convergence of CG.

The dual ascent heuristic will be embedded in a CG generation framework used to solve the multi-activity tour scheduling problem addressed in this thesis. In Chapter 4 we will present the resolution method employed to solve the pricing problem.

Chapter 4

Pricing Problem

This chapter describes how we can model and build feasible schedules to be assigned to the employees. We use regular language to model the complex constraints and we detail the algorithm employed to define feasible schedules. The proposed method models and solves the pricing problem in three phases, that reflect the layered structure of the schedule.

The construction of feasible schedules has been tackled in the literature with different methods. We find for instance MIP models (Côté et al. (2011b)), dynamic programming (Gérard et al. (2016)), context-free language (Côté et al. (2013) and Restrepo et al. (2016)). In this chapter we focus on regular language and dynamic programming. The first is used to construct feasible daily shifts, while the second is employed to build feasible schedules. A regular grammar is defined by a deterministic finite automaton (DFA), an easy and flexible tools able to capture the complex rules arising in multi-activity tour scheduling problems. Côté et al. (2011a) and Quimper and Rousseau (2009) have shown that context-free language is superior to regular language in defining daily shift rules. However, the method developed in this chapter builds daily shifts in two phases, and each phase employs an automaton that captures the corresponding rules. By doing so, we are able to limit the size of the extended graphs associated to the automata. In addition, preliminary experiments have shown that most of the computational effort is not employed to build the daily shifts, but in the construction of the complete schedules, which is done using dynamic programming. The use of DFAs does not seem to be a deficiency of the method presented in this chapter. For this reason and for their flexibility and easy handling, we decided to use DFAs to capture daily shifts rules.

The chapter is organized as follows: Section 4.1 introduces the basis of the theory of automata. Section 4.2 presents the model for timeslots, the daily shifts and schedules. The solving method is detailed in Section 4.3, while Section 4.5 shows extensive computational results. Conclusions are presented in Section 4.6.

4.1 Preliminaries

This section introduces the tools required for modeling feasible schedules. We recall the notions of deterministic finite automaton and regular languages. The interested reader is referred to Hopcroft et al. (2006) for further details.

4.1.1 Basic concepts

The theory of automata relies on three main concepts: the *alphabet*, the *strings* and the *language*. In the following we introduce the definitions of these terms.

Alphabet. The alphabet is a finite, nonempty set of symbols, conventionally denoted with the symbol Σ. For example, a common alphabet is $\Sigma = \{0, 1\}$, the binary alphabet.

String. A string (also called *word*) is a finite sequence of symbols chosen from an alphabet. For example, 101010 is a string from the binary alphabet. The string with zero occurrences of symbols is called *empty* string and it is denoted ϵ. The *length* of a string w is defined as the number of positions for symbols in the string and it is commonly denoted $|w|$. For instance, the string 101010 has length 6.

Language. Given an alphabet Σ, we define Σ^k as the set of strings of length k, each of whose symbol is in Σ. For example, if Σ is the binary alphabet, then $\Sigma^0 = \{\epsilon\}$, $\Sigma^1 = \{0, 1\}$, $\Sigma^2 = \{00, 01, 10, 11\}$ and so on. The set of all strings over an alphabet Σ is denoted Σ^* and is defined as $\Sigma^* = \bigcup_{k \in \mathbb{N}} \Sigma^k$. We can now give the definition of a language: if Σ is an alphabet, and $L \subseteq \Sigma^*$, then L is a language over Σ.

4.1.2 Deterministic finite automata

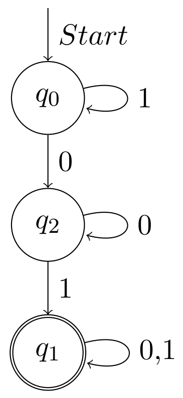
We introduce the formalism of a deterministic finite automaton, which is at the heart of the model of the scheduling rules. A *deterministic finite automaton* (DFA) consists of a five-tuple $A = (Q, \Sigma, \delta, q_0, F)$, where A is the name of the DFA, Q is a finite set of *states*, Σ is an alphabet containing the *input symbols*, $\delta : Q \times \Sigma \to Q$ is a *transition function* that takes as arguments a state and an input symbol and returns a state, $q_0 \in Q$ is the *initial state*, and $F \subseteq Q$ a set of *final* or *accepting states*. Defining a DFA by means of a five-tuple is not the only possible way. Indeed, there are two other notations for DFA presented in following two paragraphs: the *transition diagram* and the *transition table*. However, five-tuples and the transition diagrams will be used to describe DFAs all along this chapter.

Transition Diagrams. A transition diagram for a DFA $A = (Q, \Sigma, \delta, q_0, F)$ is defined as a graph where for each state in Q there is a node. Then, for each state $q \in Q$ and each symbol $a \in \Sigma$, if $\delta(q, a) = p$ there is an arc from node q to node p, labeled a. An arrow

$A = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{0, 1\}$
$F = \{q_1\}$
$\delta(q_0, 0) = q_2$
$\delta(q_0, 1) = q_0$
$\delta(q_1, 0) = q_1$
$\delta(q_1, 1) = q_1$
$\delta(q_2, 0) = q_2$
$\delta(q_2, 1) = q_1$

(A) Five-tuple



(B) Transition diagram

	0	1
$\rightarrow q_0$	q_2	q_0
$^*q_1$	q_1	q_1
q_2	q_2	q_1

(C) Transition table

FIGURE 4.1: Notations for a DFA (Hopcroft et al. (2006)).

labeled *Start* goes into the start node q_0. Finally, nodes corresponding to final states are marked by a double circle, while all the other states have a single circle.

Transition Tables. A transition table is a tabular representation of the function δ. The rows of the table correspond to the states, while the columns correspond to the symbols of the alphabet. The entry of row q and column a is the state $\delta(q, a)$.

Example 4.1. *In Figure 4.1 we show the three different notations for the DFA accepting only the strings of 0's and 1's that have the sequence 01 somewhere.*

The definition of a DFA formally states that there is a transition for any state on any input symbol, to exactly one state. However, it may be more convenient to design the DFA without some transitions, making such automaton a nondeterministic finite automaton (NFA). Indeed, an NFA is represented essentially like a DFA, with the only difference on the value returned by the transition function δ: a single state in the case of a DFA and a set of states in the case of an NFA (see Hopcroft et al. (2006)). Notice that the set of states the δ returns can be a singleton or the empty set. If we convert this NFA to a DFA, the automaton looks almost the same, but it includes one additional state, called *dead state*, and some additional arcs. Therefore, it is commonly accepted in literature to refer to an automaton as a DFA if it has at most one transition out of any state on any symbol, rather than if it has exactly one transition.

4.1.3 Extended transition function

We define an *extended transition function* $\hat{\delta}$ that describes what happens when we start in any state and follow any sequence of input symbols. $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ takes a state q and a string w and returns a state p. It is constructed from a transition function δ and it is defined by induction on the length of the input string w:

- the base step considers the empty string ϵ and defines $\hat{\delta}(q, \epsilon) = q$;

- the inductive step considers a generic string w of the form xa, where a is the last symbol of w and x is the substring without the last symbol. Then, $\hat{\delta}(q, w) = \delta(\hat{\delta}(q, x), a)$.

4.1.4 Regular language

After this introduction on DFA, we are now able to give the definition of regular language. Regular language, and therefore DFA, are the tools used all along this chapter to capture the complex rules arising from the multi-activity tour scheduling problem and to build feasible schedules. First, let us define the *language* of a DFA $A = (Q, \Sigma, \delta, q_0, F)$. This language is denoted $L(A)$ and it is stated as $L(A) = \{w \mid \hat{\delta}(q_0, w) \in F\}$. That is, the language of A is the set of strings w that take the initial state q_0 to one of the final states. If a given language L is $L(A)$ for some DFA A, then L is called a *regular language*.

4.2 Model

This section shows how regular languages can be used to model feasible schedules. We recall that a schedule is feasible if it satisfies all the rules stated in Section 1.2. Defining a DFA that consider at once the constraints describing a complete schedule, would generate an automaton with a very large number of states. Therefore, following the approach proposed by Gérard et al. (2016), we decide to decompose the problem of generating new schedules into subproblems. To be more precise, we first build feasible timeslots, then we combine timeslots with breaks and interruptions to get feasible daily shifts. Finally, daily shifts are combined into schedules. Gérard et al. (2016) develop a nested dynamic programming to solve the pricing problems. Their method decomposes the schedule into four levels: task, timeslot, daily shift and schedule. They solve the subproblems starting from the inner level (task), and going up to the outer level (schedule). We consider a similar hierarchical decomposition, but we use different resolution method based on constraint and dynamic programming at each level. In addition, besides break, we consider another type of pause which consists of the interruption. In the following we introduce the models used to define a feasible timeslot, a feasible daily shift and a feasible schedule.

4.2.1 DFA for timeslots

A timeslot is a sequence of activities performed consecutively without any break or interruption. Thus, the constraints defining its feasibility concern the following:

- employee's skills;
- employee's availability;
- forbidden succession of activities;
- minimum and maximum duration of activities;
- minimum and maximum consecutive working hours.

The DFA A accepting all feasible timeslots (strings) can be formally specified by defining the components of the five-tuple $A = (Q, \Sigma, \delta, q_0, F)$. First, its input alphabet Σ contains the different activities that the employee can perform, considering the skills. The state set Q contains the initial state q_0 and, for each activity a, it contains one state q_a^d, where d represents the duration and goes from 1 to u_a. We recall that u_a is the upper bound on the consecutive performance of activity a. The transition function δ is defined as follows: $\delta(q_0, a) = q_a^1$, meaning that the timeslot can start with any activity; $\delta(q_a^d, a) = q_a^{d+1}$ for every d from 1 to $u_a - 1$ in order to allow activity a to be performed until and no longer its maximum duration; $\delta(q_{a_1}^d, a_2) = q_{a_2}^1$ for all d from l_{a_1} to u_{a_1}, and for all activities a_2 different from a_1 that can be performed after a_1. This type of transition states that activity a_2 can start after activity a_1, as soon as a_1 has been done for at least its minimum duration l_{a_1}. The set of final states F consists of all states q_a^d for all d from l_a to u_a, meaning that the timeslot can terminate when the last activity performed satisfies the minimum and the maximum duration bounds.

The DFA A defined above captures the constraints concerning the employee's skills and availabilities, the succession between activities and their minimum and maximum durations. Therefore, A accepts all the timeslots satisfying these rules, including those violating the minimum and the maximum consecutive working hours. However, all the strings recognized by A, whose length is between the minimum and the maximum consecutive working hours, are feasible timeslots.

Example 4.2. *Let us consider timeslots with duration between* 6 *slots and* 8 *slots and three activities, denoted* a_1, a_2 *and* a_3. *The duration of these activities has to be respectively in the intervals* $[3, 4]$, $[2, 2]$ *and* $[2, 3]$. *Activity* a_1 *cannot follow* a_3 *and* a_3 *cannot follow* a_1, *while the other sequences of activities are allowed. The automaton* $A = (Q, \Sigma, \delta, q_0, F)$ *is defined as follows:*

- $Q = \{q_0, q_{a_1}^1, q_{a_1}^2, q_{a_1}^3, q_{a_1}^4, q_{a_2}^1, q_{a_2}^2, q_{a_3}^1, q_{a_3}^2, q_{a_3}^3\}$ *is the set of states;*
- $q_0 = 0$ *is the initial state;*
- $F = \{q_{a_1}^3, q_{a_1}^4, q_{a_2}^2, q_{a_3}^2, q_{a_3}^3\}$ *is the set of final states;*
- $\Sigma = \{a_1, a_2, a_3\}$ *is the alphabet;*
- δ *is the transition function defined above.*

We do not give the definition of transition function δ *because the detailed description is hard to read. Figure 4.2 shows the transition diagram which is easily readable.*

Strings recognized by A *with length equal to 6, 7 or 8 represent timeslots satisfying all the constraints. For readability, we avoid labeling all arcs and we assign color red, blue and green to arcs with label respectively* a_1, a_2 *and* a_3.

4.2.2 DFA for daily shifts

A daily shift consists of a sequence of timeslots separated by breaks and/or interruptions. The constraints defining its feasibility concern the following:

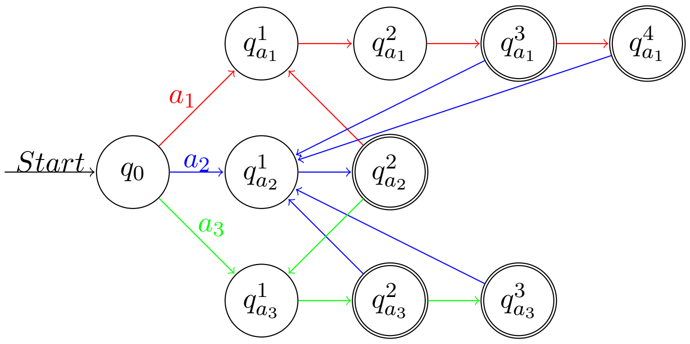


FIGURE 4.2: DFA for timeslots.

- number of breaks;
- number of interruptions;
- minimum and maximum duration of the break;
- minimum and maximum duration of the interruption;
- minimum and maximum daily working hours;
- minimum and maximum amplitude of the working day.

The configurations describing a feasible daily shift are limited to one, two or three timeslots divided by a pause (break or interruption). Therefore, we can list all the configurations within which a feasible daily shift falls:

(C1) timeslot (w);
(C2) timeslot + break + timeslot (wbw);
(C3) timeslot + interruption + timeslot (wiw);
(C4) timeslot + break + timeslot + interruption + timeslot ($wbwiw$);
(C5) timeslot + interruption + timeslot + break + timeslot ($wiwbw$);
(C6) timeslot + break + timeslot + break + timeslot ($wbwbw$);
(C7) timeslot + interruption + timeslot + interruption + timeslot ($wiwiw$);

It may be convenient to consider only a subset of configurations, since the legal constraints can forbid some of them. For instance, if the maximum number of interruptions per day is equal to 1, then configuration (C7) has to be deleted. Again, if we decide to consider daily shifts with only one or two timeslots, all configurations from (C4) to (C7) can be deleted. Therefore, three main DFA, based on the number of timeslots, can be built.

4.2.2.1 One timeslot.

The configuration considered is (C1), and the DFA $A = (Q, \Sigma, \delta, q_0, F)$ describing all feasible daily shifts is simple to define. The input alphabet Σ contains only the work activity, denoted w. The state set Q contains the initial state q_0 and one state q^d_w, for all d going from l^c_w to u^c_w, where d represents the duration of the timeslot, and l^c_w and u^c_w are respectively the minimum and the maximum bounds on the consecutive working hours. The transition function δ is defined as follows: $\delta(q_0, w) = q^d_w$, where $d = l^c_w$, states that a daily shift has to start with a timeslot of at least minimum duration, while $\delta(q_w^d, w) = q_w^{d+1}$, for all d from l_w^c to $u_w^c - 1$, states that the timeslot can have a duration between its minimum and maximum bounds. The set of final states F consists of all states q_w^d.

Example 4.3. *Let us consider daily shifts with only one timeslot, whose duration is between 6 and 8 slots. The automaton $A = (Q, \Sigma, \delta, q_0, F)$ is defined as follows:*

- $Q = \{q_0, q_w^6, q_w^7, q_w^8\}$ *is the set of states;*
- $\Sigma = \{w\}$ *is the alphabet;*
- δ *is the transition function defined above;*
- q_0 *is the initial state;*
- $F = \{q_w^6, q_w^7, q_w^8\}$ *is the set of final states.*

The transition diagram for the DFA accepting the feasible daily shifts with only one timeslot is defined in Figure 4.3.

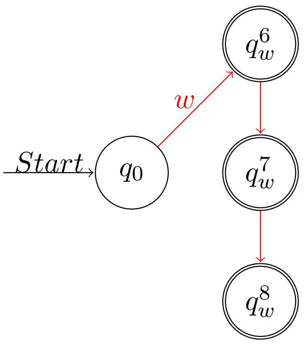


Figure 4.3: DFA for daily shift with one timeslot.

4.2.2.2 Two timeslots.

The configurations considered are (C2) and (C3). The DFA $A = (Q, \Sigma, \delta, q_0, F)$ describing all feasible daily shifts is defined by taking an input alphabet Σ that contains the work activity w, the break b and the interruption i. The state set Q contains the initial state q_0; states $q_{w_1}^d$ and $q_{w_2}^d$ represent the two timeslots, for all d from l_w^c to u_w^c; state q_b^d represents the break, for all d from l_b to u_b; state q_i^d represents the interruption, for all d from l_c to u_c. The transition function δ is defined as follows: $\delta(q_0, w) = q_{w_1}^d$, for d equal to l_w^c, states that the daily shift must start with a timeslot of at least minimum duration; $\delta(q_{w_1}^d, w) = q_{w_1}^{d+1}$ and $\delta(q_{w_2}^d, w) = q_{w_2}^{d+1}$, for d going from l_w^c to $u_w^c - 1$, $\delta(q_b^d, b) = q_b^{d+1}$, for d going from l_b to $u_b - 1$ and $\delta(q_i^d, i) = q_i^{d+1}$, for d going from l_c to $u_c - 1$, state that the timeslots, the break and the interruption have a duration between the corresponding minimum and maximum bounds; $\delta(q_{w1}^{d_w}, b) = q_b^{d_b}$ (resp. $\delta(q_{w1}^{d_w}, i) = q_i^{d_i}$), for d_w going from l_w^c to u_w^c and d_b equal to l_b (resp. d_i equal to l_c), imposes that a break (resp. interruption) can be assigned after a timeslot whose duration is between its minimum and maximum bounds; finally, $\delta(q_b^{d_b}, w) = q_{w2}^{d_w}$ (resp. $\delta(q_i^{d_i}, w) = q_{w2}^{d_w}$), for d_b going from l_b to u_b (resp. d_i going from l_c to u_c) and d_w equal to l_w^c, state that the second timeslot has to be assigned after a break (resp. an interruption). The set of final states F consists of $q_{w_2}^d$ for all d going from l_w^c to u_w^c.

Example 4.4. *Let us consider daily shifts with two timeslots whose duration is between* 6 *and* 8 *slots. Furthermore, the break has a fixed duration of* 1 *slot, while the interruption has a duration between* 2 *and* 3 *slots. The automaton* $A = (Q, \Sigma, \delta, q_0, F)$ *is defined as follows:*

- $Q = \{q_0, q^6_{w_1}, q^7_{w_1}, q^8_{w_1}, q^6_{w_2}, q^7_{w_2}, q^8_{w_2}\, q^1_b, q^2_i, q^3_i\}$ *is the set of states;*
- $\Sigma = \{w, b, i\}$ *is the alphabet;*
- δ *is the transition function defined above;*
- q_0 *is the initial state;*
- $F = \{q^6_{w_2}, q^7_{w_2}, q^8_{w_2}\}$ *is the set of final states.*

The transition diagram for the DFA accepting the feasible daily shifts with two timeslots is defined in Figure 4.4.

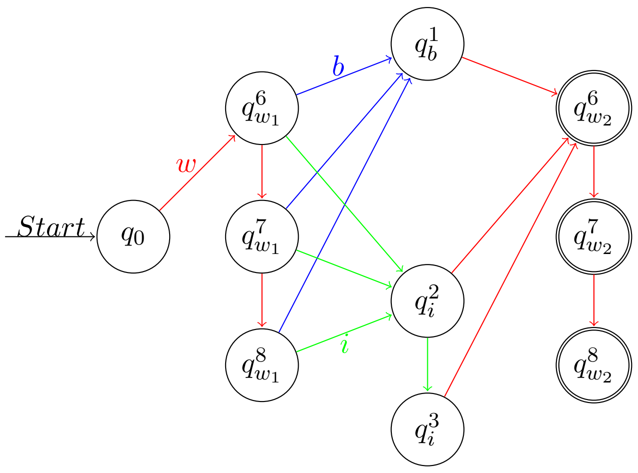


FIGURE 4.4: DFA for daily shift with two timeslots.

We remark that configuration (C1) can be easily integrated by adding, to the set of final states F, the states $q^d_{w_1}$ for all d going from l^c_w to u^c_w. In Example 4.4, we would have the $F = \{q^6_{w_1}, q^7_{w_1}, q^8_{w_1}, q^6_{w_2}, q^7_{w_2}, q^8_{w_2}\}$.

4.2.2.3 Three timeslots.

The configurations considered are (C4), (C5), (C6) and (C7). The DFA $A = (Q, \Sigma, \delta, q_0, F)$ describing all feasible daily shifts extends the DFA with two timeslots defined in Section 4.2.2.2. The input alphabet Σ contains the work activity w, the break b and the interruption i. The state set Q contains the initial state q_0; states $q^d_{w_1}$, $q^d_{w_2}$ and $q^d_{w_3}$ represent the three timeslots, for all d from l^c_w to u^c_w; states $q^d_{b_1}$ and $q^d_{b_2}$ represents the two breaks, for all d from l_b to u_b; states $q^d_{i_1}$ and $q^d_{i_2}$ represent the two interruptions, for all d from l_c to u_c. The transition function δ is defined as follows: $\delta(q_0, w) = q^d_{w_1}$, for d equal to l^c_w, states that the daily shift must start with a timeslot; the timeslots, the breaks and the interruptions have a duration between the corresponding minimum and maximum bounds, imposed by the transitions $\delta(q^d_{w_1}, w) = q^{d+1}_{w_1}$, $\delta(q^d_{w_2}, w) = q^{d+1}_{w_2}$ and $\delta(q^d_{w_2}, w) = q^{d+1}_{w_2}$, for d going from l^c_w to $u^c_w - 1$, $\delta(q^d_{b_1}, b) = q^{d+1}_{b_1}$ and $\delta(q^d_{b_2}, b) = q^{d+1}_{b_2}$, for d going from l_b to $u_b - 1$, $\delta(q^d_{i_1}, i) = q^{d+1}_{i_1}$

and $\delta(q^d_{i_2}, i) = q^{d+1}_{i_2}$, for d going from l_c to $u_c - 1$; $\delta(q^{d_w}_{w1}, b) = q^{d_b}_{b_1}$ and $\delta(q^{d_w}_{w2}, b) = q^{d_b}_{b_2}$ (resp. $\delta(q^{d_w}_{w1}, i) = q^{d_i}_{i_1}$ and $\delta(q^{d_w}_{w2}, i) = q^{d_i}_{i_2}$), for d_w going from l^c_w to u^c_w and d_b equal to l_b (resp. d_i equal to l_c), impose that a break (resp. interruption) can be assigned after a timeslot whose duration is between its minimum and maximum bounds; finally, $\delta(q^{d_b}_{b_1}, w) = q^{d_w}_{w2}$ and $\delta(q^{d_b}_{b_2}, w) = q^{d_w}_{w3}$ (resp. $\delta(q^{d_i}_{i_1}, w) = q^{d_w}_{w2}$ and $\delta(q^{d_i}_{i_2}, w) = q^{d_w}_{w3}$), for d_b going from l_b to u_b (resp. d_i going from l_c to u_c) and d_w equal to l^c_w, state that the second and third timeslots have to be assigned after a break (resp. an interruption). The set of final states F consists of $q^d_{w_3}$ for all d going from l^c_w to u^c_w.

Example 4.5. *Let us consider daily shifts with three timeslots whose duration is between* 6 *and* 8 *slots. Furthermore, the break has a fixed duration of* 1 *slot, while the interruption has a duration between* 2 *and* 3 *slots. The automaton* $A = (Q, \Sigma, \delta, q_0, F)$ *is defined as follows:*

- $Q = \{q_0, q^6_{w_1}, q^7_{w_1}, q^8_{w_1}, q^6_{w_2}, q^7_{w_2}, q^8_{w_2}\, q^6_{w_3}, q^7_{w_3}, q^8_{w_3}, q^1_{b_1}, q^1_{b_2}, q^2_{i_1}, q^3_{i_1}, q^2_{i_2}, q^3_{i_2}\}$ *is the set of states;*
- $\Sigma = \{w, b, i\}$ *is the alphabet;*
- δ *is the transition function defined above;*
- q_0 *is the initial state;*
- $F = \{q^6_{w_3}, q^7_{w_3}, q^8_{w_3}\}$ *is the set of final states.*

The transition diagram for the DFA accepting the feasible daily shifts with three timeslots is defined in Figure 4.5.

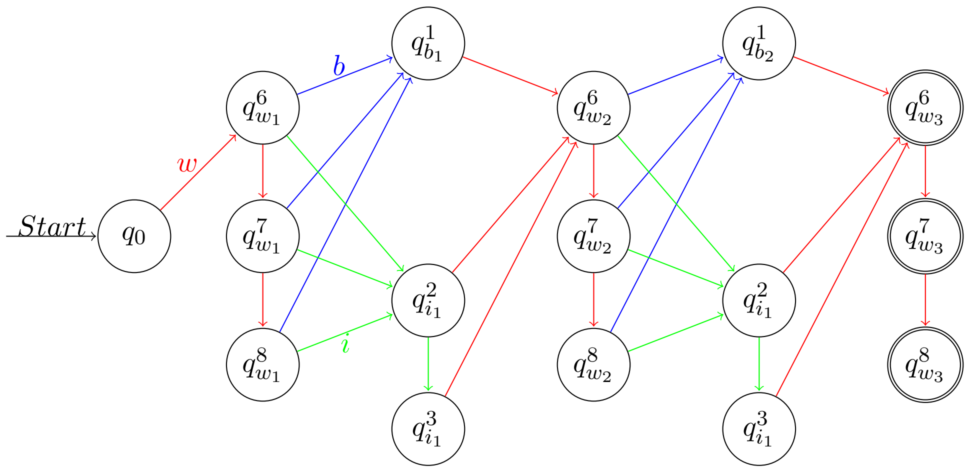


FIGURE 4.5: DFA for daily shift with three timeslots.

As previous, we remark that configuration (C1) can be easily integrated by adding the states $q^d_{w_1}$, for all d going from l^c_w to u^c_w, to the set of final states F. Furthermore, configurations (C2) and (C3) can be integrated by adding states $q^d_{w_2}$, for all d going from l^c_w to u^c_w. We have already seen that the legal constraints may exclude some possible configurations for a daily shift. For instance, suppose that the feasible daily shifts contain three timeslots, but the maximum number of breaks and the maximum number of interruptions are both equal to 1. This means that configurations (C6) and (C7) are not feasible. The DFA previously defined (see the example in Figure 4.5) does not reject strings of type $wbwbw$ and $wiwiw$. To this purpose, states $q^d_{w_2}$ need to be duplicated in order to keep track if the second timeslot has started after a break or an interruption. We do not redefine the DFA A, but we show in Figure 4.6 how this restriction can be applied to Example 4.5:

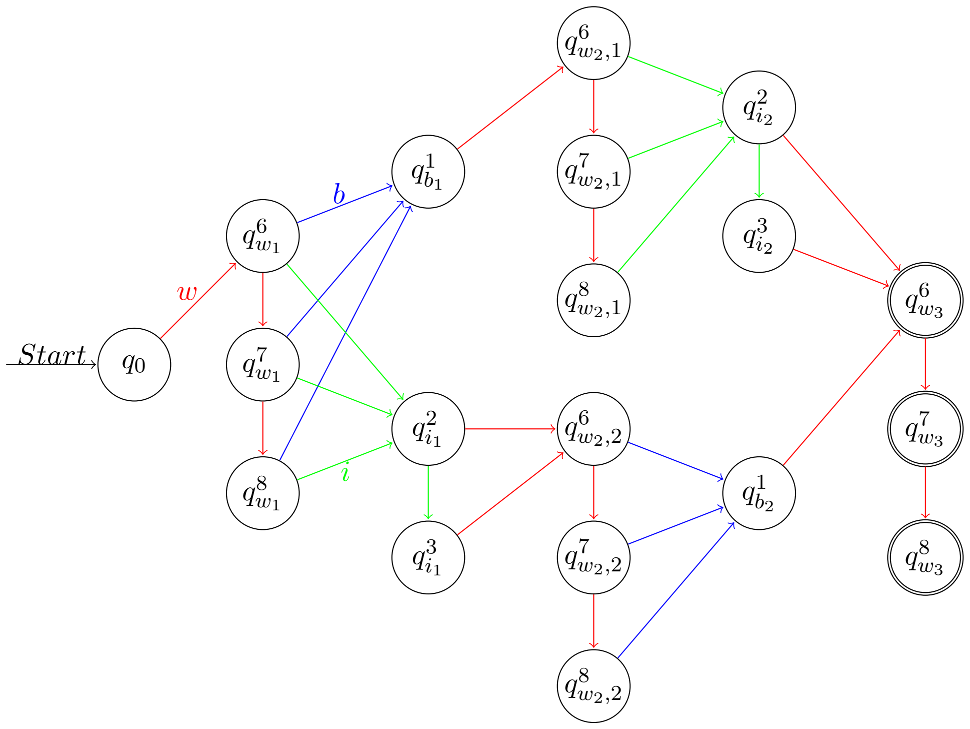


FIGURE 4.6: DFA for daily shift with three timeslots.

Remark. All DFA defined in Section 4.2.2 accept not only feasible daily shifts, but also daily shifts not satisfying the constraint on the amplitude of the working day. This is due to the fact that the transition $\delta(q_0, w) = q^d_{w_1}$, with d equal to l^c_w, means that the daily shift starts with a timeslot of minimum duration l^c_w. Analogously, this happens with the transition to a break $\delta(q^d_{w_1}, b) = q^{d_b}_{b_1}$, with d_b equal to l_b, and the transition to an interruption $\delta(q^d_{w_1}, i) = q^{d_i}_{i_1}$, with d_i equal to l_c. As consequence, a string of length 5, does not correspond to a daily shift of 5 slots. For instance, let us consider the DFA in Figure 4.6, the string $wwbwiw$ has length 6 but it does not describe a daily shift with amplitude 6 slots. Indeed, this daily shift consists of three timeslots of respectively 7, 6 and 6 slots, divided by a break of 1 slot and an interruption of 2 slots. Therefore, its total amplitude is 22 slots. Similarly, the DFA does not capture the constraint on the total daily working hours. Therefore, the DFA defined in this section can capture the constraints on the number and the duration of both breaks and interruptions, but not the constraints on the amplitude of the working day and the total daily working hours. Section 4.3 will show how to integrate them.

4.2.3 Directed acyclic graph for schedules

A schedule consists of a sequence of daily shifts and days-off covering the whole time horizon, fixed to one week in our problem. The constraints defining its feasibility concern the following:

- minimum and maximum weekly working hours;
- minimum and maximum number of daily shifts;
- minimum and maximum consecutive daily shifts;
- rest between two consecutive daily shifts.

We make use of a directed acyclic graph (DAG) $G = (V, E)$ to build feasible schedules, where V and E are respectively the set of nodes and the set of arcs. G is a layered graph where each layer corresponds to a single day d, and contains one node v_w^d for each feasible daily shift w (working day) associated to day d. Furthermore, each layer contains one node v_r^d for the day-off r (rest day). Finally, V contains two additional nodes: the source s and the sink t. The set of arcs is divided into four types: arcs (s, v_w^1) (resp. (s, r^1)) going from the source node to a daily shift (resp. day-off) node of the first day; arcs (v_w^7, t) (resp. (r^7, t)) going from a daily shift (resp. day-off) node of the last day to the sink node; arcs (r^d, v_w^{d+1}) and (v_w^d, r^{d+1}) connecting a day-off with daily shift of two consecutive days; arcs $(v_w^d, v_{w'}^{d+1})$ connecting two consecutive daily shifts. While the first three types of arcs exist for all daily shifts w, the last type exists only for tuples (w, w') such that the constraint concerning the rest of two consecutive daily shifts is satisfied.

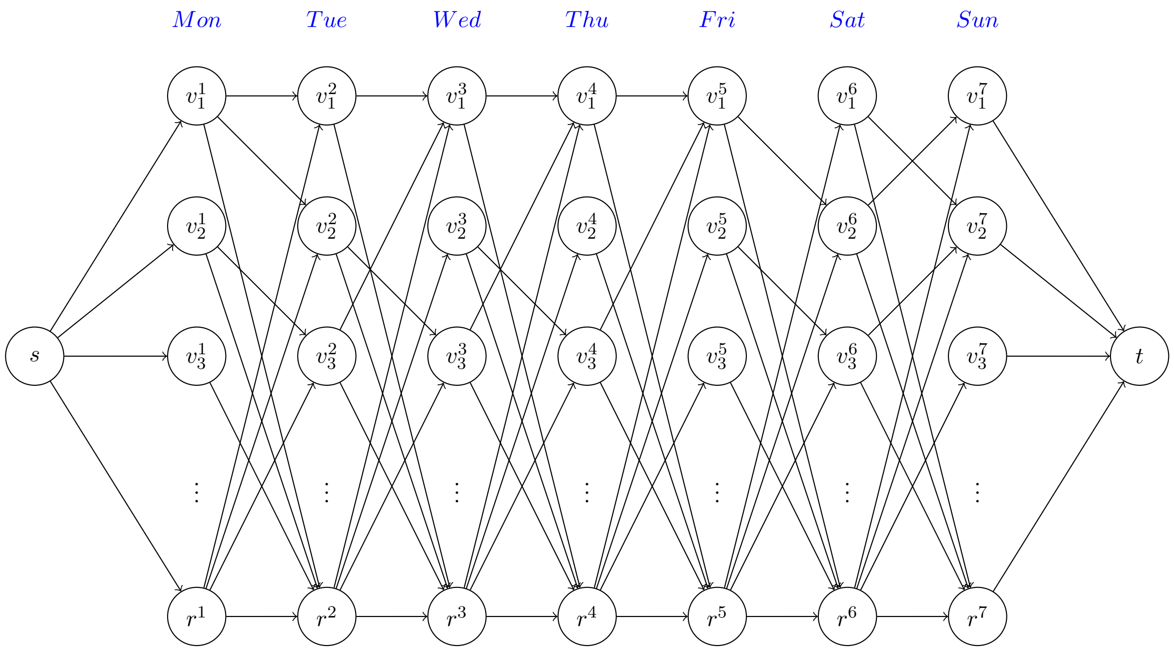


FIGURE 4.7: DAG to build schedules.

A path from the source node s to the sink node t is a schedule satisfying certainly the rest between two consecutive daily shifts, but there is no guarantee that all other constraints are satisfied. Section 4.3 will show how to integrate them.

4.3 Solving Method

This section describes the method used to solve the pricing problem in a CG algorithm, which consists in determining the schedule with the minimum reduced cost. Basically, the method reflects the layered structure of a schedule, constructing a new one in three phases. It combines activities to build feasible timeslots (phase 1). Then, it uses timeslots, breaks and interruptions to create feasible daily shifts (phase 2). Finally, it combines daily shifts and days-off to have a complete feasible schedule (phase 3). Before giving any details, we recall the concept of *expanded graph*, which is obtained from a DFA and allows to find strings recognized by the DFA.

4.3.1 Expanded graph

The concept of the expanded graph was first introduced by Pesant (2004), where the author defined the *regular language membership constraint*. A `regular(x,`A`)` constraint is defined by means of a DFA $A = (Q, \Sigma, \delta, q_0, F)$, describing the regular language to which a sequence `x` of finite-domain variables must belong. Automaton A is unfolded into a layered directed graph, called *expanded graph* (Quimper and Rousseau (2009)), where $n+1$ nodes q^i, with $0 \leq i \leq n$, are created for each state $q \in Q$. Each node belongs to a different layer N^i, and arcs only appear between consecutive layers, making the graph acyclic by construction. For each transition $\delta(q, a) = p$, an arc (q^i, p^{i+1}) labeled a is added to the graph, for $0 \leq i < n$. Finally, all nodes and arcs which are not on a path connecting q_0^0 with a final state q^n, where $q \in F$, are removed. The expanded graph has the property that paths from the first to the last layer are in one-to-one correspondence with a string recognized by the DFA A.

Example 4.6. *Let us consider the automaton defined in Example 4.1. For readability, we avoid labeling all arcs of the expanded graph and we assign color red to arcs with label* 1 *and color blue to arcs with label* 0*. Figure 4.8 shows the corresponding expanded graph for* $n = 4$*.*

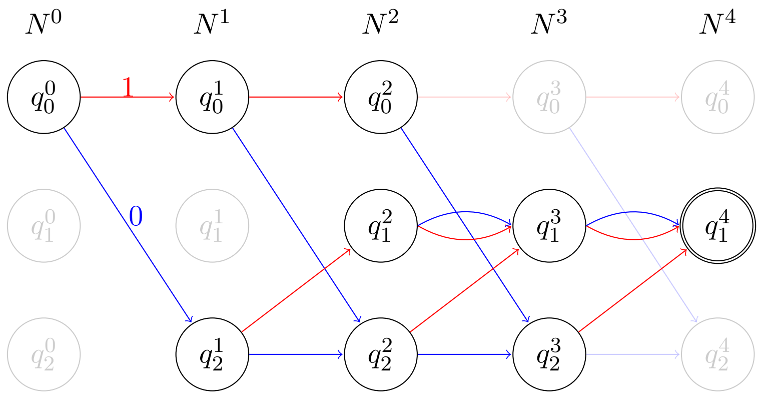


FIGURE 4.8: Expanded graph for the DFA in Figure 4.1.

We remark that nodes and arcs which are not on a path connecting q_0^0 *and* q_1^4 *are removed from the expanded graph. Furthermore, every path connecting these two nodes corresponds to a string recognized by the automaton in Figure 4.1. For instance, the path* $(q_0^0, q_0^1, q_2^2, q_2^3, q_1^4)$ *coincides with the string* 1001*, accepted by the automaton since it contains the sequence* 01*.*

We might need to find a particular path between all paths, of length n, accepted by a DFA A. For instance, suppose that every transition $\delta(q, a) = p$ of A has a given cost c_q^a. Then, the associated expanded graph presents cost c_q^a on all arcs linking nodes corresponding to states q and p on two consecutive layers, that is (q^i, p^{i+1}), for $0 \leq i < n$. The string of length n, accepted by A with the minimum cost, corresponds to the path with minimum cost between all the paths connecting node q_0^0 in the first layer, with one of the final nodes in the last layer. For instance, let us consider again the DFA in Figure 4.1 and the corresponding expanded graph in Figure 4.8. Suppose that every transition with symbol 1 has cost 5 and every transition with symbol 0 has cost 10. Therefore, all red arcs of the expanded graph have cost 5, while all blue arcs have cost 10. The string of length 4 with minimum cost accepted by the automaton, coincides with the shortest path from node q_0^0 to node q_1^4. The minimum cost is 25 and one of the shortest path is $(q_0^0, q_0^1, q_0^2, q_2^3, q_1^4)$, which corresponds to string 1101. We mention Demassey et al. (2006), who extended the `regular` constraint to the `cost-regular` constraint in order to accept only strings whose costs fall within an interval. We will see that we cannot apply the filtering algorithm proposed by the authors, to reduce the size of the expanded graph.

The model presented in Section 4.2 exploits DFA to capture the rules describing feasible timeslots and daily shifts. In the following, we will make use of the associated expanded graph to build, in practice, timeslots and daily shifts.

4.3.2 Phase 1: build timeslots

The first phase combines activities in order to build feasible timeslots. A timeslot is feasible if its duration (length) falls within the minimum l_w^c and the maximum u_w^c consecutive working hours bounds, if it covers slots where the employee is available, and if it is accepted by the DFA defined in Section 4.2.1. In order to define such timeslot, we build the expanded graph as explained in Section 4.3.1, fixing n equal to u_w^c. For simplicity, we add further nodes t_i, called sink nodes, for each layer N^i, with $l_w^c \leq i \leq u_w^c$. Furthermore, for each layer N^i, we add arcs from every terminal node q^i, with $q \in F$, to the sink node t_i. In doing so, the shortest path from node q_0^0 to node $t_{l_w^c}$ corresponds to the minimum cost timeslot with duration l_w^c. The costs to assign to the arcs of the expanded graph are given by dual variables of the reduced master problem (RMP) (2.74)-(2.78).

In this first phase, we build the timeslot with minimum cost *for each starting slot* and *for each duration* that is feasible for the consecutive working hour constraint. The starting slots are selected to be able to complete a timeslot of minimum duration l_w^c within each day. Furthermore, only timeslots that cover slots where the employee is available are generated.

Remark. The filtering algorithm proposed by Demassey et al. (2006) for the `cost-regular` constraints cannot be applied here, since we do not know the interval within which the cost of the shortest path falls. We need to find a timeslot for each starting slot and duration, and its cost can be either positive or negative. We cannot keep only the negative timeslots since positive timeslots may be used to build the final optimal schedule.

Example 4.7. *Let us consider the automaton defined in Example 4.2. Figure 4.9 shows the corresponding expanded graph for $l_w^c = 4$ and $u_w^c = 6$. For readability, in the figure we avoid writing the index of the layer to which nodes correspond. The costs to assign to the*

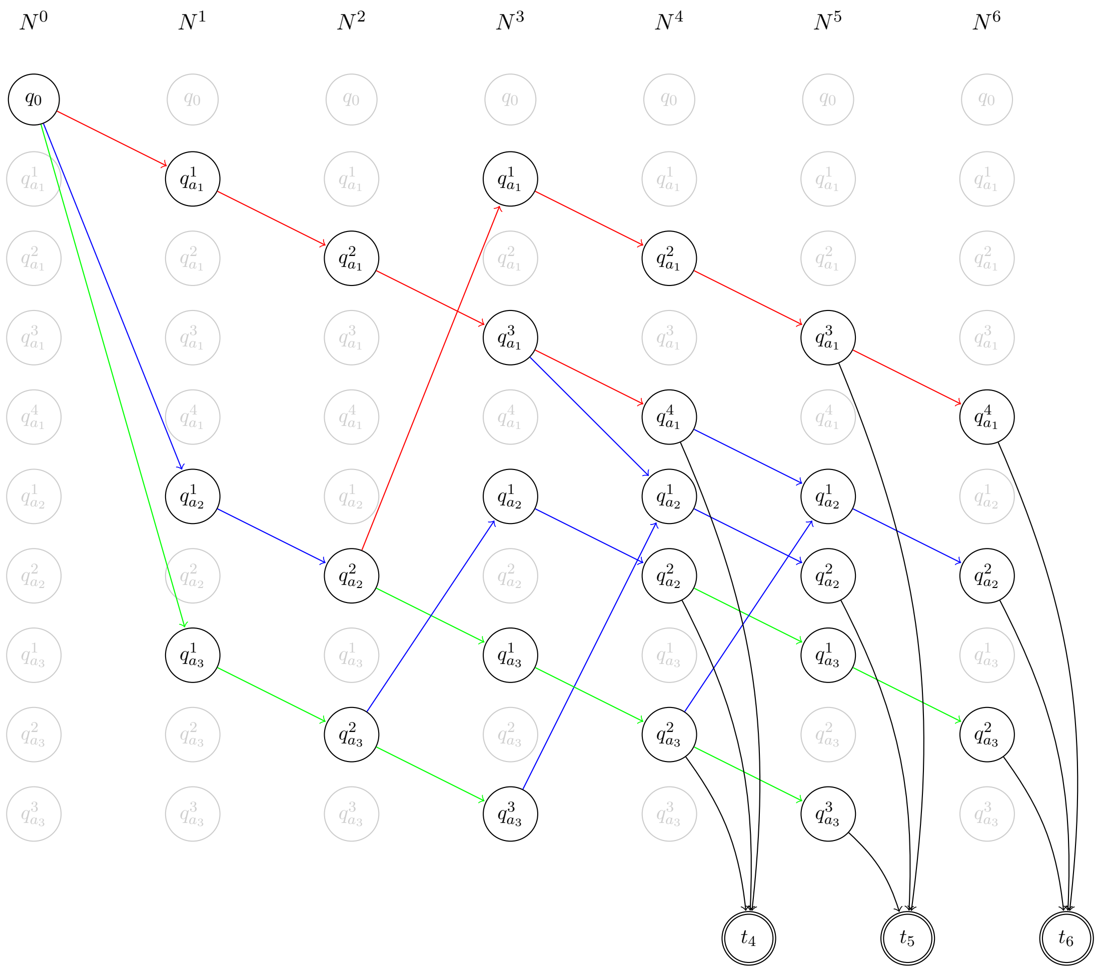


FIGURE 4.9: Expanded graph for the DFA in Figure 4.2

arcs are given by the dual variables of the reduced master problem. Suppose that we want to build the minimum cost timeslot starting at slot j. Thus, arc $(q_0, q^1_{a_1})$ has a cost equal to the dual variable $-u_{ja_1}$ of problem (2.79)-(2.81), corresponding to the workload constraint in the reduced master problem related to activity a_1 and slot j. The minimum cost timeslot starting at j with duration 4, is the shortest path from node q_0 to node t_4.

4.3.3 Phase 2: build daily shifts

The second phase combines timeslots, breaks and interruptions to build feasible daily shifts. A daily shift is feasible if:

- it satisfies the bounds on the daily working hours;
- it satisfies the bounds on the amplitude of the working day;
- it is accepted by one of the DFA defined in Section 4.2.2.

We recall that this DFA capture the constraints on the number and the duration of both breaks and interruptions, but not the constraints on the amplitude of the working day and the total daily working hours. In order to define such a feasible daily shift, we first build the associated expanded graph as explained in Section 4.3.1. Unfortunately, we do not know, a priori, the value of n, that is the number of layers in the expanded graph. Indeed, the length of a string does not coincide with the amplitude of the working day. Therefore, layers of the expanded graph do not correspond to slots, as for the timeslots. However, we note that all the DFAs describing a daily shift are acyclic and n cannot be greater than the longest path between all paths connecting the initial state q_0 with one of the final state. For instance, in Example 4.5, the number of layers is not greater than 13, which corresponds to the length of the longest path from state q_0 and $q^8_{w_3}$.

Concerning the sink nodes, we might need to add a sink for each layer N^i for $1 \leq i \leq n$, due to the fact that the constraints on the amplitude are missing or they are loose. Therefore, we add all sink nodes and, through a post-processing, we remove some of them. More in details, a sink node t_i can be deleted if the daily shift with the minimum (resp. maximum) amplitude reaching node t_i exceed the maximum (resp. minimum) bound on the amplitude. This is done by assigning to every arc its contribution in terms of slots. For instance, let us consider again Example 4.5, the arc of the extended graph linking node q_0 in layer N^0 with node $q^6_{w_1}$ in layer N^1 has cost 6, while the one linking $q^6_{w_1}$ and $q^7_{w_1}$ has cost 1. The shortest and longest paths are evaluated from node q_0 in the first layer to all sink nodes, and the infeasible ones are removed. However, paths reaching the remaining sink nodes are not guaranteed to satisfy the bounds on the amplitude, and, in addition, they are not necessary feasible for the constraint on the daily working hours. For this reason, feasible daily shifts do not coincide simply with paths but with *resource constrained paths*, where the resources concern the amplitude and the working hours.

In this second phase, we build the daily shift with minimum cost *for each starting slot*, *for each amplitude* and *for each working hours.* To find such daily shifts, we implement a label algorithm (see Irnich and Desaulniers (2005)), where paths are characterized by the consumption of resources, in addition to their costs. More in detail, each path p is associated to a label L_p defined as vector

$$L_p = (C_p, W_p, A_p),$$

where C_p is the cost of path p, W_p and A_p are the resources concerning respectively the total daily working hours and the amplitude of the daily shift. The algorithm is applied several times, one for every starting slot selected in order to be able to complete a daily shift of minimum length. Every time a starting slot is selected, the expanded graph does not change, but the costs on the arcs are updated and the label algorithm is applied. The cost of the arcs is given by the cost of the corresponding timeslots found during the first phase in Section 4.3.2.

Dominance pruning. If p_1 and p_2 are two different paths from the source node to a given node, with labels L_{p_1} and L_{p_2} respectively, then p_1 dominates p_2 if $C_{p_1} \leq C_{p_2}$, $W_{p_1} = W_{p_2}$ and $A_{p_1} = A_{p_2}$. This means that path p_1 dominates p_2 if its cost is lower and both resources on daily working hours and amplitude are equal. We remark that the equality on resources W_p and A_p is necessary due to the fact that they are not only bounded from above, but also from below. By eliminating paths through this dominance relation, the complete enumeration of the paths is avoided and only labels corresponding to non-dominated paths are kept.

Remark. We want to point out that the expanded graph defined in Section 4.3.2 allows to find feasible timeslots by solving shortest path problems. The reason lies in the fact that each state of the DFA (Section 4.2.1) corresponds to one slot. As a consequence, the length of each path is exactly the duration of the timeslot. This is not the case for the DFA used to define daily shifts (Section 4.2.2), where there is no one-to-one correspondence between the length of a path and the duration (amplitude) of the daily shift. For instance, if we consider the DFA of Example 4.4, the strings wbw and wiw have both length 3 but they correspond to daily shifts with amplitude respectively 13 and 14. We may ask ourself why we do not define a DFA for daily shift such that the length of a path is the duration of the daily shift. There are three main reasons: first, the presence of the constraint on the total daily working hours cannot be treated by the DFA and it forces us to solve a resource constrained shortest path problem; second, we want to reduce the size of the DFA and, consequently, the size of the expanded graph; third, we need to find all non-dominated paths from the source to the sink nodes, that will be used in phase 3, presented in the next paragraph, to generate complete schedules.

4.3.4 Phase 3: build schedules

The third phase combines daily shifts and days-off to build a feasible schedule. A schedule is feasible if:

- it satisfies the bounds on the total weekly working hours;
- it satisfies the bounds on the the number of working days;
- it satisfies the bounds on the consecutive number of daily shifts;
- it satisfies the rest between two daily shifts.

We recall that the the rest between two daily shifts is directly integrated in the DAG defined in Section 4.2.3, since arcs on consecutive daily shifts are added only if the rest time is satisfied. In order to guarantee that a path from the source node s to the sink node t is a feasible schedule, we need to consider three resources concerning all other three constraints.

In this third phase, we build the schedule with minimum cost by finding the *resource constrained shortest path*. To solve this problem, we implemented a label algorithm similar to the one in phase 2. Each path p is associated to a label L_p defined as vector

$$L_p = (C_p, W_p, R_p, D_p),$$

where C_p is the cost of path p, W_p is the total weekly working hours, R_p is the number of working days and D_p corresponds to the consecutive working days.

Dominance pruning. To avoid exploring all paths and to keep only non-dominated labels, we define the following dominance relation: if p_1 and p_2 are two different paths from the source node to a given node, with labels L_{p_1} and L_{p_2} respectively, then p_1 dominates p_2 if $C_{p_1} \leq C_{p_2}$, $W_{p_1} = W_{p_2}$, $R_{p_1} = R_{p_2}$ and $D_{p_1} = D_{p_2}$. That is, path p_1 dominates p_2 if its cost is lower and resources on weekly working hours, number of working days and consecutive working days are equal. As previously, the equality on resources W_p, R_p and D_p is needed since they have both lower and upper bounds.

Bound pruning. Labels can be pruned not only by feasibility or dominance, but also by bound (Lozano and Medaglia (2013)). Before applying the label algorithm, we evaluate an optimistic prediction C_{v_k} of the cost on the path from each node v_k to the sink node. This lower bound is obtained by calculating shortest paths that do not take into account the resources consumption, but only the costs. Therefore, given a partial path p to node v_k, if $C_p + C_{v_k} \geq 0$, then path p can be discarded since it cannot produce a path with negative cost.

Bi-directional search. We apply the bi-directional search proposed by Righini and Salani (2006). This approach proceeds in three steps: 1) labels are extended forward from the source node s without creating states that cover more than half of the planning horizon; 2) similarly, labels are extended backward from the sink node t; and 3) pairs of forward and backward labels associated with the same node are joined together to obtain a complete path covering the whole planning horizon. Both infeasible and unpromising paths which cannot lead to an improvement of the best reduced cost known so far, are rejected. More precisely, before doing the feasibility test, the approach checks if the reduced cost of the joined label improves the best reduced cost known.

Schedule cost evaluation. Given a vector defining the cost u_{ja} of performing each activity $a \in A$ in each slot $j \in J$, we are now able to evaluate the schedule of the minimum cost. Indeed, u_{ja} is assigned to the corresponding arc in the expanded graph used to build feasible timeslots, and the minimum cost timeslots are built. Then, the costs of the timeslots obtained are given to the corresponding arcs of the expanded graph used to build feasible daily shifts, and the minimum cost daily shifts are built. Finally, the costs of the daily shifts are associated to the corresponding arcs of the DAG used to build schedules, and the minimum cost schedule is built. In a CG algorithm, costs u_{ja} coincide with the dual variables of the reduced master problem (RMP) (2.74)-(2.78).

4.4 Heuristics

The exact resolution of the pricing problem usually takes too much time, and it can be computationally expensive if performed at each iteration of a CG algorithm. The most considerable effort is required on phase 3, where daily shifts are combined with days-off to build complete schedules. Depending on the slot time unit (15 minutes, 30 minutes, 60 minutes, etc.) and on the tightness of the constraints bounds, the number of feasible daily shifts per day can be very high. It might be convenient to solve the pricing problem in a fast but heuristic way, especially during the first iterations of CG, where a negative reduced cost column is likely to be found without much effort. The layered structure of both model and solving method presented in the previous sections, makes the implementation of different heuristics easy. In Section 4.4.1 we describe different strategies that concern phase 2, while in Section 4.4.2 the strategies presented concern phase 3.

4.4.1 Starting slots selection strategy

The first strategies concern the selection of the starting slots in phase 2 (Section 4.3.3), in order to reduce the number of daily shifts generated and the number of times that the label algorithm presented in Section 4.3.3 is applied. The idea is to select slots that are more likely to be good starting slots. Three different strategies are defined based on the workload profile.

(S1) Select the slots where the workload increases. More in details, this strategy selects all the slots with workload higher than the previous slots, including the first slots with demand. In the example shown in Figure 4.10, blue slots are the one selected by this strategy.

(S2) Select the slots with strategy (S1). Moreover, if two consecutive chosen slots j and j' are distant more than u^c_w slots, this strategy select also slots $j+u^c_w$, $j+2u^c_w$, etc., if they corresponds to slots with workload greater than zero. In the example in Figure 4.10, blue and red slots are selected by this strategy. More in details, the blue ones are the first to be identified. Then, since slots 6 and 20 are distant more than $u^c_w = 4$ slots, also slots 10, 14 and 18 are chosen. Similarly, slots 28 and 32 are selected.

(S3) Select all slots.

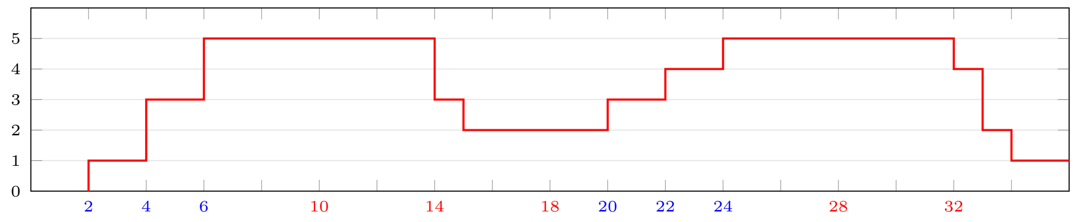


FIGURE 4.10: Example of starting slot selection strategy, with $u^c_w = 4$

4.4.2 Daily shifts selection strategy

The second heuristic strategy concerns the selection of the daily shifts in phase 3, in order to further reduce the size of the DAG presented in section Section 4.3.4. We define six different strategies.

(D1) Select the daily shift with the best cost, for each day and for each daily working hour value.

(D2) Select the daily shift with the best cost, for each day, for each amplitude and for each daily working hour value.

(D3) Select the daily shift with the best cost, for each day and each daily working hour value as in (D1). In addition, the first daily shift, the last daily shift and the one in the middle of the day are selected, for each day and daily working hours value. Ties are broken selecting the daily shift with the best cost. Indeed, by selecting only the best daily shifts, it may happen that a feasible complete schedule cannot be found due to the constraint on the rest between working days. In order to make this less likely to happen, this strategy selects other daily shifts that cover together the whole day.

(D4) Select the daily shift with the best cost, for each day, amplitude and daily working hour value as in (D2). Furthermore, analogously to (D3), the first daily shift, the last daily shift and the one in the middle of the day are selected, for each day, amplitude and daily working hours value. As previous, the goal is to select different daily shifts that cover the entire day.

(D5) Select the daily shift with the best cost, for each slot and for each value of daily working hours.

(D6) Select the daily shift with the best cost for each starting slot, for each amplitude and for each daily working hour value. This strategy consists in selecting all daily shifts.

Remark. When the starting slot selection strategy (S3) is used together with (D6), and all feasible configurations (C1)-(C7) are considered, then the subproblem is solved to optimality.

4.5 Computational results

This section compares the performances of the different heuristics presented in Section 4.4.1, concerning the starting slots selection in phase 2, and in Section 4.4.2, concerning the daily shifts selection in phase 3.

Given one instance of the multi-activity tour scheduling problem, we aim at focusing on the resolution of one pricing problem, which consists in solving a resource constrained shortest path problem. We consider 5 different vectors, that are used to define the costs of the objective function of the pricing problem (2.82). These vectors are not randomly generated, but they are extracted while solving the linear relaxation of the master problem by means of CG. By doing so, each instance of multi-activity tour scheduling results in 5 different instances of resource constrained shortest path. Each of them is solved multiple times, varying the configurations (C1)-(C7), the heuristics on the starting slots (S1)-(S3) and the heuristics on the daily shifts (D1)-(D6). We are interested in evaluating the quality of the solution obtained solving the pricing problem and the computational time needed to solve it. Furthermore, we are interested in the number of daily shifts per day considered in each level of the DAG used in phase 3 (see Figure 4.7). Indeed, most of the computational effort to find the optimal solution is employed in this last phase, where a complete schedule is built, and the number of daily shifts considered is a crucial factor.

The algorithms are implemented in C# and computational experiments are performed on a 64-bit Windows operating system with 977 GB of RAM and 16 processors (only one core is used) Intel Core running at 2.00 GHz. Experiments have been done on the three sets of instances RGR, RGR flexible and Real presented in Section 1.4.

RGR instances. The configurations that need to be considered while building feasible daily shifts in phase 2, are (C1) with one single timeslot, and (C2) with two timeslots divided by a break. The selection of the starting slots for the daily shifts is one of the feature characterizing the three groups G1, G2 and G3. For this reason, we do not perform tests varying strategies (S1)-(S3) on the selection of the starting slots. We fix it to (S3) and we compare the heuristics concerning the selection of the daily shifts (D1)-(D6). Due to the predefined types of daily shifts, we remark that the feasible amplitudes are 4, 6 and 9 hours corresponding to the 4, 6 and 8 daily working hours. As a consequence, heuristics (D1) and (D2) coincide, such as heuristics (D3) and (D4) and heuristics (D5) and (D6).

Instance		(D1)-(D2)			(D3)-(D4)			(D5)-(D6)		
name	#a	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds
20_1_7_v1_G1	1	5.6	0.11	4	1.0	0.14	12	0.0	2.46	144
20_1_7_v2_G1	1	11.1	0.13	4	5.1	0.18	12	0.0	4.57	215
25_1_7_v1_G1	1	2.6	0.12	4	0.4	0.13	12	0.0	2.42	144
25_1_7_v2_G1	1	0.0	0.14	4	0.0	0.16	12	0.0	4.62	216
40_1_7_v1_G1	1	0.0	0.11	4	0.0	0.12	12	0.0	2.41	144
40_1_7_v2_G1	1	*28.8	0.14	4	5.4	0.19	12	0.0	4.65	216
20_3_7_v1_G2	3	0.3	0.39	3	0.3	0.39	9	0.0	1.26	95
20_3_7_v2_G2	3	12.5	0.33	3	8.0	0.33	8	0.0	2.00	143
20_3_7_v3_G2	3	6.5	0.31	3	2.8	0.32	8	0.0	2.01	141
20_1_7_v1_G3	1	5.6	0.06	4	0.0	0.08	10	0.0	0.12	10
20_1_7_v2_G3	1	12.0	0.06	4	4.2	0.08	11	0.0	0.14	15
25_1_7_v1_G3	1	18.9	0.06	4	0.0	0.07	10	0.0	0.12	10
25_1_7_v2_G3	1	0.0	0.06	4	0.0	0.08	11	0.0	0.15	16
40_1_7_v1_G3	1	0.0	0.06	4	0.0	0.08	10	0.0	0.12	10
40_1_7_v2_G3	1	63.5	0.07	4	27.7	0.08	11	0.0	0.15	16
20_3_7_v1_G3	3	0.1	0.14	4	0.0	0.15	10	0.0	0.20	10
20_3_7_v2_G3	3	74.2	0.12	4	15.2	0.14	11	0.0	0.19	16
20_3_7_v3_G3	3	49.7	0.11	4	13.9	0.13	11	0.0	0.19	15
20_5_7_v1_G3	3	3.3	0.08	4	1.9	0.09	11	0.0	0.16	16
20_5_7_v2_G3	3	8.6	0.26	4	2.1	0.28	11	0.0	0.35	16
Average		**15.2**	**0.15**	**4**	**4.4**	**0.16**	**11**	**0.0**	**1.41**	**80**

Table 4.1: Comparison daily shifts heuristics on RGR instances with feasible configurations (C1)-(C2) and starting slot selection strategy (S3).

Table 4.1 shows, in the first two columns, some information about the resource constrained shortest path instance: the *name* of the multi-activity tour scheduling from which we take the pricing problem; the number *#a* of activities considered in the pricing problem. Furthermore, for each daily shift selection heuristic, the table shows the optimality gap *gap(%)*, the computational time *t(s)* and the average number *#ds* of daily shifts per day that appears in the DAG of phase 3. It is important to note that #ds represents the average number of different daily shift shells per day, without considering all the possible combinations of activities that can be performed. For instance, two daily shifts starting at 8am and finishing at 12am without any break are considered the same daily shift, even though activities assigned on the same slots are different. Indeed, two daily shifts with the same starting and finishing slots can not be found in phase 3, due to the dominance applied in phase 2. We also recall that each instance of multi-activity tour scheduling results in 5 instances of resource constrained shortest path, and each line of the table reports average values over the 5 instances.

Strategies (D1)-(D4) select, for each day, some specific daily shifts between all the generated ones. Therefore, they do not guarantee that a complete feasible schedule can be found. When this happens, the gap of the single instance is set to 100 and the average value on the table is marked with a star "*".

The results show, as expected, that heuristics (D1)-(D2) are faster than the others, but the gap with the optimal solution is 15.2% on average with a high standard deviation (21.6), which measures the dispersion of the gap values. Selecting the best daily shifts for each day and for each value of working hours results in choosing 3 or 4 daily shifts per day, depending on the instance. As a consequence, the algorithm finds a poor quality schedule, or even it is not able to find a feasible one, such for the instance 40_1_7_v2_G1. Concerning (D3)-(D4), we can see that the time is comparable with the first heuristics, but the gap is improved and it is less than one third. This is due to the fact that multiple diversified daily shifts are selected for each day. Even if the diversification does not guarantee to find a feasible schedule, it helps in finding one. Finally, strategies (D5)-(D6) allow to find the optimal solution since they both select all daily shifts generated. The computational time increases considerably for instances of groups G1 and G2, while it is comparable for instances of group G3. This is due to their degree of flexibility. Indeed, in groups G1 and G2 daily shifts can start in every slot of the day, while in group G3 only 5 starting slots are selected, reducing in this way the number of feasible daily shifts

RGR flexible instances. The only configurations that can appear for a feasible daily shifts are (C1) with one single timeslots, and (C2)-(C3) with two timeslots divided by either a break or an interruption. This comes from the fact that timeslots have a minimum duration of 4 hours, and the total daily working hours cannot exceed 8 hours.

We do not show the results for every possible combination of heuristics, given that the table would be too large. For this reason, we compare the average performances of the starting slot selection strategies in Table 4.2, and the average performances of the daily shifts selection strategies in Table 4.3.

As previously, Table 4.2 presents the *name* of the instance, the number *#a* of activities in the pricing problem considered, the optimality gap *gap(%)*, the computational time *t(s)* and the number *#ds* of daily shifts per day considered in each level of the DAG in phase 3. The values indicate the average obtained by varying the daily shifts selection strategies (D1)-(D6). The average results in the last line show that the optimality gap decreases when

Instance		(S1)			(S2)			(S3)		
name	#a	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds
20_1_7_v1_G4	1	6.5	1.17	46	4.5	1.63	54	0.9	113.46	356
20_1_7_v2_G4	1	10.7	1.20	45	6.8	1.78	59	4.2	110.97	355
20_3_7_v1_G4	3	0.3	15.59	145	0.3	18.77	155	0.2	113.80	356
20_3_7_v2_G4	3	5.9	6.32	93	7.3	8.73	108	10.4	111.92	354
20_3_7_v3_G4	3	8.9	6.44	93	7.8	8.37	107	4.2	111.51	354
Average		**6.5**	**6.14**	**84**	**5.3**	**7.86**	**97**	**4.0**	**112.33**	**355**

TABLE 4.2: Comparison starting slot heuristics on RGR flexible instances with feasible configurations (C1)-(C3).

using (S2) instead of (S1), or (S3) instead of (S2). This can be explained by the fact that the set of slots selected by (S1) is contained in the set selected by (S2), which is in turn contained in the set selected by (S3). This allows to generate a higher number of daily shifts. However, there could be a drawback in considering more slots. Indeed, we note the behavior of instance 20_3_7_v2_G4, where the gap with (S2) is greater than the one with (S1). The reason for this lies exactly in the fact that (S2) considers a larger set of slots, and, therefore, the best daily shifts generated are certainly better than the ones generated by (S1). As consequence, when strategies (D1)-(D6) select the daily shifts to be considered in each level of the DAG in phase 3, it is not guaranteed that the best schedule found with (S2) is better than the one of (S1). Nevertheless, we can see that in general, strategy (S2) improves the optimality gap. Concerning the computational time, it increases as the slot strategy goes from (S1) to (S3) since a higher number of daily shifts is generated. We remark that the increase is considerable from (S2) to (S3), due to the fact that both strategies do the average considering the daily shift strategy (D6). However, the combination (S3)+(D6) solves the pricing to optimality, and it takes much more time than the combination (S2)+(D6), since (S2) restricts the number of starting slots selected. Indeed, the first combination generates more than 1200 daily shifts per day, while the second combination generates around 100 daily shifts per day.

We also remark that the computational time of instance 20_3_7_v1_G4 in (S2) and (S3) is much higher compare to the computational time of the similar instances 20_3_7_v2_G4 and 20_3_7_v3_G4. The reason for this behavior lies in the fact that the number of starting slots selected by the different strategies strongly depends on the workload profile. A higher number of starting slots selected leads to a higher number of daily shifts generated, which is the case of instance 20_3_7_v1_G4 (145 against 93 daily shifts generated in average).

Analogously, Table 4.3 presents the optimality gap *gap(%)*, the computational time *t(s)* and the average number *#ds* of daily shifts per day in the DAG of phase 3, for each daily shifts selection strategy (D1)-(D6). The values indicate the average obtained by varying the starting slot selection strategies (S1)-(S3). For this reason, the gaps in the last column are not equal to zero, even though strategy (D6) is used and all daily shifts are considered in phase 3 to generate a complete schedule. The average results in the last line show that the gap decreases while going from strategy (D1) to strategy (D6) since a higher number

Instance		(D1)			(D2)			(D3)			(D4)			(D5)			(D6)		
name	#a	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds
20_1_7_v1_G4	1	5.3	0.78	11	3.8	0.90	20	4.1	1.39	37	3.3	2.43	68	3.9	64.40	295	3.3	162.62	480
20_1_7_v2_G4	1	11.7	0.90	11	9.9	0.92	20	6.3	1.33	36	6.3	2.25	66	4.5	62.01	299	4.5	160.49	486
20_3_7_v1_G4	3	0.6	2.32	11	0.1	2.40	20	0.6	2.84	40	0.1	3.79	74	0.1	84.03	450	0.1	200.94	716
20_3_7_v2_G4	3	17.7	1.59	11	11.5	1.77	20	6.3	2.11	38	6.3	3.12	70	2.6	69.34	373	2.6	176.01	599
20_3_7_v3_G4	3	12.8	1.62	11	10.6	1.63	20	7.0	2.13	37	6.3	2.92	69	2.5	68.27	373	2.5	176.06	598
Average		**9.6**	**1.44**	**11**	**7.2**	**1.52**	**20**	**4.9**	**1.96**	**38**	**4.5**	**2.90**	**69**	**2.7**	**69.61**	**358**	**2.6**	**175.22**	**576**

TABLE 4.3: Comparison daily shifts heuristics on RGR flexible instances with feasible configurations (C1)-(C3).

of daily shifts are considered for each day, while the computational time increases for the same reason. We remark that strategies (D3) and (D4) almost halve the gap while keeping low computational time if compared to strategies (D5) and (D6).

Finally, Table 4.4 compares the combination of heuristics (S1)+(D1) with the combination of heuristics (S3)+(D6). The first select the smallest subset of starting slots and the smallest subsets of daily shifts, while the second one solves the problem to optimality.

Instance		(S1)+(D1)			(S3)+(D6)		
name	#a	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds
20_1_7_v1_G4	1	7.6	0.35	11	0.0	481.56	1232
20_1_7_v2_G4	1	15.0	0.39	11	0.0	474.20	1232
20_3_7_v1_G4	3	0.6	1.97	11	0.0	477.69	1232
20_3_7_v2_G4	3	8.7	1.21	11	0.0	478.51	1232
20_3_7_v3_G4	3	14.9	1.09	11	0.0	478.26	1232
Average		**9.4**	**1.00**	**11**	**0.0**	**478.04**	**1232**

TABLE 4.4: Comparison between the heuristic (S1)+(D1) and the optimal resolution (S3)+(D6) with configurations (C1)-(C3), on RGR flexible instances.

As expected, these results confirm that finding the optimal schedule becomes more costly when the degree of flexibility in the problem increases. Indeed, RGR flexible instances differ from RGR instances for a higher level of flexibility in terms of timeslot duration, amplitude of the daily shift, pause type and pause duration. More in detail, timeslots in RGR have a duration of either 4 hours or 6 hours, while all durations in between are feasible for RGR flexible. Furthermore, the pause considered in RGR is a break of 1 hour, that can be placed only between two timeslots of 4 hours. RGR flexible instances, instead, consider both break and interruptions, which can be assigned as soon as consecutive working hours constraints are satisfied. As consequence, the number of feasible daily shift generated each day goes from a maximum of 216 for RGR, to 1232 for RGR flexible. The next paragraph considers a set of instances with an even higher degree of flexibility.

Real instances. Table 4.5 compares the average performances of the configurations considered to build feasible daily shifts as explained in Section 4.2.2. Due to work regulations on interruptions (see Section 1.2.1) that limit the number of daily interruptions at one, all configurations (C1)-(C6) allow to build feasible daily shifts, and only (C7) can be excluded. As in the previous tables, we report the *name* of the instance, the number *#a* of activities in the pricing problem considered, the optimality gap *gap(%)*, the computational time *t(s)* and the number of daily shifts *#ds* per day considered in each level of the DAG in phase 3. Instances are grouped according the time unit, meaning that the first (second and third) group of instances considers a time unit of 60 minutes (30 minutes and 15 minutes). The values reported indicate the average obtained by varying both starting slots selection strategies (S1)-(S3) and daily shifts selection strategies (D1)-(D6). Due to the high level of flexibility, the number of feasible daily shifts generated when solving the instances with

Instance		(C1)			(C1)-(C3)			(C1)-(C6)		
name	#a	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds
57_11_60_le	6	0.0	0.17	25	0.0	4.47	106	0.0	6.63	108
75_14_60_ll	8	0.0	0.10	19	0.0	2.97	91	0.0	4.54	104
43_15_60_lb	8	0.0	0.16	23	0.0	2.89	91	0.0	4.22	93
38_05_60_nh	5	0.0	0.14	24	0.0	2.08	80	0.0	2.82	80
23_05_60_nn	5	13.8	0.11	20	0.0	1.38	63	0.0	1.77	63
Average (60min)		**2.8**	**0.14**	**22**	**0.0**	**2.76**	**86**	**0.0**	**3.99**	**90**
57_11_30_le	6	0.0	1.59	59	0.0	229.30	355	0.0	745.12	389
75_14_30_ll	8	0.0	0.75	47	0.0	169.73	332	0.0	544.38	402
43_15_30_lb	8	0.0	1.42	54	0.0	251.60	375	0.0	597.38	379
38_05_30_nh	5	0.0	1.67	63	0.0	74.22	274	0.0	140.03	274
23_05_30_nn	5	13.3	1.47	55	0.0	55.61	219	0.0	124.56	219
Average (30min)		**2.7**	**1.38**	**56**	**0.0**	**156.09**	**311**	**0.0**	**430.29**	**333**
57_11_15_le	6	0.0	22.19	110	0.0	456.53	477	0.0	431.81	499
75_14_15_ll	8	0.0	7.78	83	0.0	220.16	405	0.0	374.95	491
43_15_15_lb	8	0.0	21.16	100	0.0	430.71	530	0.0	424.91	536
38_05_15_nh	5	0.0	27.82	130	0.0	177.31	383	0.0	204.85	395
23_05_15_nn	5	13.4	26.02	119	0.0	237.57	342	0.0	184.14	354
Average (15min)		**2.7**	**20.99**	**108**	**0.0**	**304.46**	**427**	**0.0**	**324.13**	**455**

TABLE 4.5: Comparison configurations (C1)-(C6) on Real instances.

15 minutes time unit to optimality (i.e., combining configurations (C1)-(C6) with strategies (S3) and (D6)) was in average 15000 per day. The computational time that was exceeding 12 hours, and the algorithm was not able to solve to optimality the pricing problem. For this reason, we decided to solve these instances reducing the number of shifts selected in phase 3, and considering daily shifts selection strategies (D1)-(D5) (i.e., excluding (D6)).

The most remarkable observation arising from the results concerns the fact that most of the gap values are equal to zero, except for 3 instances. This can be explained by the fact that there are many equivalent solutions, as result of the increased degree of flexibility, and a solution with minimal cost can be found for the different configurations. In particular, it seems that allowing one break or one interruption per day, a sufficient level of flexibility is introduced to be able to find an optimal solution. Indeed, all gap values are equal to zero when the instances are solved with configurations (C1)-(C3). This will be confirmed later in Table 5.5. We will see that allowing a higher level of flexibility with configurations (C4)-(C6), does not bring to descrease the optimal value (cost saving).

Table 4.6 and Table 4.7 compare the average performances of the starting slots selections strategies (S1)-(S3), and the average performances of the daily shifts selection strategies (D1)-(D6), respectively. As previously, the tables report the *name* of the instance, the number *#a* of activities in the pricing problem, the optimality gap *gap(%)*, the computational time *t(s)* and the number *#ds* of daily shifts considered in each level of the DAG in phase 3. The values reported indicate the average obtained by varying both configurations (C1)-(C6) and daily shifts selection strategies (D1)-(D6). In Table 4.6, we remark that the average gap values are the same for strategies (S1) and (S2). Indeed, instances 23_05_60_nn, 23_05_30_nn and 23_05_15_nn present the same gap with the two slot strategies. This is due to the fact that the average values in each line consider also the results obtained using

Instance		(S1)			(S2)			(S3)		
name	#a	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds
57_11_60_le	6	0.0	1.28	56	0.0	2.47	75	0.0	7.52	108
75_14_60_ll	8	0.0	0.83	50	0.0	2.19	72	0.0	4.59	93
43_15_60_lb	8	0.0	1.45	58	0.0	1.68	61	0.0	4.13	88
38_05_60_nh	5	0.0	1.01	50	0.0	1.58	61	0.0	2.44	73
23_05_60_nn	5	6.8	0.42	34	6.8	0.55	39	0.1	2.29	72
Average (60min)		**1.4**	**1.00**	**50**	**1.4**	**1.69**	**62**	**0.0**	**4.19**	**87**
57_11_30_le	6	0.0	29.03	151	0.0	54.43	198	0.0	892.55	454
75_14_30_ll	8	0.0	32.04	159	0.0	94.18	228	0.0	588.64	395
43_15_30_lb	8	0.0	52.70	184	0.0	61.36	198	0.0	736.32	427
38_05_30_nh	5	0.0	22.19	144	0.0	29.14	160	0.0	164.59	306
23_05_30_nn	5	6.7	8.91	95	6.7	8.88	105	0.0	163.85	294
Average (30min)		**1.3**	**28.97**	**147**	**1.3**	**49.60**	**178**	**0.0**	**509.19**	**375**
57_11_15_le	6	0.0	95.62	245	0.0	98.97	271	0.0	715.93	570
75_14_15_ll	8	0.0	51.81	222	0.0	64.10	262	0.0	486.99	494
43_15_15_lb	8	0.0	160.68	299	0.0	163.08	308	0.0	553.02	558
38_05_15_nh	5	0.0	57.65	225	0.0	63.70	243	0.0	288.62	441
23_05_15_nn	5	6.7	48.54	188	6.7	50.27	197	0.0	348.91	430
Average (15min)		**1.3**	**82.86**	**236**	**1.3**	**88.02**	**256**	**0.0**	**478.69**	**499**

Table 4.6: Comparison starting slot heuristics (S1)-(S3) on Real instances.

configuration (C1), that was the only one giving a positive gap in these specific instances when using strategies (S1) and (S2). However, when using strategy (S3) that selects all starting slots of the day, these gap values decrease considerably or go to 0. Besides these three instances, all the others have gaps equal to zero. Similarly, in Table 4.7 we can see that the gap values do not decrease by varying strategies (D1)-(D6). In other words, the selection of the daily shifts does not affect the gap in average for these instances, unlike the configurations considered or the slot selection strategy selected. The intuition is that there are many equivalent solutions in these pricing problems due to the high flexibility, and even though (D1)-(D6) select a subset of daily shifts, they allow to find an optimal solution. Slot stratgies (S1)-(S3), instead, fix the starting slots of the daily shifts reducing the space of feasible daily shifts, and the selected slots may be not of good quality.

Table 4.8 compares the combination of heuristics that leads to the fastest computational time with the one that leads to the optimal or the best value. The fastest heuristic is obtained combining (C1)+(S1)+(D1). The optimal value is obtained only for instances with 60 and 30 minutes time unit, combining ((C1)-(C6))+(S3)+(D6). As previously said, it was not possible to solve to optimality instances with 15 minutes time unit, since the average number of daily shifts generated per day was 15000. The best value is obtained using daily shifts selection strategy (D5) instead of (D6), that is combining ((C1)-(C6))+(S3)+(D5). The results underline the differences in the computational effort needed by the two combinations. However, most of the instances are solved by the fastest heuristic with a gap equal to zero.

Instance		(D1)			(D2)			(D3)			(D4)			(D5)			(D6)		
name	#a	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds
57_11_60_le	6	0.0	0.18	8	0.0	0.36	25	0.0	0.30	27	0.0	1.72	83	0.0	1.95	87	0.0	18.02	249
75_14_60_ll	8	0.0	0.13	7	0.0	0.19	21	0.0	0.19	21	0.0	0.91	66	0.0	1.32	80	0.0	12.47	232
43_15_60_lb	8	0.0	0.18	8	0.0	0.33	25	0.0	0.33	27	0.0	1.72	81	0.0	1.49	75	0.0	10.49	197
38_05_60_nh	5	0.0	0.09	8	0.0	0.19	19	0.0	0.20	22	0.0	0.81	58	0.0	1.16	77	0.0	7.62	182
23_05_60_nn	5	4.6	0.07	8	4.6	0.15	18	4.6	0.17	23	4.6	0.78	55	4.5	0.82	57	4.5	4.52	129
Average (60min)		**0.9**	**0.13**	**8**	**0.9**	**0.24**	**22**	**0.9**	**0.24**	**24**	**0.9**	**1.19**	**69**	**0.9**	**1.35**	**75**	**0.9**	**10.62**	**198**
57_11_30_le	6	0.0	1.28	15	0.0	3.07	67	0.0	2.05	49	0.0	26.15	230	0.0	31.03	248	0.0	1888.44	998
75_14_30_ll	8	0.0	1.04	13	0.0	1.87	58	0.0	1.39	42	0.0	11.65	201	0.0	21.37	244	0.0	1392.40	1005
43_15_30_lb	8	0.0	1.58	15	0.0	4.17	81	0.0	2.42	49	0.0	34.54	276	0.0	22.58	214	0.0	1635.45	983
38_05_30_nh	5	0.0	0.76	14	0.0	1.99	52	0.0	1.41	45	0.0	11.86	177	0.0	18.22	215	0.0	397.60	718
23_05_30_nn	5	4.4	0.64	14	4.4	1.61	49	4.4	1.31	45	4.4	9.68	158	4.4	14.53	178	4.4	335.51	545
Average (30min)		**0.9**	**1.06**	**14**	**0.9**	**2.54**	**61**	**0.9**	**1.72**	**46**	**0.9**	**18.78**	**208**	**0.9**	**21.55**	**220**	**0.9**	**1129.88**	**850**
57_11_15_le	6	0.0	13.06	27	0.0	40.72	203	0.0	17.75	94	0.0	405.53	718	0.0	1040.48	768	-	-	4777
75_14_15_ll	8	0.0	10.26	23	0.0	20.66	177	0.0	10.35	81	0.0	217.71	631	0.0	745.85	717	-	-	4621
43_15_15_lb	8	0.0	16.45	27	0.0	60.04	263	0.0	19.53	96	0.0	747.19	915	0.0	618.09	640	-	-	4872
38_05_15_nh	5	0.0	7.65	26	0.0	22.30	158	0.0	11.44	86	0.0	213.50	542	0.0	428.40	702	-	-	3493
23_05_15_nn	5	4.5	6.34	26	4.5	22.46	148	4.5	9.14	85	4.5	181.44	481	4.5	526.83	618	-	-	2887
Average (15min)		**0.9**	**10.75**	**26**	**0.9**	**33.23**	**190**	**0.9**	**13.64**	**89**	**0.9**	**353.07**	**657**	**0.9**	**671.93**	**689**			**4130**

TABLE 4.7: Comparison daily shifts heuristics (D1)-(D6) on Real instances.

Instance		(C1)+(S1)+(D1)			((C1)-(C6))+(S3)+(D6)		
name	#a	gap(%)	t(s)	#ds	gap(%)	t(s)	#ds
57_11_60_le	6	0.0	0.05	5	0.0	67.79	523
75_14_60_ll	8	0.0	0.03	4	0.0	41.30	477
43_15_60_lb	8	0.0	0.05	5	0.0	33.79	387
38_05_60_nh	5	0.0	0.02	5	0.0	19.79	312
23_05_60_nn	5	20.5	0.03	5	0.0	17.49	302
Average (60min)		**4.1**	**0.04**	**5**	**0.0**	**36.0**	**400**
					((C1)-(C6))+(S3)+(D6)		
57_11_30_le	6	0.0	0.17	8	0.0	12146.38	2843
75_14_30_ll	8	0.0	0.15	6	0.0	7949.96	2570
43_15_30_lb	8	0.0	0.14	8	0.0	9175.21	2532
38_05_30_nh	5	0.0	0.12	9	0.0	1819.46	1654
23_05_30_nn	5	20.0	0.12	9	0.0	1945.65	1560
Average (30min)		**4.0**	**0.14**	**8**	**0.0**	**6607.33**	**2232**
					((C1)-(C6))+(S3)+(D5)		
57_11_15_le	6	0.0	0.94	14	0.0	3764.84	2172
75_14_15_ll	8	0.0	0.62	10	0.0	3977.60	2148
43_15_15_lb	8	0.0	0.66	14	0.0	2201.02	1710
38_05_15_nh	5	0.0	0.69	17	0.0	1525.16	1557
23_05_15_nn	5	20.1	0.69	17	0.0	1511.85	1557
Average (15min)		**4.0**	**0.72**	**14**	**0.0**	**2596.09**	**1829**

TABLE 4.8: Comparison between the heuristic (C1)+(S1)+(D1) and the optimal resolution ((C1)-(C6))+(S3)+(D6) (or best resolution ((C1)-(C6))+(S3)+(D5)) on Real instances.

4.6 Conclusions

In this chapter we proposed a method to generate new schedules that minimize a given objective function (2.82), where costs correspond to dual variables when using CG. The proposed method models and solves the problem is three phases, that reflect the layered structure of schedules. The first phase combines activities to build feasible timeslots. Rules defining timeslots are modeled by means of an automaton, and shortest path problems are solved to generate timeslots. The second phase combines timeslots, breaks and interruptions to build feasible daily shifts. Rules are modeled by means of an automaton, and resource constrained shortest paths problems are solved to generate daily shifts. Finally, the third phase generates a complete schedule by combining daily shifts and days-off. Rules are modeled with a directed acyclic graph and a resource constrained shortest path problem is solved to generate the optimal schedule.

The high degree of flexibility in terms of timeslot duration, amplitude of the daily-shift, pause duration and pause type forces us to develope different heuristics concerning the selection of the starting slots in phase 2, and the selection of the daily shifts in phase 3. Computational results have been performed on three sets of instances with an increasing level of flexibility: instances from Restrepo et al. (2016) (RGR), a more flexible version of these instances (RGR flexible), and real instances coming from a fast food restaurant chain (Real). Results show the optimality gap and the solving time obtained on average varying configurations (C1)-(C6), starting slot selection strategies (S1)-(S3) and daily shift selection strategies (D1)-(D6). For the RGR and RGR flexible instances, the optimality gap deteriorates when using strategies that restrict the solution space, while the computational time decreases. This is not the case for the Real instances, where most of the heuristic strategies allow finding the optimal solution. This may be explained by the high number of equivalent solutions due to the degree of flexibility that characterizes these instances.

The method proposed in this chapter is used, together with the dual ascent heuristic presented in Chapter 3, in a CG framework which solves the linear relaxation of the master problem (2.69)-(2.73). In order to obtain integer solution, CG is embedded in a B&B procedure. The resulting B&P algorithm is presented in Chapter 5.

Chapter 5

Branch-and-Price

This chapter presents a B&P algorithm for solving the multi-activity tour scheduling problem presented in Chapter 1. The B&P method makes use of CG to solve the linear relaxation of the Dantzig-Wolfe decomposition (2.69)-(2.73), where the integrality constraints $x_p^i \in \{0, 1\}$ are replaced by $0 \leq x_p^i \leq 1$. In order to find an optimal integer solution starting from the optimal fractional solution of the linear relaxation, a branching procedure needs to be employed besides CG. This procedure eliminates the current optimal fractional solution and divides the solution space in two parts, without eliminating integer solutions. In addition, the branching procedure needs to be managed in the pricing problems of CG, since, at each node of the search tree, CG is used to solve the linear relaxation of the current integer problem. The reader can refer to introductory works like Lübbecke and Desrosiers (2005) and Barnhart et al. (1998) to have an exhaustive insight to the subject. Three elements are the key of a B&P algorithm: 1) a CG framework to compute the lower bound at each node of the search tree; 2) a branching rule to choose the fractional variables to branch on; 3) a heuristic to determine an upper bound on the optimal value of the problem. The goal of this chapter is to embed the methods proposed in Chapter 3 and Chapter 4 in a B&P scheme. We use the dual ascent (DA) heuristic and the pricing solving method in CG, for obtaining a lower bound on the optimal solution, and a large neighborhood search heuristic for computing an upper bound at each node of the search tree. We use the branching rules proposed in Section 5.2.

The chapter is organized as follows: Section 5.1 presents the framework of the CG used, Section 5.2 details the branching rule adopted, and Section 5.3 introduces a LNS heuristic for obtaining an upper bound on the optimal value. Finally, computational results are reported in Section 5.4, and conclusions are drawn in Section 5.5.

5.1 Column generation

A basic component of any B&P algorithm is CG that provides a lower bound at each node of the search tree. The use of CG is justified by the fact that the linear relaxation of the master problem (2.69)-(2.73) presents an exponential number of variables. The CG framework in the proposed B&P makes use of the DA presented in Chapter 3, that has proven to speed up the convergence of CG and to soften the oscillations of the dual variables. We recall that DA is based on a parametric reformulation of the reduced master problem, and it uses the Lagrangian relaxation and the subgradient method for obtaining dual solutions in order to generate new columns. The details on the CG scheme that employs DA are presented in Section 3.4.3. We remark that if CG makes use only of DA, it cannot converge since the optimal dual variables are needed to prove optimality. In order to develop an exact B&P algorithm, we need to compute the lower bound at each node of the search tree. For this reason, as soon as CG with DA stops, the classical CG, where an LP solver is used to evaluate the optimal dual variables, is employed until convergence. The pricing problems are solved with the algorithm presented in Chapter 4.

It is well known that CG suffers from several drawbacks that lead to a slow convergence of the algorithm (Vanderbeck (2005) and Lübbecke and Desrosiers (2005)). However, different strategies directed at speeding up CG can be adopted. These strategies are the key elements for implementing an efficient CG algorithm. Desaulniers et al. (2002) describe many acceleration ideas in the context of specific applications such as vehicle routing and crew scheduling. In the following we describe the techniques used in our CG.

Intensification and diversification. Instead of adding only the column with minimum reduced cost at each iteration of CG, several columns with negative reduced cost can be added to the reduced master problem. This generally decreases the number of CG iterations, and it is particularly easy when the method used to solve the pricing problem allows to compute several columns, such as label algorithm. Among the strategies that can be used to select the columns, we find intensification and diversification (Touati-Moungla et al. (2010)). The first selects many columns between the most negative and similar ones can be added, while the second one has the purpose of selecting a diverse set of columns. Unfortunately, intensification typically overloads the reduced master problem, and generates a huge number of useless variables. Some strategies can be used to generate a smaller number. Touati-Moungla et al. (2010) have studied the characteristics and the pertinence of the information brought by the generated columns to the restricted master problem description. The authors state that diversification, which consists of generating complementary columns, allows to characterize quickly a good approximation of the feasible space of the master problem, without increasing considerably its size.

In the CG implemented in this thesis, intensification selects the 3 columns with the lowest negative reduced cost for each pricing problem. Diversification is performed in the third phase of the pricing resolution method (Section 4.3.4). Recall that in this phase, daily shifts are combined into feasible schedules using a label algorithm. We use the diversification by resolution, which consists of computing complementary schedules, i.e., each computed schedule contains daily shifts that do not belong to any other computed schedule. More in detail, the first resolution finds the schedule with minimum reduced cost, then the daily shifts covered by this schedule are deleted from the DAG, and the algorithm finds the best schedule covering the remaining nodes. This is performed until 3 diversified schedules with negative reduced cost are defined. The advantage of this method concerns the fact that the generated schedules do not cover the same daily shifts. However, solving multiple times the third phase can be costly. For this reason, diversification is applied when the pricing problem is solved heuristically, in conjunction with the daily shifts selection heuristics that restrict the size of the DAG in the third phase.

Diversification and intensification are combined in the CG framework as follows: diversifications is applied the first iterations (= 10) of CG to quickly characterize a good approximation of the master problem without overloading it, then intensification is used in the remaining iterations in order to complete the optimal basis.

Columns elimination. Generally, the proportion of columns in the optimal basis is negligeable compared to the whole columns in the master problem. To limit the size of the reduced master problem, all variables with reduced cost that exceeding 10^{-12} are deleted when the total number of columns is greater than a given threshold. Due to the variety of the instances tested, it was not easy to define a common threshold for the number of columns kept in the reduced master problem. For this reason, we decided to set this threshold depending on the number of rows, that is 10 times the number of rows.

Heuristic pricing. When the pricing problems require a considerable computational effort, it may not be convenient to solve them to optimality at each iteration of CG. In addition, good negative reduced cost columns are very likely to be found with a heuristic mostly in the first iterations of CG. For this reason, we employ the heuristic strategies presented in Section 4.4 for selecting the starting slots in phase 2, and the daily shifts in phase 3. The main idea consists in using, at first, the fastest heuristic that most restricts the pricing solutions space. Then, when no more negative reduced cost columns can be found, a less restricting strategy is selected until the exact resolution of the pricing problem is performed. The goal is to optimally solve the pricing problem only to prove that no more negative reduced cost columns exist and CG can terminate. Due to the variety of the instances tested, the (increasing) heuristic strategies used are different from set to set of instances, and will be described in the computational results.

Partial pricing. Depending on the level of employees heterogeneity (skills, availability, contract regulations), CG can present many pricing problems. In the worst case, each employee has specific skills, availabilities and contract regulations, and a pricing problem taking into account these features is needed for each one. On the contrary, if all employees are equivalent, only one pricing problem is needed. A high computational effort is required at each iteration of CG when there is a large number of pricing problems. Furthermore, similar schedules tend to be generated since all the subproblems use the same dual variables. To overcome this issue, partial pricing can be used. Gamache et al. (1999) use this strategy in the context of airline crew rostering. The idea is to solve a fixed number of pricing problems at each iteration. They are solved one after the other until the required number of subproblems have generated at least one negative reduced cost column each. These columns are then added to the reduced master problem. In the next iteration of CG, the resolution of the subproblems picks up where it left off, using the new optimal dual variables. As CG progresses, it becomes more difficult finding the required number of subproblems generating negative reduced cost columns, and the number of subproblems solved increases. Finally, all the subproblems are solved until no more columns are generated.

5.2 Branching rule

Besides the CG framework, another key element of B&P is the strategy selected for branching (Vanderbeck and Wolsey (1996), Barnhart et al. (1998)). Different strategies have been used in the literature in the context of multi-activity tour scheduling. Gérard et al. (2016) apply a branching rule on the original variables x^i_{ja}, selecting the fractional one with the value closest to 0.5. Restrepo et al. (2016) select two employees i_1 and i_2 whose variables hold the two largest fractional values. Then variable identified for i_1 is fixed to 1, while employee i_2 is used for the branching. Indeed, the authors determine for i_2 the two variables that hold the largest fractional value, and they apply the branching rule suggested by Côté et al. (2013): by comparing the schedules associated to the variables for i_2, they identify the first divergent slot j, i.e. the first slot where the two schedules are different. According to the activities performed in j, they create a partition of the set of activities. Then, they create two nodes: each of them ensures that i_2 does not perform in slot j the activities belonging to the associated partition. However, the fact of fixing one schedule at each node makes their B&P heuristic, since the depth of the search tree cannot be greater than the number of employees.

The branching strategy used in the presented B&P is the one proposed by Gérard et al. (2016). This strategy is classically used with the goal of reducing the infeasibility at most:

- $x^i_{ja} = 0$ (branch right) forbids employee i to perform activity a in slot j;
- $x^i_{ja} = 1$ (branch left) imposes employee i to perform activity a in slot j.

Ties are broken arbitrary using the employee index i. To implement this strategy, some modifications to the pricing problems are necessary. In the following we show how the three phases of the pricing resolution method presented in Section 4.3 are modified for dealing with the variable fixing.

- $x^i_{ja} = 0$:
  - *phase 1*: when timeslots covering slot j are generated, all arcs entering nodes corresponding to activity a and slot j are deleted from the graph. Therefore, for each starting slot *start* such that $j \geq start$ and $j \leq start + u^c_w - 1$, we find the level in the expanded graph corresponding to slot j, and for all nodes in this level that correspond to activity a, the entering arcs are deleted. The arcs are restored as soon as another *start* slot is selected. None of the timeslots covering j will have activity a performed on j;

  - *phase 2*: no modifications are needed;
  - *phase 3*: no modifications are needed.
- $x^i_{ja} = 1$:
  - *phase 1*: when timeslots covering slot j are generated, all arcs entering nodes that do not correspond to activity a and slot j are deleted from the graph. Therefore, for each starting slot *start* such that $j \geq start$ and $j \leq start + u^c_w - 1$, we find the level in the expanded graph corresponding to slot j, and for all nodes in this level that do not correspond to activity a, the entering arcs are deleted. The arcs are restored as soon as another *start* slot is selected. All timeslots covering j will have activity a performed on j;
  - *phase 2*: daily shifts are generated for all starting slots, for all amplitudes and for all working hours. Starting slots j' that do not allow to cover slot j are not considered (either because j' is after j, or beacuse j' is too early and even with a daily shift of maximum amplitude we are not able to cover j). For all the other starting slots, since we generate daily shifts for all amplitudes and for all working hours, some of them may not cover slot j (for instance, due to a break). These daily shifts need to be discarded;
  - *phase 3*: rest nodes are deleted from the day to which slot j belongs.

The search tree is explored using a depth-first strategy.

5.3 Upper bound

Primal solutions provide upper bounds on the optimal value of the problem, and they are used to prune the search tree and fathom nodes whose lower bounds exceed the best upper bound found. In this section we give the main ideas of the large neighborhood search (LNS) heuristic used to determine an integer solution at each node of the search tree. However, the method is presented in the next part of the thesis, and we refer the reader to Section 6.1 for the details.

At each node of the search tree, CG solves the linear relaxation of the current integer problem and evaluates an optimal fractional solution. A first integer solution can be obtained by rounding the fractional solution, selecting the variable x^i_p with the highest value between all $p \in \tilde{P}^i$ for each employee $i \in I$. Since it is very likely that the obtained upper bound is far from the optimal value, an LNS heuristic is employed to improve the integer solution. LNS heuristic has been introduced by Shaw (1998), and it improves the quality of an initial solution by iteratively destroying (destructor operator) and repairing (constructor operator) part of the current solution. In the proposed LNS, destroying consists in removing the schedule associated to an employee, while repairing consists in building a new schedule that minimizes an objective function defined on the residual demand. The details on destructor and constructor operators are given in Section 6.1.1.1 and Section 6.1.1.2 respectively.

5.4 Computational results

This section shows the computational results of the B&P described. The algorithm has been coded in C# and tests have been performed on a 64-bit Windows operating system with 977 GB of RAM and 16 processors (only one core is used) Intel Core running at 2.00 GHz. CPLEX 12.7 has been used as LP solver. The B&P stops as soon as optimality is proven, or when the total time reaches a time limit of 8 hours. Experiments have been done on the three sets of instances RGR, RGR flexible and Real presented in Section 4.5, and compare the performances of the B&P with the solver CPLEX 12.7 used to solve the compact MILP model presented in Section 2.1.

We have previously said that the pricing problems are solved heuristically until negative reduced cost columns can be generated. The idea at the basis consists in using (increasing) heuristic strategies starting from the most restricting to the least restricting one, until the pricing problems are solved to optimality. However, the strategies used to solve the pricing problems are different from one set of instances to the other due to their diversity, and they will be detailed while presenting the results for each set of instances. We recall that the heuristic strategies concern the selection of the starting slots in phase 2 (cf. Section 4.4.1), and the selection of the daily shifts in phase 3 (cf. Section 4.4.2). Furthermore the pricing problem can be solved heuristically also by considering a subset of configurations used to generate the daily shifts (cf. Section 4.2.2).

RGR instances. When solving this set of instances, CG solves heuristically the pricing problems starting from the daily shift selection strategy (D1). When no columns with negative reduced cost are found, CG uses strategy (D3). Finally, to prove optimality, strategy (D6) is used. We recall that RGR instances consider predefined types of daily shifts: four-hour shifts, six-hour shifts or eight-hour shifts with one-break in the middle. As a consequence, there is a one-to-one correspondence between amplitude of the working day and daily working hours. Therefore, heuristics (D1) and (D2) coincide, such as heuristics (D3) and (D4), and heuristics (D5) and (D6). The selection of the starting slots is one of the feature characterizing the three groups G1, G2 and G3 of instances RGR. For this reason, the starting slot selection strategy is fixed to (S3), that selects all feasible slots. Finally, the configurations used are (C1) with one timeslot, and (C2) with two timeslots divided by a break, which are the only feasible configurations for these instances. Due to the presence of equivalent employees, the number of pricing problems does not exceed 8, therefore no partial pricing is applied.

Table 5.1 shows the results for the RGR instances. In particular, the first and the second columns show the name of the instance and the corresponding lower bound found solving the linear relaxation of the master problem (2.69)-(2.73) at the root node. The next seven columns show the results of the B&P algorithm, that is the upper bound ub achieved, the total computational time $t(s)$ reported in seconds, the final optimality gap $gap\%$, the number of nodes $\#nd$ explored in the search tree, the time $t_{\mathrm{r}}(s)$ and the gap $gap_{\mathrm{r}}\%$ at the root node, and the time $t_{1\%}(s)$ for obtaining a gap lower than 1.0%. The last three columns show the

Instance	lb	B&P							Compact MILP		
		ub	t(s)	gap%	#nd	t_r(s)	gap_r%	$t_{1\%}$(s)	ub	t(s)	gap%
20_1_7_v1_G1	52080	52080	29.67	0.00	3	12.42	0.6	12.42	52410	tl	11.6
20_1_7_v2_G1	49440	49440	83.68	0.00	3	18.70	0.4	18.70	233110	tl	79.7
25_1_7_v1_G1	60560	60560	30.36	0.00	2	20.76	0.9	20.76	60560	tl	4.1
25_1_7_v2_G1	72660	72660	14.73	0.00	1	14.73	0.0	14.73	203990	tl	65.6
40_1_7_v1_G1	100410	100410	43.69	0.00	5	18.44	0.4	18.44	124380	tl	24.1
40_1_7_v2_G1	98390	98390	6422.33	0.00	75	77.00	0.8	77.00	507090	tl	81.4
20_3_7_v1_G2	52900	53170	tl	0.51	106	323.16	0.9	323.16	-	tl	-
20_3_7_v2_G2	60120	60120	27589.73	0.00	258	235.13	0.7	235.13	-	tl	-
20_3_7_v3_G2	60450	60465	tl	0.02	257	187.09	0.8	187.09	-	tl	-
20_1_7_v1_G3	82825	82880	tl	0.07	1596	15.95	1.0	24.30	82860	tl	0.1
20_1_7_v2_G3	64675	64840	tl	0.26	1426	26.20	1.4	53.26	65170	tl	8.8
25_1_7_v1_G3	99995	100160	tl	0.17	1332	22.64	0.7	22.64	100050	tl	0.2
25_1_7_v2_G3	72660	72660	6.45	0.00	1	6.45	0.0	6.45	72660	tl	2.8
40_1_7_v1_G3	202050	202050	37.58	0.00	8	12.61	0.2	12.61	202050	tl	1.8
40_1_7_v2_G3	117677	117840	tl	0.14	692	74.49	0.8	74.49	508410	tl	81.4
20_3_7_v1_G3	71800	72325	tl	0.73	127	329.45	0.8	329.45	-	tl	-
20_3_7_v2_G3	61656	61670	tl	0.02	142	346.44	2.3	7936.81	-	tl	-
20_3_7_v3_G3	63575	63605	tl	0.05	233	156.92	2.0	717.94	-	tl	-
20_5_7_v1_G3	84554	86335	tl	2.11	122	803.60	6.0	-	-	tl	-
20_5_7_v2_G3	86915	87040	tl	0.14	131	403.44	4.9	16169.93	-	tl	-
Average			**3806.47***	**0.21**	**326**	**155.28**	**1.3**	**1381.86***			

Table 5.1: Results comparison between B&P and the Compact MILP model solved with CPLEX 12.7 on RGR instances with time limit 8 hours.

upper bound, the computational time and the gap obtained by CPLEX when solving the compact MILP model. When the value is marked with “tl”, it means that the 8 hours time limit has been reached, while the dash “-” means that the value is not available: on columns *ub* and *gap*% the dash states that a feasible solution is not found within the time limit, and on column $t_{1\%}(s)$ it states that a solution with gap lower than 1.0% is not found within the time limit. In addition, the asterisk “*” next to the values in the last line indicates that the average has been evaluated on the available values.

The results show that B&P is able to prove optimality for all the instances of groups G1, for 1 instance of group G2, and for 2 instances of group G3, with a total of 9 over 20 cases. Among these instances, we can see that for almost all the mono-activity ones, the optimal solution is quickly found exploring a few nodes of the search tree. Only 40_1_7_v2_G1 achieved optimality after 75 nodes. Concerning the unsolved instances of groups G2 and G3, B&P finds solutions with an optimality gap that does not exceed 0.73%, except for 20_5_7_v1_G3 where the final gap is 2.11%. In general, even though B&P does not prove optimality for all the instances within the time limit, it is able to find high quality integer solution. Indeed, except 20_5_7_v1_G3, in all the other instances an optimality gap lower than 1.0% is achieved within 5 hours of computational time, and within 15 minutes for most of the instances (17 over 20). In addition, in 16 cases the high quality solution is found either at the root node with the LNS heuristic or exploring only a few nodes of the search tree. Concerning the compact MILP model solved with CPLEX, we can see that none of the instances is solved within the time limit, and solutions with an optimality gap lower than

1.0% are found only for 2 instances (20_1_7_v1_G3 and 25_1_7_v1_G3). In addition, CPLEX fails in finding a feasible integer solution in all the instances with 3 and 5 activities.

RGR flexible instances. When solving this set of instances, CG solves heuristically the pricing problems starting from the daily shift selection strategy (D1). When no columns with negative reduced cost are found, CG uses strategy (D4), then (D5) and finally (D6) to prove optimality. The need of introducing further intermediate strategies to generate columns, comes from the fact that the exact resolution of one pricing problem takes 478.04 seconds in average (cf. Table 4.4). For this reason, we want to avoid solving many times the pricing problems to optimality. Indeed, in all instances, strategy (D6) was used only to prove optimality or for a few iterations. As previously, the starting slot selection strategy is fixed to (S3), that selects all feasible slots, while the configurations considered are (C1) with one break, and (C2) and (C3) with two timeslots divided respectively by a break and an interruption respectively. Due to the fact that these instances are generated from RGR instances without modifying the level of employees heterogeneity, the number of pricing problems of each instance equals the one of the corresponding RGR instances. As a consequence, CG does not make use of partial pricing.

Instance	lb	B&P							Compact MILP		
		ub	t(s)	gap%	#nd	t_r(s)	gap_r%	$t_{1\%}$(s)	ub	t(s)	gap%
20_1_7_v1_G4	47660	47760	tl	0.2	69	2190.25	0.9	2190.25	55150	tl	16.0
20_1_7_v2_G4	47440	47440	16773.14	0.0	49	753.32	0.9	753.32	245320	tl	80.7
20_3_7_v1_G4	50990	51370	tl	0.8	46	4037.52	0.9	4037.52	-	tl	-
20_3_7_v2_G4	57245	57655	tl	0.7	25	13096.05	1.9	14726.13	-	tl	-
20_3_7_v3_G4	57565	57960	tl	0.7	24	14312.07	1.2	15497.70	-	tl	-
Average				**0.5**	**43**	**6877.84**	**1.2**	**7440.98**			

TABLE 5.2: Results comparison between B&P and the Compact MILP model solved with CPLEX 12.7 on RGR flexible instances with time limit 8 hours.

The results are presented in Table 5.2. We can see that B&P is able to solve within the time limit only one of the instances. However, high quality solutions with an optimality gap lower than 1.0% are found, at the root node in 3 over 5 cases, and exploring only 2 nodes in the other cases (20_3_7_v2_G4 and 20_3_7_v3_G4). The computational time varies considerably even comparing instances with the same number of activities. This is due to the high computational effort required by the heuristic daily shift strategies (D5), and by strategy (D6) used to solve the pricing problems to optimality. Indeed, when these strategies are used only to prove that no more negative reduced cost columns exist, the computational time is relatively low (for example, 20_3_7_v1_G4), while it increases considerably when (D5) and (D6) are used even for a few iterations of CG (for example, 20_3_7_v2_G4). For the last two instances, the computational time employed at the root node to find the lower bound is almost 50% of the total time. As for the RGR instances, the compact MILP model obtains a feasible integer solution only for the mono-activity case. However, none of the instances is solved to optimality.

Real instances. When solving this set of instances, CG solves heuristically the pricing problems starting from the daily shift selection strategy (D1). Besides this strategy, the initial columns are quickly generated by considering only configuration (C1) with one timeslot. When no columns with negative reduced cost are found, CG generates daily shifts with two timeslots using configurations (C1)-(C3). Furthermore, it increases the daily shifts selection strategies as done for RGR flexible instances, considering strategy (D4), then (D5) and finally (D6) to prove optimality. CG does not generate daily shifts with three timeslots using configurations (C4)-(C7), both for practical reasons and for the fact that the use of these configurations does not lead to cost saving, as it will discuss in Table 5.5. The starting slot selection strategy is fixed to (S3), as done for the other two sets of instances. Due to the high number of employees and the high level of heterogeneity, one pricing problem need to be associated to each employee, resulting in a high number of subproblems. Partial pricing is therefore employed, and 5 subproblems giving negative reduced cost columns are solved at each iteration of CG.

Instance	lb	B&P							Compact MILP		
		ub	t(s)	gap%	#nd	t_r(s)	gap_r%	$t_{1\%}$(s)	ub	t(s)	gap%
57_11_60_le	37270	37455	tl	0.5	14	1302.92	0.9	1302.92	38880	tl	4.1
75_14_60_ll	135670	136660	tl	0.7	8	1559.51	0.7	1559.51	-	tl	-
43_15_60_lb	33015	33060	tl	0.1	18	536.43	0.2	536.43	34215	tl	3.5
38_05_60_nh	44585	44585	16483.10	0.0	30	713.74	0.9	713.74	44585	4135.67	0.0
23_05_60_nn	21095	21095	713.33	0.0	1	713.33	0.0	713.33	21095	1316.48	0.0
Average				**0.3**	**14**	**965.19**	**0.6**	**965.19**			

Table 5.3: Results comparison between B&P and the Compact MILP model solved with CPLEX 12.7 on Real instances with time limit 8 hours.

Table 5.3 shows the results for the Real instances with 60 minutes time unit. B&P is able to solve two of the instances within the time limit: 23_05_60_nn at the root node, while 38_05_60_nh after exploring 30 nodes. However, as for RGR flexible, high quality solutions with an optimality gap lower than 1.0% are found in all cases. These solutions are determined at the root node with the LNS heuristic, with an average gap of 0.6%. The instance requiring the highest time at the root node presents also the highest number of employees, that is 75. However, from these instances it is not clear which factors affect the most the performance of CG, since 43_15_60_lb has an average number of employees and the highest number of activities, but the lowest computational time at the root node. We recall that these are real instances from a fast food restaurant chain with a high degree of flexibility and heterogeneity, and there are many factors that interact and influence the performance of CG. This was not the case for RGR instances, where the number of activities was clearly the most important factor. The compact MILP model is able to find the optimal solution for the same two instances solved by B&P. For the instances 75_14_60_ll with 75 employees, no feasible solution is found within the 8 hours time limit.

Instances with 30 and 15 minutes time unit are solved neither with the compact MILP model nor with the B&P. The latter fails since CG is not able to solve the linear relaxation of the master problem, due to the extremely high degree of flexibility that results in a considerable computational effort for solving to optimality the pricing problems. These instances will be tackled through heuristic methods in the next part of the thesis.

Cost saving RGR and RGR flexible instances. As explained in Section 1.4 and as the name suggests, RGR flexible instances are similar to RGR instances, but they present a higher degree of flexibility. Indeed, in RGR we have daily shifts of 4 hours, 6 hours, and 8 hours with 1 hour of break in the middle, while no particular structure is imposed in RGR flexible instances. Timeslots have a duration between 4 and 6 hours, and two timeslots may be divided not only by a break, but also by an interruption whose duration goes from 2 to 5 hours. For example, 20_1_7_v1_G4 and 20_3_7_v1_G4 are the flexible version of 20_1_7_v1_G1 and 20_3_7_v1_G2. The following results have the goal of evaluating the advantage of a higher degree of flexibility. In the first column, Table 5.4 shows the name of the instance without specifying the group to which it belongs. The second column reports the lower bound *lb* obtained by CG when solving instances of groups G1 and G2. The last two columns show the results of the instances of group G4, that is the lower bound *lb* and the cost saving *save*% achieved. We can see that increasing the flexibility allows a cost saving that goes up to 9.3% and it is 5.5% in average. However, as previously remarked, the resolution of the pricing problems requires a high computational effort and the total time of the CG increases considerably.

Instance	G1/G2	G4	
	lb	lb	save%
20_1_7_v1_*	52080	47660	9.3
20_1_7_v2_*	49660	47440	4.7
20_3_7_v1_*	52900	50990	3.7
20_3_7_v2_*	60120	57245	5.0
20_3_7_v3_*	60450	57565	5.0
Average			**5.5**

Table 5.4: Comparison cost saving RGR instances and RGR flexible instances.

Cost saving Real instances. From the tests presented in Section 4.5 on the resolution of the pricing problem with the different combinations of starting slots and daily shifts selection heuristics, we remarked that configurations (C1)-(C3) seem to give a degree of flexibility sufficient to find the minimum reduced cost column, and according to our intuition, also the optimal value of the linear relaxation of the problem. For this reason, we solved instances with 60 minutes time unit considering only the configuration (C1), then configurations (C1)-(C3) and finally configurations (C1)-(C6). In Table 5.5 we compare the lower bound obtained by the CG. The goal is to see how increasing the degree of flexibility allows saving in terms of cost. The results show that the use of two timeslots divided by either a break or an interruption, i.e. configurations (C1)-(C3), bring to a cost saving up to 18.8% for the instance with the highest number of employees (75_14_60_ll). However, when the flexibility is further increased, the cost saving becomes negligible. Indeed, only for instance 75_14_60_ll, configurations (C1)-(C6) allows a saving of 0.6%, while for all the other instances the cost is equal. This is probably due to the use of the interruption pause, that allows enough flexibility for finding an optimal or near optimal solution. In addition, we have that schedules with three timeslots and two pauses are not appreciated by the employees, even though this type of schedules is feasible and respects all legal regulations. Indeed, by analyzing the planning validated by the managers of some clients of the company Horizontal Software, we remarked that in almost all planning, daily shifts with three timeslots were absent.

Instance	(C1)	(C1)-(C3)		(C1)-(C6)	
	lb	lb	save%	lb	save%
57_11_60_le	43345	37270	16.3	37270	0.0
75_14_60_ll	161210	135670	18.8	134820	0.6
43_15_60_lb	36370	33015	10.2	33015	0.0
38_05_60_nh	48180	44585	8.1	44585	0.0
23_05_60_nn	22115	21095	4.8	21095	0.0
Average			**11.6**		**0.1**

Table 5.5: Comparison cost saving using configurations (C1), configurations (C1)-(C3) and configurations (C1)-(C6) on Real instances.

After these considerations, we decided to consider only configurations (C1)-(C3), since the results show that they allow to find optimal or near optimal solutions, and because they generate daily shifts that are more desirable for the employees.

5.5 Conclusions

This chapter presented an exact B&P algorithm for obtaining integer solution to the multi-activity tour scheduling problem. The method employs CG for solving the linear relaxation of the master problem at each node of the search tree. Besides using the DA heuristic presented in Chapter 3 to stabilize CG, other accelerating strategies have been presented. The branching rule is based on the original variables x^i_{ja} of the MILP model described in Chapter 2. Furthermore, a LNS heuristic is employed to find an upper bound at each node. Computational results have been performed on the three set of instances introduced in Section 1.4 (RGR, RGR flexible, and Real instances). The main feature of these sets is their variety. Indeed, the first set comes from the literature and a particular structure is imposed to the daily shifts. The second set is a more flexible variant of the first, and removes some restrictions on the daily shift structure. The third set is the one with the highest degree of flexibility, and the instances come from a fast food restaurant chain where employees have different availabilities, skills and contract regulations. The results show that B&P is able to prove optimality for some instances (12 over a total of 40). However, high quality solutions with an optimality gap lower than 1.0% are found within the time limit, except for one instance of the RGR set, where B&P achieves an optimality gap of 2.11%.

When increasing the degree of flexibility (RGR flexible and Real instances), a solution with an optimality gap lower than 1.0% is found by exploring only few nodes of the search tree, probably due to the existence of many equivalent optimal solutions. However, the higher flexibility is the cause of the considerable computational effort needed to solve the pricing problems, resulting in a high computational time for CG to prove optimality. Indeed, only instances with 60 minutes time unit of the Real set are solved. The other instances with 30 minutes and 15 minutes time unit are tackled with heuristic methods in the next part of the thesis.

Part III

Heuristic methods

Introduction to Part III

Part III of this thesis focuses on heuristic methods for solving the multi-activity tour scheduling problem. In the last decades, heuristic methods have been successfully used for attacking a variety of difficult combinatorial optimization problems that arise in many practical areas. They have been designed to deal with problems where classical exact methods have failed to solve them and to be efficient, either because they are difficult in their own right, or because the practical real-world instances make them intractable. Heuristics have often been used for solving personnel scheduling problems. The popularity of these methods lies in the fact that they determine reasonable good feasible solutions within a limited amount of computational time, while exact techniques may not return any feasible solution for a long time. In addition, most heuristics are relatively easy to implement and they allow to incorporate and exploit problem specific information. However, these methods cannot demonstrably converge to an optimal solution. Different heuristic methods have been proposed in the literature for dealing with (multi-activity) shift and tour scheduling, based on local search (Musliu et al. (2004) and Meisels and Schaerf (2003)), large neighborhood search combined with formal languages (Quimper and Rousseau (2009)), and tabu search (Dahmen and Rekik (2015)). More recently, sophisticated matheuristics have been proposed. In general, these methods either exploit mathematical programming techniques in a heuristic framework, giving to mathematical programming approaches the effectiveness which characterize heuristics, or exploit the mathematical programming models in the customization of a heuristic (Boschetti et al. (2009)). In the recent literature, we find Gérard et al. (2016) who propose a matheuristic method based on B&P where the pricing problem is solved heuristically, and Hernández-Leandro et al. (2018) who solve exactly a restricted set covering problem whose columns have been identified through the Lagrangian relaxation of the initial problem.

In the following we present the heuristic methods developed mostly to deal with the large-scale real instances that could not be tackled through the B&P method previously presented. Part III is organized as follows: Chapter 6 presents four different heuristics. The first is a large neighborhood search (LNS) which makes use of the pricing solving method presented in Chapter 4 for exploring the neighborhood. The second combines CG and LNS that work on and exchange sets of columns in a primal-dual framework. The third consists of a hybrid method that makes use of a greedy heuristic, tabu search and LNS. Finally, a diving heuristic that partially explores the search tree in a B&P framework is presented.

Chapter 6

Heuristic Methods

This chapter presents four heuristic methods for finding good feasible solutions of the multi-activity tour scheduling problem. These methods aim at overcoming the difficulty of the exact B&P algorithm in finding a feasible solution of the large-scale instances. The chapter is organized as follows: Section 6.1 describes a LNS that iteratively destroys and repairs an initial solution in order to improve it. Section 6.2 presents a primal-dual heuristic that makes use of CG to get a lower bound, and LNS to get an upper bound and a feasible integer solution by working on and exchanging sets of columns. Section 6.3 introduces a hybrid heuristic that combines a greedy heuristic to determine an initial solution satisfying workload, a tabu search to integrate a particular class of constraints, and LNS to obtain a feasible solution. Section 6.4 presents a diving heuristic that partially explores a B&P search tree for quickly finding a feasible solution. Finally, Section 6.5 closes the chapter by comparing the performances of the different heuristics.

6.1 Large Neighborhood Search

This section describes a Large Neighborhood Search (LNS) heuristic, which has recently shown good results in solving transportation and scheduling problems. We first recall the general idea of LNS. Then, Section 6.1.1 details the proposed method which makes use of the techniques presented in Chapter 4 for generating schedules. Finally we report some computational results in Section 6.1.2 and conclusions in Section 6.1.3.

The large neighborhood search method was first proposed by Shaw (1998), and it allows to improve the quality of an initial solution, through an iterative process that destroys and repairs part of the current solution (cf. Algorithm 1). Destroying part of the current solution corresponds to the definition of its neighborhood, that needs to be explored in order to build an improving solution. The reconstruction phase likely leads to a better solution when the neighborhood is well chosen. For instance, too constrained neighborhoods are not recommended, since they may rebuild the current solution. Unlike local search, LNS considers large neighborhood, providing the opportunity to escape local optima. However, particular care needs to be taken with high dimension neighborhood. A complete exploration is very likely to find an improving solution, but it usually requires a considerable computational effort. For this reason, partial exploration can be considered.

The concept of destroying and repairing is well adapted to problems which naturally can be decomposed into a master problem that covers a number of activities or clients, and a set of subproblems that need to satisfy some given constraints. As a consequence, problems well suited for the Dantzig-Wolfe decomposition are good candidates for applying LNS heuristics. For instance, Prescott-Gagnon et al. (2009) implement a LNS algorithm for the vehicle routing problem with time windows, where the destructor operator chooses some clients that are removed from the current solution. Then, in the reconstruction phase, the problem restricted to the selected neighborhood is modeled as set partitioning, where the variables correspond to feasible routes. A heuristic B&P method is then employed for the exploration of the neighborhood. Quimper and Rousseau (2009) propose a LNS algorithm for the multi-activity shift scheduling, where the destructor operator selects an employee and removes all the activities he is performing. Formal languages are then used to explore the neighborhood and build an improving schedule that better covers the demand. We refer the reader to Ahuja et al. (2002) and Ropke and Pisinger (2006), who define and survey the class of very large scale neighborhood search method to which LNS belongs.

Algorithm 1 presents the pseudo-code for a generic LNS. Variable X_i is the initial solution, which can be built by the algorithm or it can be given as input. The best solution found during the search is stored in X_b, while variables X_c and X_t are respectively the current and the temporary solution, which can be discarded or accepted to be the new current solution. The destructor operator is represented by the function `Destroy(·)`, and it returns a copy of the current solution X_c that has been partially destroyed. The obtained solution is then reconstructed by the function `Repair(·)`, and a feasible solution built from the destroyed one is obtained. In lines 2, 3 and 4, the initial solution is built and it is used to initialize both current and best variables. The current solution is destroyed and repaired in lines 6 and 7. Line 8 checks whether the temporary solution X_t is better than the best solution known, which is updated if necessary in line 9. The temporary solution may be accepted and become the new current solution or it may be rejected (line 11). All steps defined in lines 6-11 are repeated until a stopping condition is satisfied (line 5). Different stopping criteria may be defined, and the most commonly used are based on limiting the number of iterations or the computational time.

The proposed method takes inspiration from the LNS proposed by Quimper and Rousseau (2009) and from the greedy heuristic proposed by Gérard et al. (2016). It differs from these works for the definition of the neighborhood to explore, due to the fact the multi-activity tour scheduling problem considered is different, and for the objective function used to guide the search.

6.1.1 The method

The proposed LNS makes use of the pricing solving method proposed in Chapter 4 to generate new feasible schedules, both to define the initial solution and to explore the neighborhood

Algorithm 1: Generic LNS algorithm

```
1  Data: $X_i$, $X_b$, $X_c$, $X_t$;
2  $X_i \leftarrow$ Initialize();
3  $X_c \leftarrow X_i$;
4  $X_b \leftarrow X_i$;
5  while stopping condition do
6      $X_t \leftarrow$ Destroy($X_c$);
7      $X_t \leftarrow$ Repair($X_t$);
8      if $X_t$ is better than $X_b$ then
9          $X_b \leftarrow X_t$;
10     end if
11     $X_c \leftarrow$ Select($X_c$, $X_t$);
12 end while
13 return $X_b$;
```

when the current solution needs to be reconstructed. In the following we detail the main elements defining the proposed LNS: the *destructor operator*, the *constructor operator* and how to define the *initial solution.*

6.1.1.1 Destructor operator

The destructor operator chooses one employee $i \in I$, and removes the current schedule assigned. As a consequence, all activities performed by this employee are deleted and some of them may be uncovered. The obtained partial solution preserves the schedules of the other employees, and the neighborhood of this solution is defined by the set of all feasible schedules that can be assigned to employee i. Various strategies may be defined for choosing employee i, and each of them results in a different neighborhood. This is the case when the employees are heterogeneous, with different skills, availabilities and contract regulations. Indeed, each employee has a specific set of feasible schedules and therefore a specific neighborhood. However, in case of homogeneity, the neighborhood is the same for all the employees, but the objective function used may be different, since it is defined starting from the current schedule of employee i. We defined three different strategies for choosing the employee and, therefore, for defining the neighborhood of the current solution:

(E1) the first strategy simply selects one employee after the other in a round-robin way;

(E2) the second strategy orders the employees in a decreasing way considering the number of under covered slots in which they are available and not working. The idea is to try to define new schedules that better cover the under assigned slots;

(E3) the third strategy orders the employees in a decreasing way considering the number of over covered slots in which they work. The idea is to redistribute their working hours trying not to cover the over assigned slots.

6.1.1.2 Constructor operator

The destructor operator determines the neighborhood of the current partial solution by selecting an employee $i \in I$ and removing his schedule. The constructor operator explores this neighborhood in order to try to find an improving solution. In our LNS, the exploration of the neighborhood consists of finding a new schedule for the employee selected by the destructor operator, and this is done by means of the pricing algorithm presented in Chapter 4. In order to guide the search and find a schedule that improves the current solution, we need to define the costs $\{u_{ja}\}_{j\in J, a\in A}$ and the cost v_i for the objective function defined in (2.82):

$$c_p^i - \sum_{j\in J}\sum_{a\in A}\delta_{pja}^i u_{ja} - v_i.$$

More in details, u_{ja} represents the cost of covering activity $a \in A$ in slot $j \in J$, and it is defined using the residual work demand r_{ja} of the current partial solution. The residual work demand corresponds to the not satisfied demand taking into account the schedules in the current partial solution. More precisely, r_{ja} is strictly positive if activity a is under covered in slot j, while it is strictly negative if it is over covered. We define costs u_{ja} as follows:

$$u_{ja} = \begin{cases} \underline{c}_{ja}, & \text{if } r_{ja} > 0, \\ -\bar{c}_{ja}, & \text{otherwise,} \end{cases} \tag{6.1}$$

where $\underline{c}_{ja}$ and $\bar{c}_{ja}$ are respectively the under and over coverage costs. The definition of u_{ja} differs from the one of Gérard et al. (2016) since the authors use the ratio between residual and total demands. However, this definition does not take into account the cost of under and over coverage, and this can be a disadvantage, mostly when there is a considerable difference between the two costs. Quimper and Rousseau (2009) do not give details on how the costs are selected.

After evaluating u_{ja} for each activity a and for each slot j, we can define v_i, that represents the cost of the schedule removed from employee i. Let $p \in P^i$ be the index of the schedule $\{\delta_{pja}^i\}_{j,a}$ removed from employee i. Cost v_i is defined as follows:

$$v_i = -\sum_{j\in J}\sum_{a\in A}\delta_{pja}^i u_{ja}. \tag{6.2}$$

At this point, the constructor operator is able to explore the neighborhood of the current partial solution guided by the objective function (2.82) to find a new schedule s_i. The goal is to determine s_i with minimum (reduced) cost. If s_i has negative reduced costs, it means that it improves the current solution. We remark that if the neighborhood is completely explored, the reduced cost of s_i generated cannot be strictly positive, since the removed schedule belongs to the neighborhood and can be regenerated.

Due to the fact that the neighborhood contains all the feasible schedules that can be assigned to an employee, its size may be large. For this reason, the LNS makes use of partial exploration by means of the heuristic strategies presented in Section 4.4.2, that select only a subset of feasible daily shifts generated to be combined into a feasible schedule. The main idea is to use three different strategies with increasing degree of exploration, searching for an improving schedule first in a restricted part of the neighborhood, then in a larger part and finally in the complete neighborhood. The complete exploration is performed only when the computational effort required is limited.

More in detail, the neighborhood is explored first with strategy (D1), that aims at finding quickly a feasible schedule by selecting the most restricted set of feasible daily shift (i.e., in each day, the best daily shift is selected for each value of daily working hours). If the generated schedule has a positive reduced cost, it does not improve the current solution. In this case, it is discarded and a larger subset of daily shifts is considered by using strategy (D3), which select diversified daily shifts for each day and each possible value of daily working hours. Finally, if even this strategy fails in finding a negative reduced cost schedule, the complete exploration of the neighborhood is performed, when possible, using strategy (D6). As previously said, the reduced cost of the new schedule s_i can be either strictly negative or equal to zero when the complete exploration is performed. In the first case, s_i improves the current solution and it is assigned to the employee selected in the destruction phase. In the second case, s_i is a schedule equivalent or equal to the removed one. Concerning the starting slot selection strategies, it has been fixed to (S3) to avoid restricting too much the neighborhood.

6.1.1.3 Initial solution

The LNS algorithm needs an initial solution to which it applies iteratively destuctor and reconstructor operators. When this solution is not given as input, it builds for each employee a feasible schedule starting from the empty planning. One possible way is to randomly select a schedule for each employee, as done in Quimper and Rousseau (2009). However, this procedure may lead to bad quality initial solution. Gérard et al. (2016) generates a schedule for each employee defining costs based on the ratio between residual and total demand, as done all along their algorithm. As previously said, this does not consider the under and over coverage costs. For this reason we build an initial solution using costs $\{u_{ja}\}$ defined in (6.1). At the beginning, the residual demand r_{ja} is equal to the demand, and it is updated every time a new schedule is added. In addition, in the initialization phase, LNS uses strategy (D3) for generating quickly a feasible schedule. Indeed, strategy (D1) fails in some cases to find a feasible schedule due to the fact that the exploration of the neighborhood is considerably reduced. Even with strategy (D3) we do not have the guarantee that a feasible solution is found. However, in all tests performed this situation has never occurred and the reason relies on the fact that the strategy selects diversified daily shifts that cover different slots of the day.

The LNS algorithm is formalized in Algorithm 2. Variables X_i, X_b, X_c and X_t are respectively the initial, best, current and temporary solutions. Variables u and v are the costs used by the constructor operator, while I is the set of employees. The algorithm starts by finding an initial solution X_i, which is assigned both to X_c and to X_b. Then, at each iteration, the employees are sorted according to one of the strategies (E1), (E2) or (E3) given in input (line 6). For each of the employee i in the sorted list I, the destructor operator finds a temporary

Algorithm 2: LNS heuristic

```
1  Data: $X_i$, $X_b$, $X_c$, $X_t$, $u$, $v$, $I$;
2  $X_i \leftarrow$ Initialize();
3  $X_c \leftarrow X_i$;
4  $X_b \leftarrow X_i$;
5  while $iter < iter\_max$ and $X_t \neq X_c$ do
6      $I \leftarrow$ Sort ($I$);
7      foreach employee $i \in I$ do
8          $X_t \leftarrow$ Destroy($X_c$,$i$);
9          $(u, v) \leftarrow$ UpdateCosts ($X_t$);
10         $X_t \leftarrow$ Repair($X_t$, $u$, $v$);
11         if $cost(X_t) \leq cost(X_c)$ and $X_t \neq X_c$ then
12             $X_c \leftarrow X_t$;
13         end if
14     end foreach
15     if $cost(X_t) < cost(X_b)$ then
16         $X_b \leftarrow X_t$;
17     end if
18 end while
19 return $X_b$;
```

solution X_t by removing the schedule of i (line 8), costs u and v are evaluated based on the residual demand of X_t (line 9), and the constructor operator repairs X_t by finding a new feasible schedules for employee i (line 10). The current solution X_c is updated with X_t if the cost has decreased, or, in case of equivalent solutions, if X_c and X_t are different. Replacing X_c with equivalent solutions aims at escaping local optima and it can be seen as a restart. The LNS algorithm stops as soon as the maximum number of iterations has been reached, or if the temporary solution X_t is equal to the current solution X_c, meaning that the LNS has found a local optimum and it is not able to escape it.

6.1.2 Computational results

This section shows the computational results of the LNS previously described. The goal is to evaluate the performance of the LNS with the three employees selection strategies (E1), (E2) and (E3) presented in Section 6.1.1.1. The algorithm has been coded in C# and tests have been performed on a 64-bit Windows operating system with 977 GB of RAM and 16 processors (only one core is used) Intel Core running at 2.00 GHz.

Experiments have been done on the three sets of instances RGR, RGR flexible and Real presented in Section 1.4. The variety of these instances, mostly in the degree of flexibility, forces us to set up the LNS according the class considered. The only element that needs to be tuned is the constructor operator, due to the fact that the computational effort required to explore the neighborhood of a solution differs considerably from class to class. For the three sets of instances, the corresponding setting will be presented.

Preliminary results suggest that after four iterations in average the LNS was not able to improve the current solution X_c or it was exiting due to the fact that the temporary solution X_t was equal to X_c. For this reason, the maximum number of iterations of the LNS have been fixed to 5. This means that the schedule of each employee is removed and replaced at most 5 times, and the destructor and constructor operators are called 5×the total number of employees times.

RGR instances. The LNS has been set as described in Section 6.1.1.2: the constructor operator starts exploring the neighborhood and looking for an improving schedule using strategy (D1), then strategy (D3), and finally strategy (D6).

Table 6.1 compares the performance of the three employee selection strategies (E1), (E2) and (E3). It shows the name of the instance in the first column, and the lower bound *lb* known in the second column. For each employee selection strategy, the table shows the computational time $t(s)$ in seconds and the optimality gap *gap*% of the integer solution found by the LNS. In addition, boldface indicates the lowest time and the best gap obtained by the three strategies in each instance.

Instance	lb	(E1)		(E2)		(E3)	
		t(s)	gap%	t(s)	gap%	t(s)	gap%
20_1_7_v1_G1	52080	453.49	**0.2**	537.10	**0.2**	**258.27**	0.6
20_1_7_v2_G1	49440	849.93	**0.0**	974.15	**0.0**	**745.75**	1.2
25_1_7_v1_G1	60560	**570.83**	**0.0**	652.91	**0.0**	614.07	**0.0**
25_1_7_v2_G1	72660	**1067.76**	**0.3**	1127.52	**0.3**	1147.40	**0.3**
40_1_7_v1_G1	100410	1073.50	**0.0**	985.73	**0.0**	**795.47**	**0.0**
40_1_7_v2_G1	98390	1370.10	0.3	1385.63	0.2	**1030.20**	**0.0**
20_3_7_v1_G2	52900	**139.56**	2.5	169.66	2.5	182.56	**2.3**
20_3_7_v2_G2	60120	**192.04**	**1.5**	218.16	**1.5**	321.20	2.1
20_3_7_v3_G2	60450	223.85	**0.9**	**197.36**	**0.9**	236.03	**0.9**
20_1_7_v1_G3	82825	**11.95**	3.0	16.19	2.3	26.07	**1.9**
20_1_7_v2_G3	64675	31.46	**2.0**	20.27	2.5	**19.95**	2.1
25_1_7_v1_G3	99995	22.15	**2.2**	32.52	**2.2**	**10.27**	2.4
25_1_7_v2_G3	72660	**20.77**	**1.9**	24.26	**1.9**	22.07	2.2
40_1_7_v1_G3	202050	27.57	**0.4**	**17.55**	**0.4**	27.84	**0.4**
40_1_7_v2_G3	117677	22.76	**4.6**	14.66	4.7	**14.53**	4.8
20_3_7_v1_G3	71800	28.57	**1.1**	**22.40**	**1.1**	30.01	1.2
20_3_7_v2_G3	61656	33.24	**3.0**	**20.25**	**3.0**	27.31	3.1
20_3_7_v3_G3	63575	**20.61**	2.7	44.90	**2.0**	29.19	2.4
20_5_7_v1_G3	84554	40.91	9.0	**26.25**	**8.2**	29.10	9.6
20_5_7_v2_G3	86915	29.73	**6.0**	**23.76**	6.1	31.20	6.8
Average		**311.54**	**2.1**	**325.56**	**2.0**	**279.92**	**2.2**

TABLE 6.1: Comparison employee selection strategies (E1), (E3) and (E2) on RGR instances.

The last line of the table shows the average values, which reveals similar behavior of the three strategies both in terms of time (around 300 seconds) and gap (around 2%). If we analyze instance by instance, there is not a clear dominance of one strategy over the others. Indeed, even though strategy (E1) has an average gap sightly higher than (E2), it finds the best solution with the lowest optimality gap in 15 cases, against the 14 cases of (E2) and the 8 cases of (E3). Concerning the computational time, (E1) and (E3) are the fastest in 7 cases, while (E2) is the fastest in 6 cases.

These results reveal that none of the strategies dominates when considering average and individual gaps and computational times, and it seems that specific employees sorting strategies do not have general benefits.

RGR flexible instances. The LNS has been set as follows: due to the high computational effort required by a complete exploration of the neighborhood and to the fact that a new schedule needs to be defined for each employee multiple times, the constructor operator does not employ strategy (D6). It starts using strategy (D1), and if an improving schedule is not found, it uses strategy (D3). Therefore, for these instances, only the partial exploration of the neighborhood is performed.

As previous, Table 6.2 compares the performance of the three employee selection strategies (E1), (E2) and (E3), showing the name of the instance in the first column, the lower bound known in the second column, and the computational time $t(s)$ and the optimality gap $gap\%$ for each strategy.

Instance	lb	(E1)		(E2)		(E3)	
		t(s)	gap%	t(s)	gap%	t(s)	gap%
20_1_7_v1_G4	47660	**217.07**	**4.1**	234.83	4.5	221.40	6.8
20_1_7_v2_G4	47440	**107.77**	3.6	122.22	3.4	226.79	**2.6**
20_3_7_v1_G4	50990	**215.44**	7.3	220.99	7.3	225.85	**7.2**
20_3_7_v2_G4	57245	**158.03**	**1.9**	174.57	**1.9**	223.46	2.6
20_3_7_v3_G4	57565	170.89	**1.6**	183.53	**1.6**	**138.51**	2.2
Average		**173.84**	**3.7**	**187.23**	**3.7**	**207.20**	**4.3**

Table 6.2: Comparison employee selection strategies (E1), (E3) and (E2) on RGR flexible instances.

The values in the last line of the table show similar average behaviors of strategies (E1) and (E2), while strategy (E3) has the worst performance. However, when we analyze instance by instance, (E3) obtains the best solution in 2 cases as (E2), while (E1) is the best in 3 cases. In addition (E1) is the fastest in 4 cases over 5. Even though (E1) seems to have the best performance, it does not dominates the other two strategies when considering individual computational times and gaps.

Real instances. The LNS has been set as follows: for the instances with 60 minutes and 30 minutes time unit, the constructor operator is tuned as done for RGR flexible instances, i.e. it starts using strategy (D1), and if an improving schedule is not found, it uses strategy (D3). For the instances with 15 minutes time unit, the number of feasible daily shifts drastically increases, and the computational time for generating a schedule is prohibitive if the neighborhood is not sufficiently restricted. In addition, the number of employees goes up to 75 and schedules need to be generated multiple times for each employee. For these reasons, the slot selection strategy and the daily shift strategy has been modified in order to restrict the neighborhood: as slot selection strategy, we do not use (S3) but we use (S2), that allows to select only a subset of slots based on the demand profile; in addition, the constructor operator uses only the daily shift selection strategy (D1).

Table 6.3 shows the name of the instance and the corresponding lower bound in the first and second column respectively. When a lower bound is not known (instances with 30 minutes and 15 minutes time unit), the third column reports the best upper bound known, used to evaluate the optimality gap. By abuse of language, we use the term “optimality gap” even though the best upper bound may not be the optimal value. The last six columns show the computational time $t(s)$ and the optimality gap *gap*% for the three employee selection strategies.

Instance	lb	best ub	(E1)		(E2)		(E3)	
			t(s)	gap%	t(s)	gap%	t(s)	gap%
57_11_60_le	37270	-	**31.45**	5.0	50.76	**3.3**	36.71	5.4
75_14_60_ll	135670	-	37.20	1.2	30.67	1.2	**28.56**	**1.1**
43_15_60_lb	33015	-	**15.10**	**3.8**	18.94	4.0	17.52	4.5
38_05_60_nh	44585	-	**17.21**	**1.3**	30.14	1.5	25.92	**1.3**
23_05_60_nn	21095	-	**6.83**	3.2	9.77	3.2	7.34	**2.8**
Average (60min)			**21.56**	**2.9**	**28.06**	**2.6**	**23.21**	**3.0**
57_11_30_le	-	74315	**316.76**	2.2	437.47	1.8	386.72	**1.7**
75_14_30_ll	-	269530	**260.53**	**1.4**	393.08	**1.4**	262.52	1.5
43_15_30_lb	-	64540	125.29	2.0	124.67	1.7	**89.15**	**1.4**
38_05_30_nh	-	82055	**119.55**	1.5	217.76	1.6	181.19	**1.0**
23_05_30_nn	-	45090	**56.69**	**1.7**	60.84	**1.7**	69.64	**1.7**
Average (30min)			**175.76**	**1.7**	**246.76**	**1.6**	**197.84**	**1.4**
57_11_15_le	-	149040	526.39	1.8	**353.24**	**0.0**	354.21	1.5
75_14_15_ll	-	540115	583.51	**0.8**	**456.82**	**0.8**	466.58	**0.8**
43_15_15_lb	-	130210	154.48	0.9	**154.10**	0.9	154.61	**0.4**
38_05_15_nh	-	159305	425.75	1.5	**318.53**	**1.2**	324.83	1.5
23_05_15_nn	-	93060	122.05	**0.9**	**81.90**	**0.9**	83.89	**0.9**
Average (15min)			**362.44**	**1.2**	**272.92**	**0.8**	**276.82**	**1.0**

TABLE 6.3: Comparison employee selection strategies (E1), (E3) and (E2) on Real instances.

Results on instances with 60 minutes time unit reveal that the three strategies achieve similar gaps in a comparable computational time. Even though (E1) has an average gap of 2.9% (against the average gap of 2.6% of (E2)), it obtains the best gap in 2 cases over 5 (against the 1 case of (E2)), and it is the fastest in 4 cases. Concerning the instances with 30 minutes time unit, the best performance in terms of gap is obtained by (E3), with an average gap of 1.4% and the best gap achieved in 4 cases over 5. Finally, in the instances with 15 minutes time unit, (E2) finds a solution with the best gap in 4 cases with the lowest computational time, while strategies (E1) and (E3) obtain the best gap respectively in 2 and 3 cases. In general, the results on this set of instances show that the three strategies are able to find solutions with similar optimality gap. Indeed, the average gaps on all instances are 1.9%, 1.7% and 1.8% for strategies (E1), (E2) and (E3) respectively, with a standard deviation of 1.2%, 1.1% and 1.4% respectively. In addition, (E1), (E2) and (E3) achieves the best gaps in 6, 7 and 10 cases, and they are the fastest in 8, 5 and 2 cases respectively.

These results confirm the conclusion previously drawn on the fact that none of the strategies dominates the others when considering average and individual gaps and computational times.

Complete exploration vs. partial exploration on RGR instances. The following tests have the goal of comparing the performance of the LNS when the complete exploration of the neighborhood is performed using strategies (D1)+(D3)+(D6) (previous results on RGR instances), with the performance of the LNS when only partial exploration is performed using strategies (D1)+(D3).

Table 6.4 shows the name of the instance in the first column and the lower bound known in the second column. The computational time $t(s)$ and the optimality gap *gap*% of the LNS with the possibility of completely exploring the neighborhood using strategies (D1)+(D3)+(D6) are reported respectively in the third and fourth columns, while the last two columns show the time and the gap of the LNS with only partial exploration using strategies (D1)+(D3). Since no employee selection strategy has been found dominant by the previous experimentations, these tests have been performed using strategy (E1).

The results show similar optimality gaps, but computational times much lower when only partial exploration (D1)+(D3) is employed. This is evident in instances of groups G1 and G2, where all time periods are feasible as starting slot for a daily shift, and the complete exploration of the neighborhood may take up to some seconds. Even though a partial exploration seems to be a good compromise between computational time and optimality gap, we will see that a more intense effort looking for an improving solution can be rewarding, mostly when the lower bound is evaluated by the algorithm and it is used to determine the quality of the integer solution found (cf. the primal-dual heuristic in Section 6.2, Table 6.6).

6.1.3 Conclusions

This section presented a LNS heuristic for solving the multi-activity tour scheduling addressed in this thesis. The proposed method starts from an initial solution and iteratively removes the schedule assigned to one employee (destructor operator) and builds a new schedule (constructor operator) minimizing an objective function defined using residual demand

Instance	lb	(D1)+(D3)+(D6)		(D1)+(D3)	
		t(s)	gap%	t(s)	gap%
20_1_7_v1_G1	52080	453.49	0.2	42.78	0.2
20_1_7_v2_G1	49440	849.93	0.0	41.33	0.0
25_1_7_v1_G1	60560	570.83	0.0	51.73	0.0
25_1_7_v2_G1	72660	1067.76	0.3	73.86	0.3
40_1_7_v1_G1	100410	1073.50	0.0	78.81	0.0
40_1_7_v2_G1	98390	1370.10	0.3	84.89	0.4
20_3_7_v1_G2	52900	139.56	2.5	66.66	2.5
20_3_7_v2_G2	60120	192.04	1.5	34.72	1.5
20_3_7_v3_G2	60450	223.85	0.9	12.08	1.0
20_1_7_v1_G3	82825	11.95	3.0	14.09	3.0
20_1_7_v2_G3	64675	31.46	2.0	30.21	2.0
25_1_7_v1_G3	99995	22.15	2.2	23.60	2.2
25_1_7_v2_G3	72660	20.77	1.9	18.15	1.9
40_1_7_v1_G3	202050	27.57	0.4	28.82	0.4
40_1_7_v2_G3	117677	22.76	4.6	30.13	6.3
20_3_7_v1_G3	71800	28.57	1.1	30.61	1.1
20_3_7_v2_G3	61656	33.24	3.0	29.57	3.0
20_3_7_v3_G3	63575	20.61	2.7	24.82	2.8
20_5_7_v1_G3	84554	40.91	9.0	41.11	9.0
20_5_7_v2_G3	86915	29.73	6.0	17.06	6.4
Average		**311.54**	**2.1**	**38.75**	**2.2**

TABLE 6.4: Comparison strategy (D6) and (D3) with employee selection strategy (E1) on RGR instances.

and under and over coverage costs. Three different strategies have been proposed for selecting the employee in the destruction phase. However, computational results show that they have similar performances in terms of computational time and optimality gap, and no dominant strategy has been highlighted. For this reason, when LNS will be used, employees will be selected in a round-robin way, i.e., using strategy (E1). We also compare complete and partial explorations of the neighborhood, showing that the complete exploration does not improve considerably the optimality gap. However, we will see in Section 6.2 that a more intense effort may be rewarding.

6.2 Primal-dual heuristic

This section proposes a primal-dual (PD) heuristic, which makes use of CG to get a lower bound, and LNS to get upper bounds and feasible integer solutions. The two methods are embedded in the overall approach by working on and exchanging sets of columns. Even though the method is heuristic and it does not guarantee to find an optimal solution, it provides a lower bound that allows to evaluate the quality of the integer solution found. The section is organized as follows: the PD heuristic is described in Section 6.2.1, computational results are shown in Section 6.2.2, while conclusions are presented in Section 6.2.3.

6.2.1 The method

The PD approach proposed is a heuristic whose main components are CG (cf. Section 5.1) and LNS (cf. Section 6.1). The first solves the linear relaxation of the master problem (2.69)-(2.73) and it obtains both a lower bound and a fractional solution. The second improves an initial integer solution received in input from the rounding of the fractional solution, and it obtains both an upper bound and an integer solution. The advantage of this heuristic concerns the fact that valid lower and upper bounds are computed, and they can be compared to evaluate the quality of the integer solution found. This characteristic distinguishes PD from the other heuristics presented in this thesis, and from many heuristics that we can find in the literature. Indeed, these heuristics usually tries to improve an integer feasible solution of the problem without having an idea of how far this solution is from the optimal one, stopping when the maximum number of iterations is achieved, a time limit is exceeded or the best solution known does not improve.

Algorithm 3: PD heuristic

```
1  **Data:** $X_b$, $X_t$, $X_f$, $gap$, $lb$, $RMP$;
2  $gap \leftarrow 1$;
3  **while** $gap > 0.01$ *and* $iter < iter\_max$ **do**
4      $(X_f, lb) \leftarrow$ CG $(RMP)$;
5      $X_t \leftarrow$ Round $(X_t)$;
6      $X_t \leftarrow$ LNS $(X_t)$;
7      **if** $cost(X_t) < cost(X_b)$ **then**
8          $X_b \leftarrow X_t$;
9          $gap \leftarrow (cost(X_b) - lb)/lb$;
10     **end if**
11     $RMP \leftarrow$ Update $(RMP, X_t)$;
12 **end while**
13 return $X_b$;
```

Algorithm 3 presents the pseudo-code of the PD heuristic. Variables X_b, X_t and X_f represent respectively the best, temporary and fractional solutions, while *gap* and *lb* are the optimality gap and the lower bound obtained at each iteration of PD. Finally, RMP contains the columns of the reduced master problem. Function `CG(·)` solves the linear relaxation of the master problem to optimality and it returns a fractional solution X_f and the optimal value *lb* (line 4). We refer to Section 5.1 for more details on CG. In line 5, X_f is rounded by the function `Round(·)`, which returns a temporary integer solution X_t. The rounding is performed by selecting the column with the highest value for each employee. Function `LNS(·)` in line 6 takes as input X_t and it improves it by iteratively destructing and constructing the solution. We refer to Section 6.1 for more details on how LNS works. The returned solution X_t is then compared with the best solution known X_b (lines 8, 9 and 10), and both X_b and *gap* are updated in case of improvement. Finally, in line 11, function `Update(·)` updates the columns of RMP as follows: it evaluates the reduced costs of all columns and it keeps in RMP only half of the columns with reduced cost lower than 10^{-12}; then, it adds the columns of the integer solution X_t. Keeping some columns with small reduced cost has the goal of speeding up the convergence of CG from the second iteration of PD. We do not keep all of them to avoid generating only few columns in the next iteration of PD. Indeed, preliminary results has shown that CG converges in some cases after few iterations when RMP keeps all columns with reduced cost lower than 10^{-12}, and the rounded solutions does not change. In order to further speed up CG, the stopping criteria of CG is modified from the second iteration of PD. CG is not performed until optimality is proven and no negative reduced cost column is found, but it is stopped as soon as the gap between the value of RMP and lb is lower than 1%, to avoid the tailing-off effect. PD heuristic iteratively calls CG and LNS until the optimality gap is lower than 1%, or the maximum number of iterations $iter\_max$ $(= 10)$ is exceeded.

6.2.2 Computational results

This section shows the computational results of the PD heuristic described. The algorithm has been coded in C# and tests have been performed on a 64-bit Windows operating system with 977 GB of RAM and 16 processors (only one core is used) Intel Core running at 2.00 GHz. CPLEX 12.7 has been used as LP solver. Experiments have been done on the three sets of instances RGR, RGR flexible and Real presented in Section 1.4.

RGR instances. Computational results on RGR instances are shown in Table 6.5. The first and the second columns show respectively the name of the instance and the corresponding lower bound known. The next five columns report the results of the heuristic B&P proposed by Restrepo et al. (2016). More in detail, ub, $t(s)$ and $gap\%$ are respectively the upper bound, the computational time in seconds and the optimality gap, while $t_{\mathrm{r}}(s)$ and $gap_{\mathrm{r}}\%$ are the computational time and the gap evaluated at the root node. The last seven columns show the results of the PD heuristic, that is the upper bound ub, the final computational time $t(s)$, the final optimality gap $gap\%$, the number of iterations it, the computational time $t_1(s)$ and optimality gap $gap_1\%$ after the first iteration, and the percentage of time $t_{\mathrm{CG}}\%$ employed by CG.

Remark. For some instances, the results of B&P-RGR are marked with "-", since it turned out from a discussion with one of the author of Restrepo et al. (2016) that there was an error in the grammar generating the daily shifts. From an analysis of the integer solution obtained, it seemed that the grammar was generating only daily shifts with one transition between activities in each timeslot. This error does not affect the instances with only one activity, since no transition can be done. In addition, we obtain the same lower bound in 4 instances with 3 activities (20_3_7_v2_G2, 20_3_7_v3_G2, 20_3_7_v2_G3 and 20_3_7_v3_G3) due to the existence of equivalent solutions. For all the other instances with multiple activities, we obtain a lower bound that is lower than the one presented in the paper by Restrepo et al. (2016), and a comparison is not possible.

Instance	lb	B&P-RGR					PD						
		ub	t(s)	gap%	t_r(s)	gap_r%	ub	t(s)	gap%	it	t_1(s)	gap_1%	t_{CG}%
20_1_7_v1_G1	52080	52080	21.31	0.0	21.31	0.0	52410	12.42	0.6	1	12.42	0.6	87.29
20_1_7_v2_G1	49440	49660	31.00	0.4	31.00	0.4	49660	18.70	0.4	1	18.70	0.4	92.15
25_1_7_v1_G1	60560	60560	16.69	0.0	16.69	0.0	61110	20.76	0.9	1	20.76	0.9	77.95
25_1_7_v2_G1	72660	72660	6.32	0.0	6.32	0.0	72660	14.73	0.0	1	14.73	0.0	90.24
40_1_7_v1_G1	100410	100850	10.77	0.4	10.77	0.4	100850	18.44	0.4	1	18.44	0.4	83.90
40_1_7_v2_G1	98390	98940	90.52	0.6	90.52	0.6	99160	77.00	0.8	1	77.00	0.8	53.41
20_3_7_v1_G2	52900	-	-	-	-	-	53355	323.16	0.9	1	323.16	0.9	87.84
20_3_7_v2_G2	60120	60120	1735.07	0.0	733.69	3.6	60560	142.93	0.7	1	142.93	0.7	84.96
20_3_7_v3_G2	60450	60780	1507.83	0.5	697.80	2.6	60935	187.09	0.8	1	187.09	0.8	77.60
20_1_7_v1_G3	82825	82860	3.66	0.0	3.66	0.0	83520	17.39	0.8	2	15.95	1.0	63.43
20_1_7_v2_G3	64675	65280	5.91	0.9	5.91	0.9	65170	57.42	0.8	4	26.20	1.4	30.54
25_1_7_v1_G3	99995	100380	3.47	0.4	3.47	0.4	100710	22.64	0.7	1	22.64	0.7	84.89
25_1_7_v2_G3	72660	73100	1.61	0.6	1.61	0.6	72660	6.45	0.0	1	6.45	0.0	74.89
40_1_7_v1_G3	202050	202250	1.38	0.1	1.38	0.1	202380	12.61	0.2	1	12.61	0.2	76.45
40_1_7_v2_G3	117677	118170	10.46	0.4	10.46	0.4	118570	74.49	0.8	1	74.49	0.8	72.83
20_3_7_v1_G3	71800	-	-	-	-	-	72370	329.45	0.8	1	329.45	0.8	93.98
20_3_7_v2_G3	61656	62125	1408.06	0.8	730.60	5.8	63010	1220.14	2.2	10	346.44	2.3	80.75
20_3_7_v3_G3	63575	63950	1370.42	0.6	686.61	5.6	64180	576.74	0.9	8	156.92	2.0	79.75
20_5_7_v1_G3	84554	-	-	-	-	-	88770	2982.18	5.0	10	803.60	6.0	92.47
20_5_7_v2_G3	86915	-	-	-	-	-	89325	1657.43	2.8	10	403.44	4.9	87.37
Average								**388.61**	**1.0**	**3**	**150.67**	**1.3**	**78.63**

TABLE 6.5: Results comparison between B&P heuristic implemented by Restrepo et al. (2016) and the PD heuristic on RGR instances.

The results on the group of instances G1 show that both methods find a solution with an optimality gap lower than 1% at the root node (B&P-RGR) or after one iteration (PD), and the computational time is comparable. Concerning group G2, we remark that both methods achieve a high quality solution. However, PD terminates in one iteration, while B&P-RGR needs to explore the search tree. Indeed, the gaps at the root node are 3.6% and 2.6% in the two instances that we can compare (20_3_7_v2_G2 and 20_3_7_v3_G2). As a result, the computational time employed by PD is halved compared to B&P-RGR. The last group G3 presents different behaviors on the mono-activity and the multi-activity instances. When only one activity is considered, both methods achieve an optimality gap lower than 1%. However, B&P-RGR terminates at the root node, while PD needs more than one iteration in 2 over 6 cases. In addition, the computational time is higher. The reason relies on the fact that our CG uses increasing daily shift strategies (cf. Section 5.4), that may help to speed up the convergence of CG on large-scale instances or instances with a high pricing problem resolution effort, but it may slow CG on small instances (note that some of the instances are solved in less than 2 seconds by B&P-RGR). In the multi-activity case, PD terminates for optimality gap in 2 instances over 5, and one of them takes only one iteration. However, the gap of all the unsolved instances is lower or equal than 5%. Comparison with B&P-RGR can be done only on the two instances 20_3_7_v2_G3 and 20_3_7_v3_G3: a gap lower than 1% is achieved in both of them by B&P-RGR, while only in the second one by PD with a lower computational time; in the first instance PD achieves a gap of 2.2%. If we compare the gap and the time of PD after one iteration ($t_1(s)$ and $gap_1\%$) with the ones of B&P-RGR at the root node ($t_r(s)$ and $gap_r\%$), we note that PD finds a solution with a halved gap in a halved computational time. In general, PD finds an integer solution with a proven optimality gap lower than 1% in 17 over 20 instances, after only one iteration in

14 of them. The 3 unsolved instances present a gap that is lower or equal than 5%. In addition, the gap obtained after the first iteration never exceeds 6%.

In Section 6.1.2, we have compared complete and partial exploration of the neighborhood when LNS is performed (cf. Table 6.4). The results have shown that the two strategies achieve solutions with similar optimality gaps. However, partial exploration allows to decrease considerably the computational time in many cases. After these first results, we have done analogous experiments on PD, meaning that PD is performed first with complete exploration in the LNS (PD (D1)+(D3)+(D6)), then it is performed with partial exploration in the LNS (PD (D1)+(D3)).

Instance	lb	PD (D1)+(D3)+(D6)						PD (D1)+(D3)					
		ub	t(s)	gap%	it	t_1(s)	gap_1%	ub	t(s)	gap%	it	t_1(s)	gap_1%
20_1_7_v1_G1	52080	52410	12.42	0.6	1	12.42	0.6	52410	19.23	0.6	1	19.23	0.6
20_1_7_v2_G1	49440	49660	18.70	0.4	1	18.70	0.4	49660	31.00	0.4	1	31.00	0.4
25_1_7_v1_G1	60560	61110	20.76	0.9	1	20.76	0.9	61110	22.95	0.9	1	22.95	0.9
25_1_7_v2_G1	72660	72660	14.73	0.0	1	14.73	0.0	72660	15.48	0.0	1	15.48	0.0
40_1_7_v1_G1	100410	100850	18.44	0.4	1	18.44	0.4	100850	19.47	0.4	1	19.47	0.4
40_1_7_v2_G1	98390	99160	77.00	0.8	1	77.00	0.8	99160	39.89	0.8	1	39.89	0.8
20_3_7_v1_G2	52900	53355	323.16	0.9	1	323.16	0.9	53355	249.09	0.9	1	249.09	0.9
20_3_7_v2_G2	60120	60560	142.93	0.7	1	142.93	0.7	60670	155.56	0.9	1	155.56	0.9
20_3_7_v3_G2	60450	60935	187.09	0.8	1	187.09	0.8	60935	111.57	0.8	1	111.57	0.8
20_1_7_v1_G3	82825	83520	17.39	0.8	2	15.95	1.0	83520	27.58	0.8	2	24.98	1.0
20_1_7_v2_G3	64675	65170	57.42	0.8	4	26.20	1.4	65170	46.81	0.8	4	29.70	1.4
25_1_7_v1_G3	99995	100710	22.64	0.7	1	22.64	0.7	100710	24.56	0.7	1	24.56	0.7
25_1_7_v2_G3	72660	72660	6.45	0.0	1	6.45	0.0	72660	5.83	0.0	1	5.83	0.0
40_1_7_v1_G3	202050	202380	12.61	0.2	1	12.61	0.2	202380	12.44	0.2	1	12.44	0.2
40_1_7_v2_G3	117677	118570	74.49	0.8	1	74.49	0.8	118570	38.69	0.8	1	38.69	0.8
20_3_7_v1_G3	71800	72370	329.45	0.8	1	329.45	0.8	72370	276.07	0.8	1	276.07	0.8
20_3_7_v2_G3	61656	63010	1220.14	2.2	10	346.44	2.3	62635	972.87	1.6	10	278.58	2.3
20_3_7_v3_G3	63575	64180	576.74	0.9	8	156.92	2.0	64365	733.26	1.2	10	169.56	2.0
20_5_7_v1_G3	84554	88770	2982.18	5.0	10	803.60	6.0	89745	2776.31	6.1	10	816.67	6.1
20_5_7_v2_G3	86915	89325	1657.43	2.8	10	403.44	4.9	89675	1521.46	3.2	10	434.41	5.2
Average			**388.61**	**1.0**	**3**	**150.67**	**1.3**		**355.01**	**1.1**	**3**	**138.79**	**1.3**

TABLE 6.6: Results comparison between PD with complete exploration (D1)+(D3)+(D6) in LNS and PD with partial exploration (D1)+(D3) in LNS on RGR instances.

Table 6.6 shows the results obtained. In most of the instances, the two strategies find a solution with optimality gap lower than 1% in comparable computational time. We remark that the time employed by the complete exploration on groups G1 and G2 is much lower compared to the one presented in Table 6.4. This is due to the fact that a lower bound is evaluated by the PD heuristic, and it is used to stop the LNS as soon as a solution with an optimality gap lower than 1% is found. The last three instances show the advantage we may have with a complete exploration of the neighborhood. Indeed, in one case (20_3_7_v3_G3) partial exploration is not able to terminate within the maximum number of iterations, and it finds a solution with a gap 1.2%, while complete exploration takes 8 iterations to find a solution with a gap of 0.8%. In addition, in the last two instances (20_5_7_v1_G3 and 20_5_7_v2_G3), both final gap *gap*% and gap after the first iteration gap_1% are lower when complete exploration is performed.

RGR flexible instances. Computational results on RGR flexible instances are shown in Table 6.7. As previous, the first two columns show the name of the instance and the lower bound known, while the next seven columns report the results of the PD heuristic, that is the upper bound *ub*, the final computational time $t(s)$, the final optimality gap $gap\%$, the number of iterations *it*, the computational time $t_1(s)$ after the first iteration, the optimality gap $gap_1\%$ after the first iteration, and the percentage of time $t_{CG}\%$ spent by CG in the overall resolution time.

Instance	lb	PD						
		ub	t(s)	gap%	it	t_1(s)	gap_1%	t_{CG}%
20_1_7_v1_G4	47660	48070	2190.25	0.9	1	2190.25	0.9	94.7
20_1_7_v2_G4	47440	47880	753.32	0.9	1	753.32	0.9	97.9
20_3_7_v1_G4	50990	51485	4037.52	0.9	1	4037.52	0.9	98.0
20_3_7_v2_G4	57245	57795	13801.92	0.9	2	13096.05	1.9	98.0
20_3_7_v3_G4	57565	58095	15326.60	0.9	4	14312.07	1.2	97.4
Average			**7221.92**	**0.9**	**2**	**6877.84**	**1.2**	**97.2**

Table 6.7: Results of PD heuristic on RGR flexible instances.

Table 6.7 shows that PD is able to find a solution with a gap lower than 1% for all the instances, and it takes only one iteration in 3 over 5 cases. Concerning the computational time, we remark that it varies considerably from instance to instance, going from some minutes to some hours. This is due to the high computational effort required to solve the pricing problems to optimality, and to the fact that CG uses increasing daily shift strategies (cf. Section 5.4). As soon as one strategy does not find a negative reduced cost column, a less restricting strategy is selected for finding profitable columns. If the strategies requiring a high effort (for example (D5) and (D6)) are used only to prove optimality, the total computational time is relatively low (instance 20_3_7_v1_G4). On the contrary, if these strategies find negative columns, they are employed in the next iterations of CG. As consequence, the computational time increases considerably, even though these strategies are used for few iterations. The last column $t_{CG}\%$ of the table shows that in average 97.2% of the total time is employed by CG (65% of this time is used only to prove optimality at the first iteration of PD).

Real instances. Computational results on RGR flexible instances are shown in Table 6.8. As previous, the first and the second columns show the name of the instance and the lower bound, while the last seven columns report the results of the PD heuristic, that is the upper bound *ub*, the final computational time $t(s)$, the final optimality gap $gap\%$, the number of iterations *it*, the computational time $t_1(s)$ and optimality gap $gap_1\%$ after the first iteration, and the percentage of time $t_{CG}\%$ employed by CG.

The results are presented for instances with 60 minutes time unit, since CG was not able to converge within a time limit of 4 hours for instances with 30 and 15 minutes time unit, due to the extremely high degree of flexibility and the computational effort needed to prove

Instance	lb	PD						
		ub	t(s)	gap%	it	t_1(s)	gap_1%	t_{CG}%
57_11_60_le	37270	37620	1302.92	0.9	1	1302.92	0.9	99.2
75_14_60_ll	135670	136660	1559.51	0.7	1	1559.51	0.7	99.8
43_15_60_lb	33015	33075	536.43	0.2	1	536.43	0.2	99.4
38_05_60_nh	44585	45025	713.74	0.9	1	713.74	0.9	99.2
23_05_60_nn	21095	21095	713.33	0.0	1	713.33	0.0	99.5
Average			**965.19**	**0.6**	**1**	**965.19**	**0.6**	**99.4**

Table 6.8: Results of PD heuristic on Real instances.

optimality. Table 6.8 shows that PD is able to find a solution with an optimality gap lower than 1% in all the instances with only one iteration. In addition, optimality is proven for instance 23_05_60_nn, where PD achieves a gap equal to 0.0%. The computational time is in average 965.19 seconds, most of which (99.4%) is employed by CG. The reason relies on the fact that most of the time is used in CG to prove optimality, as previously remarked for RGR flexible instances.

6.2.2.1 Primal-dual heuristic with time limit

The PD heuristic iteratively calls CG and LNS and it gets both lower and upper bounds. The results previously presented show that most of the computational effort is employed by CG, and the solving time may increase considerably with the degree of flexibility. As a consequence, some instances (Real with 30 and 15 minutes time unit) were not solved. For this reason, we propose a variant of the PD heuristic, by imposing a time limit (= 180 seconds) to both CG and LNS, and a total time limit (= 1200 seconds) to PD. In addition, when function `Update(·)` is called to update the columns of the reduced master problem RMP (cf. Algorithm 3, line 11), all columns with reduced cost lower than 10^{-12} are kept in RMP, in order to further speed up CG, and to hope that it converges and finds the lower bound within the time limit of 180 seconds in the next iteration.

In this section we call PD the primal-dual heuristic without time limit, while we call PD-tl the variant with time limit. We remark that PD-tl does not guarantee to find a lower bound, since CG may stop before proving optimality. However, in some cases CG converges after some iterations of PD-tl. We also remark that LNS may terminates before the time limit of 180 seconds, due to the maximum number of iterations or due to the fact that the temporary solution found is equal to the current solution (cf. Algorithm 2, line 5).

RGR instances. Table 6.9 shows the results on RGR instances. Since CG converges in less than 180 seconds for most of them, we report only the 5 instances for which CG terminates due to the time limit. When the optimality gap *gap*% is marked with (*), it means that CG is able to converge not at the first iteration of PD-tl but at the next iterations, finding the lower bound *lb* and proving that the optimality gap *gap*% is actually the value reported.

Instance	lb	PD-tl			
		ub	t(s)	gap%	gap_1%
20_3_7_v1_G2	52900	53380	1058.74	0.9	0.9
20_3_7_v1_G3	71800	72155	350.49	*0.5	0.5
20_3_7_v2_G3	61656	62580	1139.44	*1.5	3.0
20_5_7_v1_G3	84554	90050	1160.50	6.5	10.5
20_5_7_v2_G3	86915	89005	1127.24	2.4	3.6
Average			**967.28**	**2.4**	**3.7**

Table 6.9: Results of PD heuristic with time limit 1200 seconds on RGR instances.

The results show that PD-tl is able to find a lower bound in 2 cases over 5. In particular, the computational time of instance 20_3_7_v1_G3 is much lower than the other instances. This is due to the fact that CG converges at the second iteration of PD-tl within the time limit of 180 seconds. Therefore, the gap of 0.5%, that was already found after the first iteration, can be proven and the algorithm can stop for optimality. We have a different situation with the instance 20_3_7_v2_G3, where the lower bound is found, but PD-tl is not able to determine a solution with a gap lower than 1% and it stops only when the total time limit is reached. In the other 3 cases, CG is never able to converge and to find the lower bound. Therefore, PD-tl terminates without being able to prove the optimality gap. We remark that for the first instance 20_3_7_v1_G2, the computational time of PD-tl is triple the time of PD (1058.74 against 323.16 seconds). The reason relies on the the fact that PD finds the lower bound and stops for optimality gap lower than 1%, while PD-tl continues looking for the lower bound and the best integer solution until it reaches the time limit.

RGR flexible instances. Table 6.10 shows the results on RGR flexible instances. Due to the high computational effort needed to prove optimality, CG does not converge within the time limit of 180 seconds and it does not find a lower bound. We recall that solving one of the pricing problem to optimality takes in average 478.04 seconds (cf. Section 4.5, Table 4.4), and the convergence of CG within the time limit is out of reach.

The results show that the optimality gap achieved by PD-tl is always lower than 5%, and in 4 over 5 cases it does not exceed 1.1%. In addition, PD-tl finds a solution with a null gap for one of the instances (20_1_7_v2_G4). We also remark that PD-tl improves the initial gap found after the first iteration of 12.5%, going from an average value of 1.6% to an average value of 1.4%.

Real instances. Table 6.11 shows the results on the Real instances. The first column and the second columns show the name of the instance and the lower bound. When the latter is not available (instances with 30 minutes and 15 minutes time unit), the best upper

Instance	lb	PD-tl			
		ub	t(s)	gap%	gap_1%
20_1_7_v1_G4	47660	48045	1138.94	0.8	0.9
20_1_7_v2_G4	47440	47440	1141.60	0.0	0.0
20_3_7_v1_G4	50990	53065	1048.02	4.1	4.1
20_3_7_v2_G4	57245	57885	954.18	1.1	1.3
20_3_7_v3_G4	57565	58180	1051.30	1.1	1.6
Average			**1066.81**	**1.4**	**1.6**

Table 6.10: Results of PD heuristic with time limit 1200 seconds on RGR flexible instances.

Instance	lb	best ub	PD-tl			
			ub	t(s)	gap%	gap_1%
57_11_60_le	37270	-	37430	1075.97	0.4	0.7
75_14_60_ll	135670	-	135670	1029.93	0.0	0.0
43_15_60_lb	33015	-	33100	1073.77	0.3	0.4
38_05_60_nh	44585	-	44585	986.35	0.0	0.0
23_05_60_nn	21095	-	21095	1061.85	0.0	0.0
Average (60min)				**1045.57**	**0.1**	**0.2**
57_11_30_le	-	74315	74315	988.30	0.0	1.7
75_14_30_ll	-	269530	270530	972.57	0.4	0.6
43_15_30_lb	-	64540	64540	1046.34	0.0	0.2
38_05_30_nh	-	82055	82055	1048.74	0.0	0.1
23_05_30_nn	-	45090	45090	1028.67	0.0	0.2
Average (30min)				**1016.92**	**0.1**	**0.6**
57_11_15_le	-	149040	151570	1095.21	1.7	6.1
75_14_15_ll	-	540115	540630	1165.11	0.1	16.1
43_15_15_lb	-	130210	130210	1028.58	0.0	0.9
38_05_15_nh	-	159305	159305	1112.16	0.0	1.5
23_05_15_nn	-	93060	93060	959.80	0.0	0.0
Average (15min)				**1072.17**	**0.4**	**4.9**

Table 6.11: Results of PD heuristic with time limit 1200 seconds on Real instances.

bound known is reported in the third column. The last four columns show the results of the PD-tl heuristic, that is the upper bound ub, the computational time $t(s)$, the optimality gap $gap\%$, and the optimality gap $gap_1\%$ after the first iteration. By abuse of language, we use the term "optimality gap" even though the best upper bound may not be the optimal.

Concerning the instances with 60 minutes time unit, the results show that PD-tl is able to find an integer solution with an optimality gap lower than 0.5% in all 5 cases, and an initial gap that does not exceed 0.7%. However, CG is not able to converge finding a lower bound within the time limit, and, therefore, the gap cannot be proven. Concerning the instances with 30 minutes and 15 minutes time unit, PD-tl finds a solution with a gap that does not exceed 0.4%, except for one instance (57_11_15_le), where the gap is 1.7%. In addition, in 7 over 10 cases PD-tl has a null gap, meaning that this heuristic finds the solution with the best value known.

6.2.3 Conclusions

In this section we proposed a PD heuristic that makes use of CG to get lower bounds and fractional solutions, and LNS to get upper bounds and feasible integer solutions. The advantage of this method concerns the fact that the quality of the solution found can be evaluated, since both lower and upper bounds are available. However, PD is subject to the convergence of CG. As a consequence, PD is not able to solve instances with a very high degree of flexibility (Real instances with 30 minutes and 15 minutes time unit). In order to obtain feasible solutions also for these instances, we propose a variant of the PD heuristic (PD-tl), that imposes a time limit to both CG and LNS. Even though the optimality gap cannot be proven, PD-tl achieves the best upper bound known in most of the instances not solved by PD.

6.3 Hybrid heuristic

This section presents a hybrid heuristic for solving the multi-activity tour scheduling problem addressed. This method combines a greedy heuristic, tabu search (TS) and large neighborhood search (LNS). We have seen in Section 6.1 how LNS can be employed to find a feasible solution to the multi-activity tour scheduling problem. The algorithm starts with an empty planning and it builds the initial solution based on the residual demand (cf. Section 6.1.1.3). However, an initial solution can be given as input. For instance, in the PD heuristic presented in Section 6.2, the LNS takes as initial solution the rounding of the optimal fractional solution of the master problem, obtained by means of CG. In this section we describe a heuristic based on TS for obtaining an initial solution, which is used as starting point by the LNS. The intuition behind is the following: we remarked that the common feature of most of the legal constraints is the duration, especially on consecutive slots. Indeed, many constraints impose lower and upper bounds on the assignment of some entity (task, timeslot, break, interruption or daily shift). For instance, we have minimum and maximum duration on activities, on breaks and on interruptions. Furthermore, we have minimum and maximum duration on timeslots, which is nothing but the consecutive working hours constraint, and minimum and maximum duration of daily shifts, which corresponds to the constraint on the amplitude of the working day. In the following, we will refer to these constraints as *duration constraints*. A TS heuristic focuses on this remark to find a solution satisfying this class of constraints.

The section is organized as follows: Section 6.3.1 describes the hybrid heuristic developed for computing an integer feasible solution, and it details the three components, i.e. the greedy heuristic, the TS and the LNS. Computational results are shown in Section 6.3.2, and conclusions are presented in 6.3.3.

6.3.1 The method

The proposed hybrid heuristic essentially combines a greedy heuristic, TS and LNS. Starting from an initial solution satisfying workload obtained with the greedy heuristic, TS aims at integrating a particular class of the legal constraints by keeping the demand satisfied. Then, LNS completely repairs all schedules making them feasible, and it aims at minimizing the total cost of the planning.

6.3.1.1 Greedy heuristic

The greedy algorithm builds a solution assigning employees to activities in order to satisfy only workload constraints. Initially, no employee is assigned and therefore the planning is empty. Activities are treated one by one and available employees are assigned in slots where the considered activity is required.

Algorithm 4: Greedy heuristic

```
$X_i \leftarrow [\,]$;
foreach activity $a$ do
    foreach employee $i$ do
        foreach slot $j$ do
            if $b_{ja} > 0$ and $i$ is available in $j$ then
                $X_i \leftarrow$ Assign($i$,$j$,$a$);
                $b_{ja} - 1$;
            end if
        end foreach
    end foreach
end foreach
return $X0$;
```

The greedy heuristic considers only workload requirements and it assigns employees to activities in each slot. This is done without checking if the assignments cause the violation of other constraints. If there are enough available employees, the resulting planning satisfies exactly the demand without any under or over coverage.

6.3.1.2 Tabu search

Tabu search is a heuristic procedure which goes back to Glover (1989) and has been used in many applications such as Al-Turki et al. (2001), Burke et al. (2006), Chiarandini et al. (2000) and De Bruecker et al. (2014). In our hybrid method, TS follows the greedy heuristic and it receives an initial solution satisfying workload constraints. As previously explained, this solution is built considering only the demand in each slot and no check is done on the other constraints. TS aims at integrating a particular class of constraints, which imposes minimum and maximum bounds on the assignment of some entity. With the term "entity" we refer to either a task, a timeslot, a break, an interruption or a daily shift. The constraints belonging to this class, called *duration constraints*, are the following:

(L1) *activity duration*: it imposes bounds on the duration of a task;
(L2) *consecutive working time*: it imposes bounds on the duration of a timeslot;
(L3) *break and interruption duration*: it imposes bounds on the duration of a pause;
(L5) *amplitude of the working day*: it imposes bounds on the duration of a daily shift.

We introduce an objective function to measure how far a solution is from satisfying the duration constraints. This function is given by the sum of all violations to constraints on the duration of tasks (2.9), timeslots (2.20), breaks (2.37), interruptions (2.38) and daily shifts (2.49). Let us define the set E of all entities, that is $E = K \cup T \cup B \cup C \cup S$ where K, T, B, C and S are respectively the set of tasks, timeslots, breaks, interruptions and daily shifts. Furthermore, for each entity $e \in E$, let $x_e \in \{0, 1\}$ be the binary variable equal to 1 if e is performed, and let $d_e \in \mathbb{R}^+$ be the positive variables equal to its duration. For instance, if e is a task $k \in K$, we have that x_e corresponds to variable x_k^K, which is equal to 1 if task k is performed, 0 otherwise (cf. Section 2.1.1.3). In addition d_e is equal to the duration of task k, given by the sum $\sum_{j \in J}(y_{kj}^K - z_{kj}^K)$ (cf. constraint (2.9)). Finally, for each entity $e \in E$, let l_e and u_e be respectively the lower and upper bound of the duration of entity e. For instance, if e is a timeslot $t \in T$, we have that l_e and u_e correspond to the lower bound l_w^c and the upper bound u_w^c on the consecutive working time (i.e. the duration of the timeslot).

We say that entity e violates the corresponding duration constraint if it is performed, i.e. $x_e = 1$, and its duration d_e is lower than l_e or greater than u_e. Therefore, we can define its violations as follows:

$$v(x_e, d_e) = \max\{0;\, l_e x_e - d_e;\, d_e - u_e x_e\}. \tag{6.3}$$

The TS heuristic aims at finding the best quality solution, with the lowest total violation to duration constraints, satisfying workload requirements. Therefore, the goal is the minimization of the total violations to duration constraints. The violations $v(x_e, d_e)$ are weighted using multipliers λ_e, which are iteratively updated.

$$\min \sum_{e \in E} \lambda_e \, v(x_e, d_e). \tag{6.4}$$

In order to minimize the total violation (6.4) of the duration constraints, we use a basic TS method combined with intensification and diversification techniques. In the following we describe the features of our TS heuristic.

Neighborhood. The neighborhood of a solution is defined by the operator called `Swap`. Given a subset of consecutive slots $J' = \{j, j+1, \dots\}$ (this subset can contain also one single slot) and two employees i_1 and i_2, the solution obtained after applying `Swap` $(i_1, J', a_1) \rightarrow$

(i_2, J', a_2) is equal to the current solution except that employees i_1 and i_2 exchange their activities in all the slots j in J'. The neighborhood of a solution s is built according to the following: we select the most violated constraint, $N(s)$ consists of all the solutions we can achieve by applying a `Swap` move on a subset J' of slots of the selected constraint. We remark that, for each slot j in J', the two employees embedded in the move, exchange the activities in j, ensuring that all the solutions in the neighborhood keep on satisfying workload constraints. Regarding the neighborhood exploration, we tested both best and first improvement. The first one consists in looking for the `Swap` that gives the best objective function improvement, while the second one consists in accepting the first `Swap` that improves the objective function. Due to the high number of evaluations needed, the first improvement gives better results considering both solving time and quality of solutions.

Tabu list. We employ a dynamic tabu list. We fix minimum l_{min} and maximum l_{max} lengths. The tabu list length changes between l_{min} and l_{max} according to the evolution of the objective function: it increases when the best solution known does not improve after $it\_tabu (= 10)$ iterations, while it decreases when the best solution known improves. Regarding the tabu list update, we use a FIFO policy.

Multipliers λ_e update. Multipliers λ_e are iteratively updated according the evolution of the corresponding violation $v(x_e, d_e)$ after applying the `Swap` move. Fixing $\delta \in (0, 1)$ as small constant value, $\lambda_e = (1 + \delta)\lambda_e$ if the violation increases and $\lambda_e = (1 - \delta)\lambda_e$ if the violation decreases.

Algorithm 5: Tabu search heuristic

```
$X_c \leftarrow X_i$;
$X_b \leftarrow X_i$;
for ($t$ = 1,.., $div\_max$) do
    while ($cost\_best > 0$ and $it < it\_max$) do
        $X_c \leftarrow$ ExploreNeighborhood($X_c$);
        if (f($X_c$) < f($X_b$)) then
            $X_c \leftarrow$ Intensification($X_c$);
            $X_b \leftarrow X_c$;
        else
            UpdateTabuList();
        end if
    end while
    $X_c \leftarrow$ Diversification($X_c$);
    $it\_max \leftarrow 2 * it\_max$;
end for
return $X_b$;
```

Intensification. The basic TS is combined with intensification. When an improving `Swap` $(i_1, J', a_1) \rightarrow (i_2, J', a_2)$ is found, the neighborhood is deeply explored, and moves in adjacent slots are evaluated.

Diversification. We also combine the basic TS with diversification. In our heuristic, when the best solution cannot be improved any more using the basic TS with intensification, we employ a perturbation operator to destruct the obtained local optimum. Starting from the best solution known, we apply `Swap` move in all slots in which we have a violated duration constraint. To be more precise, for each slot we first evaluate the swap moves with all other employees, then we select the move that leads the objective function to the lowest deterioration. As a result, the new solution preserves part of the best solution feasibility and differs where duration constraints are violated. Then, the basic TS with intensification is restarted. Diversification is performed $div\_max(= 5)$ times.

Stopping criteria. Many stopping conditions can be used for TS, such as the fixed numbers of iterations, the maximum number of iterations without any improvement of the objective function and the total amount of computational time. Since the basic TS is integrated with a perturbation operator, it stops when the best solution cannot be improved within a given number of iteration $it\_max(= 100)$.

6.3.1.3 Large neighborhood search

The last component of the hybrid heuristic is the LNS. We recall that LNS iteratively destroys and repairs part of the current solution with the goal of improving it. At this step we have a solution that satisfies exactly workload constraints and it has hopefully a low violation to duration constraints. In the first place, the LNS is employed to integrate completely all the constraints that are still violated, by removing (destructor operator) the current schedule of each employee, and building a new feasible schedule (constructor operator). Then, the LNS iteratively applies destructor and constructor operators to improve the current solution minimizing the total cost of the planning, that considers both under and over coverage. We refer to Section 6.1 for the details of the LNS framework.

6.3.2 Computational results

This section shows the computational results of the hybrid heuristic described. The algorithm has been coded in C# and tests have been performed on a 64-bit Windows operating system with 977 GB of RAM and 16 processors (only one core is used) Intel Core running at 2.00 GHz. Experiments have been done on the three sets of instances RGR, RGR flexible and Real presented in Section 1.4. Due to the different degrees of flexibility of these instances, the LNS component requires to be set up. For more details on how it has been tuned, we refer to the Section 6.1 that describes the LNS (cf. Section 6.1.2).

RGR instances. Computational results on RGR instances are shown in Table 6.12. The first and the second columns report respectively the name of the instance and the corresponding lower bound known. The next five columns show the results of the heuristic B&P method proposed by Restrepo et al. (2016). More in detail, ub, $t(s)$ and $gap\%$ are

Instance	lb	B&P-RGR					Hybrid		
		ub	t(s)	gap%	t_r(s)	gap_r%	ub	t(s)	gap%
20_1_7_v1_G1	52080	52080	21.31	0.0	21.31	0.0	52190	507.42	0.2
20_1_7_v2_G1	49440	49660	31.00	0.4	31.00	0.4	49440	476.94	0.0
25_1_7_v1_G1	60560	60560	16.69	0.0	16.69	0.0	60560	424.31	0.0
25_1_7_v2_G1	72660	72660	6.32	0.0	6.32	0.0	72660	561.10	0.0
40_1_7_v1_G1	100410	100850	10.77	0.4	10.77	0.4	100410	594.81	0.0
40_1_7_v2_G1	98390	98940	90.52	0.6	90.52	0.6	98830	1380.68	0.4
20_3_7_v1_G2	52900	-	-	-	-	-	54100	600.76	2.3
20_3_7_v2_G2	60120	60120	1735.07	0.0	733.69	3.6	60780	182.65	1.1
20_3_7_v3_G2	60450	60780	1507.83	0.5	697.80	2.6	62020	222.30	2.6
20_1_7_v1_G3	82825	82860	3.66	0.0	3.66	0.0	84650	345.39	2.2
20_1_7_v2_G3	64675	65280	5.91	0.9	5.91	0.9	68340	106.32	5.7
25_1_7_v1_G3	99995	100380	3.47	0.4	3.47	0.4	103280	129.25	3.3
25_1_7_v2_G3	72660	73100	1.61	0.6	1.61	0.6	72860	99.34	0.3
40_1_7_v1_G3	202050	202250	1.38	0.1	1.38	0.1	202930	365.31	0.4
40_1_7_v2_G3	117677	118170	10.46	0.4	10.46	0.4	123780	269.01	5.2
20_3_7_v1_G3	71800	-	-	-	-	-	72750	827.19	1.3
20_3_7_v2_G3	61655.5	62125	1408.06	0.8	730.60	5.8	64155	72.25	4.1
20_3_7_v3_G3	63575	63950	1370.42	0.6	686.61	5.6	65320	69.20	2.7
20_5_7_v1_G3	84553.9	-	-	-	-	-	95460	193.04	12.9
20_5_7_v2_G3	86915	-	-	-	-	-	92515	245.65	6.4
Average								**383.65**	**2.6**

TABLE 6.12: Results comparison between B&P heuristic implemented by Restrepo et al. (2016) and the hybrid heuristic on RGR instances.

respectively the upper bound, the computational time and the optimality gap, while $t_r(s)$ and $gap_r\%$ are the computational time and the gap evaluated at the root node. The last three columns show the results of the hybrid heuristic, that is the upper bound ub, the computational time $t(s)$ and the optimality gap $gap\%$.

The results show that for instances of group G1, the hybrid heuristic is able to find a solution with a gap lower than 0.4%, and in 4 cases over 6 the upper bound equals the lower bound, resulting in a null gap. However, the computational time is much higher than the one of B&P-RGR. For these instances, most of the time is employed by LNS, due to the fact that daily shift selection strategies (D1)+(D3)+(D6) are used to find an improving schedule when constructing the partial solution, and complete exploration of the neighborhood may be performed.

Concerning the instances of group G2, we have that B&P-RGR finds a solution with a gap lower than 0.5% for both instances where the comparison can be done, while the gaps achieved by the hybrid heuristic are 1.1% and 2.6%. However, if we compare the gaps with the one obtained at the root node by B&P-RGR ($gap_r\%$), the hybrid heuristic is able to find a solution with a halved or an equal gap, and the computational time is lower.

Finally, from the instances of group G3 we remark a deterioration in the gaps obtained by the hybrid heuristic on the mono-activity case, if we compare them to G1. The reason relies on the fact that group G3 restricts the feasible starting slots of the daily shifts, while groups G1 and G2 allow the daily shifts to start in every time period of the day. During the TS phase, the hybrid heuristic does not take into account the predefined starting slots and, in some cases, this may lead to a bad quality initial solution for the LNS. However, the average gap obtained in these 6 instances is 2.8% and the highest gap is 5.7%. The computational time is much higher than the one of B&P-RGR. This is due to the fact that TS takes in average 90% of the total computational time for integrating the duration constraints. The time employed by TS is not only affected by the number of employees and activities, but also, and in a strong way, by the demand profile. When the workload constraints require a high number of employees, the solution found by the greedy heuristic is dense and the number of `Swap` movements leading to an improvement is limited, mostly in the mono-activity context. Therefore, TS needs to perform a deep exploration of the neighborhood to find an improving movement. Analogously to group G2, the hybrid heuristic finds a solution with a gap lower than the one obtained at the root node by B&P-RGR in the instances of group G3 with 3 activities.

In general, the hybrid heuristic finds a feasible solution with an average optimality gap of 2.6% and an average computational time of 383.65 seconds. In addition, the gap is lower than 1% and 5% respectively in 40% and 90% of the instances. The highest gaps (12.9% and 6.4%) are obtained when 5 activities and predefined starting slots are considered.

RGR flexible instances. Computational results on RGR flexible instances are shown in Table 6.13. The first and the second columns report the name of the instance and the lower bound known, while the last three columns report the upper bound *ub*, the computational time $t(s)$ and the optimality gap *gap*% obtained by the hybrid heuristic.

Instance	lb	Hybrid		
		ub	t(s)	gap%
20_1_7_v1_G4	47660	50230	385.02	5.4
20_1_7_v2_G4	47440	47440	345.58	0.0
20_3_7_v1_G4	50990	52890	543.39	3.7
20_3_7_v2_G4	57245	58080	385.69	1.5
20_3_7_v3_G4	57565	58375	368.56	1.4
Average			**405.65**	**2.4**

Table 6.13: Results hybrid heuristic on RGR flexible instances.

The results show that the proposed heuristic finds a feasible solution with an average optimality gap of 2.4% and an average computational time of 405.65 seconds. The highest gap is 5.4% while the lowest gap is 0.0%, meaning that a feasible solution with a value that equals the lower bound has been found. The computational time of the hybrid heuristic is not affected by the number of employees since all the instances consider 20 employees. However, it is affected by the number of activities and, most important, by the demand profile, as previously remarked. Indeed, we can see that some instances with 3 activities are solved in less time than instances with one activity. For example, 20_1_7_v1_G4 and 20_3_7_v3_G4 takes respectively 385.02 and 368.56 seconds. Even though the first one considers only one activity, the total number of employees required goes up 20 in some time periods, while the second one never exceeds 9 employees.

Real instances. Computational results on Real instances are shown in Table 6.14. The first and the second columns report the name of the instance and the lower bound. When a lower bound is not known (instances with 30 minutes and 15 minutes time unit), the third column show the best upper bound known, which is used to evaluate the optimality gap. By abuse of language, we will use the term "optimality gap" even if the best upper bound may not be the optimal value. The last three columns report the results of the hybrid heuristic, that is the upper bound *ub*, the computational time $t(s)$ and the optimality gap *gap*%.

Instance	lb	best ub	Hybrid		
			ub	t(s)	gap%
57_11_60_le	37270	-	39100	104.06	4.9
75_14_60_ll	135670	-	136140	917.53	0.3
43_15_60_lb	33015	-	34250	41.62	3.7
38_05_60_nh	44585	-	45535	121.78	2.1
23_05_60_nn	21095	-	21450	65.64	1.7
Average (60min)				**250.12**	**2.6**
57_11_30_le	-	74315	76275	533.78	2.6
75_14_30_ll	-	269530	270160	1568.11	0.2
43_15_30_lb	-	64540	65660	131.92	1.7
38_05_30_nh	-	82055	82480	364.28	0.5
23_05_30_nn	-	45090	45565	212.82	1.1
Average (30min)				**562.18**	**1.2**
57_11_15_le	-	149040	151515	807.74	1.7
75_14_15_ll	-	540115	540480	1849.12	0.1
43_15_15_lb	-	130210	130750	275.65	0.4
38_05_15_nh	-	159305	160000	840.83	0.4
23_05_15_nn	-	93060	93500	318.44	0.5
Average (15min)				**818.35**	**0.6**

Table 6.14: Results hybrid heuristic on Real instances.

The results show that the hybrid heuristic finds a solution with an optimality gap always lower than 5%, and in almost half of the instances, the gap does not exceed 0.5%. In these cases, the high degree of flexibility of these instances helps in finding a good solution. The differences in the computational time are mainly due to the TS heuristic. Indeed, as previously remarked, its computational time is not only affected by the number of employees (instance 75_14_*_ll with 75 employees has the highest time), but also by the demand profile. For example, even though instances 43_15_60_lb and 23_05_60_nn have respectively 43 and

23 employees, the computational time of the first one is lower than the computational time of the second one.

6.3.3 Conclusions

In this section we proposed a hybrid heuristic that makes use of a greedy heuristic to find an initial solution that satisfies exactly workload constraints, TS to integrate a particular class of constraints (*duration constraints*), and LNS to bring the solution to a complete feasibility by integrating all still violated constraints, while minimizing the total cost of the planning. Computational results are performed on three sets of instances with different degree of flexibility and different characteristics. The results show that the optimality gap is on average 2.1% and it does not exceed 5% in 88% of the instances. However, the results show two main weaknesses of the hybrid method that concern mainly TS. The first one is the fact that the performance of TS is strongly affected by the workload constraints. The second one is the fact that TS does not take into account predefined starting slots.

6.4 Diving heuristic

This section presents a diving heuristic used to quickly obtain a feasible integer solution of the multi-activity tour scheduling problem. This method consists in a depth first heuristic that partially explores the search tree in a B&P framework. The branching rule employed in diving heuristics is usually quite different from the ones in the exact B&P method. Indeed, these heuristics aim at finding quickly a good feasible integer solution, and not at having a balanced search tree. After branching, the reduced master problem is modified to deal with the partial solution fixed, and then it is solved again. We refer the reader to the work of Sadykov et al. (2018) for a deeper insight on diving methods. Applications in the context of multi-activity tour scheduling can be found in the work of Gérard et al. (2016).

The diving heuristic is described in Section 6.4.1, while computational results are shown in Section 6.4.2. Finally, conclusions are presented in Section 6.4.3.

6.4.1 The method

The diving heuristic developed for our problem solves, at each node of the search tree, the reduced master problem by means of the CG algorithm presented in Section 5.1. In order to obtain a fast heuristic, a time limit is introduced for CG at each node of the tree. As a consequence, CG may stop before achieving optimality. We remark that when this happens at the root node, a valid lower bound of the master problem (2.69)-(2.73) is not available. A classical branching strategy used in diving heuristic corresponds at rounding up or down a variable of the linear relaxation of the current reduced master problem (Sadykov et al. (2018)). At each node of the search tree, the variable x_p^i with the highest value is fixed to 1, i.e., a complete schedule is selected for the employee $i \in I$ such that column $p \in P^i$, which is set in the partial solution. After branching, the reduced master problem is modified as follows:

- all columns associated to employee i are deleted;
- the demand b_{ja} is updated for each slot $j \in J$ and for each activity $a \in A$, according to the fixed schedule c, i.e. $b_{ja} - \delta^i_{pja}$ where $\delta^i_{pja} \in \{0, 1\}$ is equal to 1 if c covers activity a in slot j, 0 otherwise;
- the pricing problem corresponding to employee i is no more called in the next nodes, in order to avoid generating other columns for this employee.

The resulting (residual) reduced master problem is then solved again. We remark that by fixing the partial solution as previously described, CG is not affected and no modification to the pricing problems needs to be done. Therefore, the residual master problem can be tackled in the same way as the reduced master problem at the root node. The exploration of the search tree is performed without backtracking. As a consequence, the depth of the search tree coincides with the number of employees $|I|$, and exactly $|I|$ nodes are solved in the diving heuristic.

6.4.2 Computational results

This section shows the computational results of the diving heuristic described. The algorithm has been coded in C# and tests have been performed on a 64-bit Windows operating system with 977 GB of RAM and 16 processors (only one core is used) Intel Core running at 2.00 GHz. CPLEX 12.7 has been used as LP solver. Experiments have been done on the three sets of instances RGR, RGR flexible and Real presented in Section 1.4. The time limit for the diving heuristic has been fixed to 1800 seconds, and, at each node of the search tree, a time limit of $1800/|I|$ seconds has been imposed to CG.

RGR instances. Computational results on RGR instances are shown in Table 6.15. The first and the second columns report respectively the name of the instance and the corresponding lower bound known. The next five columns show the results of the heuristic B&P method proposed by Restrepo et al. (2016). More in detail, *ub*, $t(s)$ and *gap*% are respectively the upper bound, the computational time and the optimality gap, while $t_{\mathrm{r}}(s)$ and gap_{r}% are the computational time and the gap evaluated at the root node. The last three columns show the results of the diving heuristic, that is the upper bound *ub*, the computational time $t(s)$ and the optimality gap *gap*%.

The results show that the mono-activity instances of both group G1 and group G3 are solved with an optimality gap that does not exceed 0.7%, and in 7 over 12 cases the integer solution found has a value that equals the lower bound, resulting in a null gap. However, even though the time limit has not been reached, the computational time is higher when compared to the B&P-RGR which solves all these instances at the root node. This is due to the fact that no heuristic is employed to determine an upper bound in any node of the search tree, and an integer solution is in general available only at the end of the diving

Instance	lb	B&P-RGR					Diving		
		ub	t(s)	gap%	t_r(s)	gap_r%	ub	t(s)	gap%
20_1_7_v1_G1	52080	52080	21.31	0.0	21.31	0.0	52080	165.79	0.0
20_1_7_v2_G1	49440	49660	31.00	0.4	31.00	0.4	49440	420.28	0.0
25_1_7_v1_G1	60560	60560	16.69	0.0	16.69	0.0	60560	215.20	0.0
25_1_7_v2_G1	72660	72660	6.32	0.0	6.32	0.0	72660	515.61	0.0
40_1_7_v1_G1	100410	100850	10.77	0.4	10.77	0.4	100410	491.52	0.0
40_1_7_v2_G1	98390	98940	90.52	0.6	90.52	0.6	98500	784.42	0.1
20_3_7_v1_G2	52900	-	-	-	-	-	54510	1794.70	3.0
20_3_7_v2_G2	60120	60120	1735.07	0.0	733.69	3.6	60745	1741.68	1.0
20_3_7_v3_G2	60450	60780	1507.83	0.5	697.80	2.6	60925	1468.73	0.8
20_1_7_v1_G3	82825	82860	3.66	0.0	3.66	0.0	82990	122.38	0.2
20_1_7_v2_G3	64675	65280	5.91	0.9	5.91	0.9	65060	172.12	0.6
25_1_7_v1_G3	99995	100380	3.47	0.4	3.47	0.4	100710	177.93	0.7
25_1_7_v2_G3	72660	73100	1.61	0.6	1.61	0.6	72660	37.29	0.0
40_1_7_v1_G3	202050	202250	1.38	0.1	1.38	0.1	202050	136.56	0.0
40_1_7_v2_G3	117677	118170	10.46	0.4	10.46	0.4	117840	246.99	0.1
20_3_7_v1_G3	71800	-	-	-	-	-	72445	1845.16	0.9
20_3_7_v2_G3	61656	62125	1408.06	0.8	730.60	5.8	62880	1774.47	2.0
20_3_7_v3_G3	63575	63950	1370.42	0.6	686.61	5.6	64275	1798.06	1.1
20_5_7_v1_G3	84554	-	-	-	-	-	89200	1766.88	5.5
20_5_7_v2_G3	86915	-	-	-	-	-	89280	1801.43	2.7
Average								**873.86**	**0.9**

TABLE 6.15: Results comparison between B&P heuristic implemented by Restrepo et al. (2016) and the diving heuristic time limit 1800 seconds on RGR instances.

heuristic. When dealing the multi-activity instances, the optimality gap increases, but it never exceeds 5.5%, and in most of the cases it is lower than or equal to 3.0%.

RGR flexible instances. Computational results on RGR flexible instances are shown in Table 6.16. The first and the second columns report the name of the instance and the lower bound known, while the last three columns report the upper bound *ub*, the computational time $t(s)$ and the optimality gap *gap*% obtained by the diving heuristic.

Instance	lb	Diving		
		ub	t(s)	gap%
20_1_7_v1_G4	47660	47980	1337.87	0.7
20_1_7_v2_G4	47440	47815	1268.80	0.8
20_3_7_v1_G4	50990	56010	1871.93	9.8
20_3_7_v2_G4	57245	60335	1776.13	5.4
20_3_7_v3_G4	57565	60840	1749.00	5.7
Average			**1600.75**	**4.5**

TABLE 6.16: Results diving heuristic time limit 1800 seconds on RGR flexible instances.

Due to the limited time imposed to CG at each node, we wanted to avoid spending too much time in solving the pricing problems by means of daily shifts selection strategies that require a considerable computational effort, such as (D5) and (D6). For this reason, we uses only the most restricting strategy (D1).

The results show that for the mono-activity instances the optimality gap achieved is lower than 1% in both cases. For these instances, the diving heuristic terminates before the time limit of 1800 seconds imposed. The reason relies on the fact that in the last nodes explored, when most of the employees have been fixed, the CG terminates before the time limit since it does not find negative reduced cost columns with the strategies used to heuristically solve the pricing problem. Concerning the multi-activity instances, we can see that the optimality gap deteriorates, exceeding 5.4% in the three cases and reaching a maximum gap of 9.8%.

Real instances. Computational results on Real instances are shown in Table 6.17. The first and the second columns report the name of the instance and the lower bound. When a lower bound is not known (instances with 30 minutes and 15 minutes time unit), the third column show the best upper bound known, which is used to evaluate the optimality gap, even though the best upper bound may not be the optimal value. The last three columns report the results of the diving heuristic, that is the upper bound *ub*, the computational time $t(s)$ and the optimality gap *gap*%.

Instance	lb	best ub	Diving		
			ub	t(s)	gap%
57_11_60_le	37270	-	37550	1740.67	0.8
75_14_60_ll	135670	-	135670	1846.82	0.0
43_15_60_lb	33015	-	33125	1578.75	0.3
38_05_60_nh	44585	-	44585	1466.21	0.0
23_05_60_nn	21095	-	21095	1092.21	0.0
Average (60min)				**1544.93**	**0.2**
57_11_30_le	-	74315	77465	1784.27	4.2
75_14_30_ll	-	269530	269530	1866.70	0.0
43_15_30_lb	-	64540	67650	1646.55	4.8
38_05_30_nh	-	82055	83410	1774.33	1.7
23_05_30_nn	-	45090	45505	1539.00	0.9
Average (30min)				**1722.17**	**2.3**
57_11_15_le	-	149040	160470	1720.82	7.7
75_14_15_ll	-	540115	540115	1843.17	0.0
43_15_15_lb	-	130210	141455	1874.91	8.6
38_05_15_nh	-	159305	161205	1838.32	1.2
23_05_15_nn	-	93060	94050	1738.90	1.1
Average (15min)				**1803.22**	**3.7**

TABLE 6.17: Results diving heuristic with time limit 1800 seconds on Real instances.

As previously done for RGR flexible instances, the pricing problems are solved with the daily shift selection strategy (D1). The reason lies not only in the computational effort needed when using these strategies, but also in the high number of pricing problems. We recall that for these instances, each employee differs from the others, requiring a pricing problem for each one. In addition, when solving the instances with 15 minutes time unit, the pricing solution space is further restricted by using starting slot selection strategy (S2) instead of (S3). Finally, only configurations (C1)-(C3) are considered.

The results show that for all instances with 60 minutes time unit, the diving heuristic finds an integer solution with an optimality gap lower than or equal to 0.8%, and in 3 over 5 cases the gap is 0.0%. When the time unit decreases, we can remark a deterioration of the optimality gaps, that reaches 4.8% for the 30 minutes and 8.6% for the 15 minutes. However, in 2 instances (75_14_15_ll and 75_14_15_ll) the gap is equal to 0.0%, meaning that the diving heuristic has found an integer solution with the best value known.

6.4.3 Conclusions

In this section we presented a diving heuristic which consists of a depth first heuristic search in a B&P tree obtained by branching on the master problem variables x_p^i. At each node, a complete schedule is selected for one employee and it is set in the partial solution. Due to the high computational time needed to solve exactly the root node in most of the instances, we set a time limit for CG at each node of the search tree. The results show that 75% of the instances are solved with an optimality gap that does not exceeding 2.0%, while 63% of the instances has a gap not exceeding 1.0%.

6.5 Heuristics comparison

This section has the purpose of comparing the performances of the heuristics methods presented in this chapter. Table 6.18, Table 6.19 and Table 6.20 present the computational time $t(s)$ and the optimality gap *gap*% on RGR, RGR flexible and Real instances respectively. The results are shown for the heuristics *LNS* (cf. Section 6.1), the *PD* method and its variant *PD-tl* with time limit (cf. Section 6.2), the *Hybrid* heuristic (cf. Section 6.3), and the *Diving* heuristic (cf. Section 6.4).

From Table 6.18 we can see that the best average optimality gap is achieved with the diving heuristic, while the fastest is PD-tl. If we analyze instance by instance, the diving obtains the best gap in 13 over 20 cases, where in half of them (7 instances) the optimal solution is found with a null gap. Concerning the other heuristics, LNS, PD, PD-tl and hybrid find a solution with the best optimality gap respectively in 3, 8, 10 and 4 cases over 20. Therefore, PD-tl is competitive with the diving heuristic in terms of solution quality, while it is faster than diving in all the instances. We remark that the computational time of PD and PD-tl is the same in the majority of the instances. The reason relies on the fact that PD-tl imposes a time limit of 180 seconds on CG, which is not exceeded in most of the instances. As consequence, PD and PD-tl coincide in these cases.

Instance	LNS		PD		PD-tl		Hybrid		Diving	
	t(s)	gap%	t(s)	gap%	t(s)	gap%	t(s)	gap%	t(s)	gap%
20_1_7_v1_G1	453.49	0.2	**12.42**	0.6	**12.42**	0.6	507.42	0.2	165.79	**0.0**
20_1_7_v2_G1	849.93	**0.0**	**18.70**	0.4	**18.70**	0.4	476.94	**0.0**	420.28	**0.0**
25_1_7_v1_G1	570.83	**0.0**	**20.76**	0.9	**20.76**	0.9	424.31	**0.0**	215.20	**0.0**
25_1_7_v2_G1	1067.76	0.3	**14.73**	**0.0**	**14.73**	**0.0**	561.10	**0.0**	515.61	**0.0**
40_1_7_v1_G1	1073.50	**0.0**	**18.44**	0.4	**18.44**	0.4	594.81	**0.0**	491.52	**0.0**
40_1_7_v2_G1	1370.10	0.3	**77.00**	0.8	**77.00**	0.8	1380.68	0.4	784.42	**0.1**
20_3_7_v1_G2	**139.56**	2.5	323.16	**0.9**	1058.74	**0.9**	600.76	2.3	1794.70	3.0
20_3_7_v2_G2	192.04	1.5	**142.93**	**0.7**	**142.93**	**0.7**	182.65	1.1	1741.68	1.0
20_3_7_v3_G2	223.85	0.9	**187.09**	**0.8**	**187.09**	**0.8**	222.30	2.6	1468.73	**0.8**
20_1_7_v1_G3	**11.95**	3.0	17.39	0.8	17.39	0.8	345.39	2.2	122.38	**0.2**
20_1_7_v2_G3	**31.46**	2.0	57.42	0.8	57.42	0.8	106.32	5.7	172.12	**0.6**
25_1_7_v1_G3	**22.15**	2.2	22.64	**0.7**	22.64	**0.7**	129.25	3.3	177.93	**0.7**
25_1_7_v2_G3	20.77	1.9	**6.45**	**0.0**	**6.45**	**0.0**	99.34	0.3	37.29	**0.0**
40_1_7_v1_G3	27.57	0.4	**12.61**	0.2	**12.61**	0.2	365.31	0.4	136.56	**0.0**
40_1_7_v2_G3	**22.76**	4.6	74.49	0.8	74.49	0.8	269.01	5.2	246.99	**0.1**
20_3_7_v1_G3	**28.57**	1.1	329.45	0.8	350.49	**0.5**	827.19	1.3	1845.16	0.9
20_3_7_v2_G3	**33.24**	3.0	1220.14	2.2	1139.44	**1.5**	72.25	4.1	1774.47	2.0
20_3_7_v3_G3	**20.61**	2.7	576.74	**0.9**	576.74	**0.9**	69.20	2.7	1798.06	1.1
20_5_7_v1_G3	**40.91**	9.0	2982.18	**5.0**	1160.50	6.5	193.04	12.9	1766.88	5.5
20_5_7_v2_G3	**29.73**	6.0	1657.43	2.8	1127.24	**2.4**	245.65	6.4	1801.43	2.7
Average	**311.54**	**2.1**	**388.61**	**1.0**	**304.81**	**1.0**	**383.65**	**2.6**	**873.86**	**0.9**

TABLE 6.18: Comparison heuristics on RGR instances.

A different situation arises from the experiments on the RGR flexible instances, whose results are presented in Table 6.19. Indeed, LNS is the fastest heuristic in all the instances and the computational time is much lower compared to the others. The best average gap is achieved by the PD method, with a value of 0.9% in all the instances. Furthermore, PD finds the best solution in 3 over 5 cases. However, the time goes up to 4 hours when 3 activities are considered, due to the high computational effort required to find a valid lower bound in the first iteration of PD. PD-tl and the hybrid heuristic find the optimal solution for one of the instances with a null gap, while the diving heuristic is the best in terms of optimality gap in 1 over 5 cases.

Instance	LNS		PD		PD-tl		Hybrid		Diving	
	t(s)	gap%	t(s)	gap%	t(s)	gap%	t(s)	gap%	t(s)	gap%
20_1_7_v1_G4	**217.07**	4.1	2190.25	0.9	1138.94	0.8	385.02	5.4	1337.87	**0.7**
20_1_7_v2_G4	**107.77**	3.6	753.32	0.9	1141.60	**0.0**	345.58	**0.0**	1268.80	0.8
20_3_7_v1_G4	**215.44**	7.3	4037.52	**0.9**	1048.02	4.1	543.39	3.7	1871.93	9.8
20_3_7_v2_G4	**158.03**	1.9	13801.92	**0.9**	954.18	1.1	385.69	1.5	1776.13	5.4
20_3_7_v3_G4	**170.89**	1.6	15326.60	**0.9**	1051.30	1.1	368.56	1.4	1749.00	5.7
Average	**173.84**	**3.7**	**7221.92**	**0.9**	**1066.81**	**1.4**	**405.65**	**2.4**	**1600.75**	**4.5**

TABLE 6.19: Comparison heuristics on RGR flexible instances.

The results on the Real instances are shown in Table 6.20. We recall that a valid lower bound is not available for the instances with 30 and 15 minutes time unit, due to the extremely

Instance	LNS		PD		PD-tl		Hybrid		Diving	
	t(s)	gap%	t(s)	gap%	t(s)	gap%	t(s)	gap%	t(s)	gap%
57_11_60_le	**31.45**	5.0	1302.92	0.9	1075.97	**0.4**	104.06	4.9	1740.67	0.8
75_14_60_ll	**37.20**	1.2	1559.51	0.7	1029.93	**0.0**	917.53	0.3	1846.82	**0.0**
43_15_60_lb	**15.10**	3.8	536.43	**0.2**	1073.77	0.3	41.62	3.7	1578.75	0.3
38_05_60_nh	**17.21**	1.3	713.74	0.9	986.35	**0.0**	121.78	2.1	1466.21	**0.0**
23_05_60_nn	**6.83**	3.2	713.33	**0.0**	1061.85	**0.0**	65.64	1.7	1092.21	**0.0**
Average	**21.56**	**2.9**	**965.19**	**0.5**	**1045.57**	**0.1**	**250.13**	**2.5**	**1544.93**	**0.2**
57_11_30_le	**316.76**	2.2	-	-	988.3	**0.0**	533.78	2.6	1784.27	4.2
75_14_30_ll	**260.53**	1.4	-	-	972.57	0.4	1568.11	0.2	1866.70	**0.0**
43_15_30_lb	**125.29**	2.0	-	-	1046.34	**0.0**	131.92	1.7	1646.55	4.8
38_05_30_nh	**119.55**	1.5	-	-	1048.74	**0.0**	364.28	0.5	1774.33	1.7
23_05_30_nn	**56.69**	1.7	-	-	1028.67	**0.0**	212.82	1.1	1539.00	0.9
Average	**175.76**	**1.8**			**1016.924**	**0.1**	**562.18**	**1.2**	**1722.17**	**2.3**
57_11_15_le	**526.39**	1.8	-	-	1095.21	**1.7**	807.74	**1.7**	1720.82	7.7
75_14_15_ll	**583.51**	0.8	-	-	1165.11	0.1	1849.12	0.1	1843.17	**0.0**
43_15_15_lb	**154.48**	0.9	-	-	1028.58	**0.0**	275.65	0.4	1874.91	8.6
38_05_15_nh	**425.75**	1.5	-	-	1112.16	**0.0**	840.83	0.4	1838.32	1.2
23_05_15_nn	**122.05**	0.9	-	-	959.80	**0.0**	318.44	0.5	1738.90	1.1
Average	**362.44**	**1.2**			**1072.172**	**0.4**	**818.36**	**0.6**	**1803.22**	**3.7**

Table 6.20: Comparison heuristics on Real instances.

high degree of flexibility which leads to the difficulty for CG in solving the linear relaxation of the master problem (2.69)-(2.73). As consequence, the gap is evaluated considering the best upper bound found by all the heuristics tested, and we call it "optimality gap" by abuse of language. Another consequence consists in the fact that PD is not able to solve instances with 30 and 15 minutes time unit, since the method requires CG to converge in order to find a valid lower bound. The results on these instances are marked with "-". LNS is the fastest heuristic for all the instances, and the computational time is lower of at least one order of magnitude in some cases when compared to the other heuristics. Concerning the quality of the solution, PD-tl achieves the best optimality gap in 12 over 15 cases, and, except in one instance where the gap is 1.7%, the gap never exceeds 0.4%. In addition it finds the optimal solution in 3 over 5 cases for the 60 minutes instances, while it finds the best solution known in 7 over 10 cases for the 30 and 15 minutes instances.

In conclusion, the LNS heuristic results the fastest in 75% of the instances, mostly in the large-scale ones. It achieves an optimality gap that is in average 2.2% and exceeds 5.0% only in 3 cases. The heuristic that gives the best compromise between computational time and optimality gap is PD-tl. Indeed, the first is on average 677.59 seconds, while the second on average 0.8% and it exceeds 5.0% only in one case.

Conclusions

This thesis originates from a partnership between the company Horizontal Software and LIPN of Université Paris 13, and it addresses the multi-activity tour scheduling problem through different exact and heuristic methods. In the following we sum up the main contributions and we outline some future improvements and research directions.

Main contributions

The problem addressed in this thesis takes inspiration from the personnel scheduling problem arising in the context of the restaurant business, one of the challenging areas in which Horizontal Software offers its E-Optim tool. The problem is classified as multi-activity tour scheduling problem and its main characteristic is the high degree of schedule flexibility, coming from the work regulations and from the possibility of assigning long pauses (interruptions), which may span up to 5 hours.

Part I of this thesis presents two modeling approaches. The first is a compact MILP model which extends the one proposed by Gérard et al. (2016) to deal with the particular features of the addressed problem. The second is based on Dantzig-Wolfe decomposition, which is well-suited for multi-activity tour scheduling and it allows to decompose the problem in smaller ones. As done in Gérard et al. (2016) and Restrepo et al. (2016), the master problem is based on a generalized set partitioning problem with convexity constraints, where binary variables are associated to schedules, and positive continuous variables are associated to under and over demand coverage.

Part II is devoted to the exact resolution of the problem through a B&P method, whose key element is CG. As a first step towards the independence from commercial solvers, Chapter 3 is dedicated to the resolution of the reduced master problem and it proposes a DA heuristic for obtaining feasible dual solutions of the generalized set partitioning problem with convexity constraints. The generality of this model allows the use of DA for solving problems that arise from different contexts. Indeed, computational experiments are done not only on multi-activity tour scheduling instances, but also on minimum sum coloring instances and randomly generated instances. The results reveals the quality of the dual solutions evaluated by the DA, which allows to speed up the convergence of CG and soften the oscillations of the dual variables.

Chapter 4 is dedicated to the resolution of the pricing problem(s) and it proposes an exact three phases method based on both constraint and dynamic programming. In the first phase, formal language (regular grammar) models the rules defining feasible timeslots. In the second phase, timeslots, breaks and interruptions are assembled into feasible daily shifts. Regular grammar is used to model part of the rules defining daily shifts, while a label algorithm is used to determine, in the expanded graph, the resource constrained shortest path that captures the remaining daily shifts rules. In the third phase, daily shifts and days-off are combined into schedules by using a bi-directional label algorithm. The layered structure of the proposed methods makes it suitable for the use of different strategies to solve heuristically the pricing problem, essentially when dealing with the large-scale instances. Computational experiments compare the different strategies proposed and highlight the higher degree of flexibility of the real instances provided by Horizontal Software when compared with instances from the literature.

The DA and the pricing resolution method are combined into a CG framework, along with different accelerating strategies such as intensification, diversification, columns elimination, heuristic pricing and partial pricing. This framework is at the basis of the B&P of Chapter 5. Computational experiments show that optimality is proven only for small-scale instances. In addition, it fails in finding a feasible solution for the large-scale instances of Horizontal Software, due to their degree of flexibility which results in high number feasible schedules.

In order to deal with the large-scale real instances and efficiently find good quality solutions, Chapter 6 proposes four different heuristics: a LNS that iteratively destroys and repairs an initial solution using the pricing solving method proposed in Chapter 4; a PD heuristic that makes use of CG to get a lower bound, and LNS to get an upper bound and a feasible integer solution; a hybrid heuristic that combines a greedy heuristic to determine an initial solution satisfying workload, a TS to integrate a particular class of constraints (duration constraints), and LNS to obtain a feasible solution; finally, a diving heuristic that partially explores a B&P search tree. Computational experiments reveals that LNS is well-suited for obtaining quickly feasible integer solutions, while PD determines high quality solutions. However, PD is subject to the convergence of CG which suffers on the large-scale real instances. The variant of PD with time limit is a good alternative for finding high quality integer solutions.

In addition to the work of this thesis, a contribution on a different research area has been done. It deals with a problem related to the energy management in a decentralized setting, where the interactions between a generation company and various micro-grids are taken into account. Therefore, it has been an interesting opportunity to cope with combinatorial and bilevel optimization applications in a rather different domain than the main one of the thesis. Appendix A addresses a variant of the power generation problem where a generation company interacts with the micro-grids by buying and selling power to them. The problem is modeled as a bilevel stochastic optimization problem, and a heuristic one-level reformulation is proposed to deal with realistic size instances.

Research directions

The work on the multi-activity tour scheduling problem addressed has left several tasks to be accomplished shortly after the end of this thesis, and some interesting research paths that will be investigated on a longer-term horizon. They concern both the academic and the industrial point of view.

The limitations of the pricing solving method based on constraint and dynamic programming have been highlighted on instances with a high degree of flexibility (RGR flexible and Real). The main reason relies on the number of feasible daily shifts, which affects the size of the DAG used for generating schedules in the third phase. As a consequence, dynamic programming algorithms become slow or they are not even able to solve the pricing problem. Real-world instances with a high degree of flexibility have been solved with time unit of 60 minutes. To solve instances with a smaller time unit, a further step needs to be done to improve the third phase. In parallel, it could be interesting to understand how much the flexibility characterizing the real instances is useful, taking advantage of the managers know-how and/or analyzing the historical planning approved.

In order to deal with even more real instances it would be interesting to take into account the distribution constraints that appear, for instance, when employees need to attend a formation course. At this stage, it is not easy to consider these constraints when generating the schedules due to the fact that the proposed pricing solving method “forgets” the activity performed within each timeslot when building daily shifts. A first idea that comes in mind consists in generating timeslots (daily shifts) where the employee works and timeslots (daily shifts) where the employee takes the course. For instance, if the course can be attended every day from 9am to 11am, all timeslots and daily shifts covering these time periods are generated first forbidding and then imposing that the course is attended. In addition, a further resource is necessary in the label algorithm of the third phase. However, the consequence of these modifications lies in the increasing size of the DAG used for generating schedules. A second class of constraints arising in various personnel scheduling problems consists in the fair assignment of hard activities that may require an intense physical or mental effort (equity constraints). These constraints are essential for the satisfaction and the well-being of the employees. For instance, in the context of the fast food restaurant chain where the open hours are longer than usual restaurants, a fair assignment of the opening and closing activities is required.

Another aspect of the real instances that could be interesting to study is stochasticity, which may be due to endogenous and exogenous factors to the company. The first concern demand uncertainty while the second comes from the employees. One of the assumptions of the thesis is that all availabilities of the employees are known before the scheduling process. In addition, we assume that all employees perform the assigned schedule. However, it may happen that some of them do not show up at work without informing the manager. In these case, re-optimization methods could be employed with the goal of generating a new solution which is as close as possible to the old one.

Appendix A

Optimizing power generation in the presence of micro-grids

In this appendix we present a work that has been done during the 117th European Study Group with Industry, in collaboration with Wim van Ackooij (EDF R&D), Jérôme De Boeck (Graphes et optimisation mathématique, Université Libre de Bruxelles), Michael Poss (UMR CNRS 5506 LIRMM, Université de Montpellier) and Boris Detienne (Institut de Mathématiques de Bordeaux, Université de Bordeaux). The problem concerns energy management in a decentralized setting. In today's system, energy distribution is made possible by the power distribution grid, a system of transmission that allow electricity to be transferred from the point of generation to clients houses. Therefore, all problems are all looked at from the eye of a centralized planner, a large GenCo. Only to mention one of these problems, the unit-commitment aims at finding the most cost-effective production schedule while satisfying the operational constraints of the units. Network of the future generation will replace traditional and centralize power distribution grid with smart-grids. They will introduce new actors (micro-grids) and new type of iterations between them, which need to be defined and investigated. In this work we focus on the relation between the main grid and the micro-grids which will be defined through means of contracts.

This paper was accepted for publication in *European Journal of Operational Research* in May 2018.

Optimizing power generation in the presence of micro-grids

Wim van Ackooij. EDF R&D. OSIRIS, 7 Boulevard Gaspard Monge, F-91120 Palaiseau Cedex France

Jérôme De Boeck. Graphes et optimisation mathématique, Université Libre de Bruxelles, B-1050 Brussels, Belgium and INOCS, INRIA Lille Nord-Europe, France

Boris Detienne. Institut de Mathématiques de Bordeaux, Université de Bordeaux, Inria Bordeaux-Sud-Ouest, 153 cours de la libération, 33400 Talence, France

Stefania Pan. UMR 7030 CNRS LIPN, Université Paris 13, 99 avenue Jean-Baptiste Clément, 93430 Villetaneuse, France

Michael Poss. UMR CNRS 5506 LIRMM, Université de Montpellier, 161 rue Ada, 34392 Montpellier Cedex 5, France

Abstract. In this paper we consider energy management optimization problems in a future wherein an interaction with micro-grids has to be accounted for. We will model this interaction through a set of contracts between the generation companies owning centralized assets and the micro-grids. We will formulate a general stylized model that can, in principle, account for a variety of management questions such as unit-commitment. The resulting model, a bilevel stochastic mixed integer program will be numerically tackled through a novel preprocessing procedure. As a result the solution for the bilevel (or single leader multiple follower) problem will be neither "optimistic" nor "pessimistic". We will numerically evaluate the difference of the resulting solution with the "optimistic" solution. We will also demonstrate the efficiency and potential of our methodology on a set of numerical instances.

Keyword. Integer programming, power generation, bilevel optimization, stochastic optimization, single leader multiple follower games.

A.1 Introduction

In the upcoming future the energy landscape will significantly change from the current picture by incorporating more and more decentralized elements. Of particular importance is the advent of so-called micro-grids (MG). These are subparts of the system with an advanced energy management system interacting with programmable elements in the grid including good monitoring and control functions, a pervasive communication system and specific items such as smart meters, programmable loads, switchable storage systems and a variety of controllable energy sources including solar, wind and wave generators. Some of the major changes introduced by smart grids are the following:

- Micro-grids: smaller nearly isolated sub-grids that interact only with the global system when a load/offer mismatch occurs. Most importantly these sub-grids can be managed to follow a local economical target (which may be different and contrary to a system-wide interest).
- Partial storage, perhaps through electrical vehicles or powerful batteries. These storage devices can partially mitigate the intermittency of local decentralized production such as wind / solar generation.
- Demand management tools: use advanced information technology to pilot electricity use. For instance, shut down electrical heating, reprogram hot water tank recharging etc...

In this new setting it becomes of great interest to examine the interaction with more traditional elements composing the power system. For instance what will be the new role for large centralized generation assets such as nuclear, thermal or hydro generation? It also becomes of interest to examine how classical energy management questions, such as unit-commitment (e.g., Tahanan et al. (2015)), scheduling maintenance of large power plants or cascaded reservoir management (e.g., de Matos et al. (2017); Taktak and D'Ambrosio (2017)) should evolve to account for this new context. Needless to say, these classical management questions are already difficult on their own without considering a potential interaction with micro-grids. On top of this comes also the need to consider and account for uncertainty which are of great importance for obtaining meaningful management solutions.

The interaction between such micro-grids and centralized assets may be mathematically cast into the setting of a "game". We will make the specific choice of considering hierarchical games wherein a (single) large centralized operator (the leader or upper level) interacts with one or several micro-grids (the follower(s) or lower level) in a specific way. Such problems also go by the name of single leader multi-follower or in the case of a single follower by the naming of Stackelberg game or principal agent problems. We follow the terminology of Dempe et al. (2015) speaking of bilevel optimization problems, i.e., an optimization problem containing constraints that certain variables belong to the optimal set of other parametric optimization problems. Generally speaking these problems fall into the class of mathematical problems with equilibrium constraints, where Facchinei and Pang (2003a,b) are key references. The situation with multiple leaders is significantly harder to analyse and falls into the class of Equilibrium problems with Equilibrium constraints (EPECs). We refer to, e.g., Aussel et al. (2017a,b); Henrion et al. (2012); Heymann and Jofré (2016) for some applications in energy and to e.g., Pang and Fukushima (2005, 2009) and references therein for structural studies. Under some appropriate structure, a bilevel optimization problem involving only continuous variables can be cast as a DC (difference of convex) - problem. We refer to Pang et al. (2016); van Ackooij and de Oliveira (2017) for more on such methodology.

In Hobbs and Nelson (1992) the authors claim to investigate bilevel programming in the electric utility industry for the first time. The lower level (or follower), representing customers deals with maximizing benefit while investing in energy conservation. From a technical viewpoint, the authors substitute the Karush-Kuhn-Tucker (KKT) conditions for the lower level problem and process the subsequent mathematical program with equilibrium constraints (MPEC), by employing an exhaustive search on the complementarity conditions (one classic problem is solved for each state of the complementarity conditions). This procedure means that for an n dimensional complementarity constraint, 2^n problems need to be solved. Substituting the lower level KKT conditions in order to obtain a one-level more classic problem is commonplace in the energy literature (e.g., Asimakopoulou et al. (2013); Cervilla et al. (2015); Kardakos et al. (2016) just to name a few). However it should be noted that, following Dempe and Dutta (2012), this does not necessarily lead to an equivalent formulation of the original problem. In other words, the solution resulting from the KKT reformulation need not be a solution to the original bilevel program. This phenomenon can occur even in the simplest linear setting. We also refer to the recent Adam et al. (2018) illustrating a similar phenomenon. The essential observation is that the original problem is augmented with a given set of Lagrange multipliers. Consequently any local solution to the original problem must be a local solution to the augmented problem for all such Lagrange multipliers. However, whenever the augmented problem is solved, one ends up with just one specific Lagrange multiplier. In principle it should thus be verified if the thus obtained solution remains a local solution for all other Lagrange multipliers. Yet these are in general not readily available. For a simple 2 variable example of a situation wherein the KKT reformulation provides local solutions that are not local solutions of the original problem, see (Dempe et al., 2015, Example 3.1). The authors of Haurie et al. (1992) also consider a leader-follower framework in the electrical industry but rather in the objective of benchmarking a given investment strategy in cogeneration against a centrally planned strategy. In order to do so, they develop an iterative algorithm, which numerically seems to converge to a Betrand-Cournot equilibrium. The comparison with the centrally planned solution reveals inefficiencies of the given investment strategy.

Recently, demand side management (or demand response) was recognized to be an important aspect of the upcoming electrical system. As such, the authors of Fernández-Blanco et al. (2016) claim to be the first to consider such demand response within a bilevel framework. The lower level (or follower) problem is an optimal power flow problem under a direct current assumption. The lower level power balance equation and its dual multiplier in turn help fixing the global price of energy in the upper level (or leader) problem (e.g., Hao and Zhuang (2003); Momoh and Mili (2009) suggest a similar model but without solving it). From a methodological viewpoint, bilinear terms are linearized using McCormick envelopes, the lower level KKT conditions are substituted for the lower level problem and the special model structure can be exploited through several substitutions. In view of those claims our work is one of the first to consider the effect of demand side management on classical energy management questions. Moreover we will highlight that demand side management is responsible for ensuring that the optimistic and pessimistic solutions of the bilevel program are not identical. Although in our work we will focus on the interaction between a utility and several end-users grouped within one or several smart-grids, we will not consider the interaction between users as done in Mohsenian-Rad et al. (2010) where the focus is on reducing peak-to-average load ratios while accounting for potential games between users.

As usual in these works, we too will adopt a market-like setting wherein a predetermined set of contracts is offered to the customers.

In this paper we thus contribute to this difficult question by considering a stylized interaction between a generation company (GenCo) owning a (large) set of centralized assets and a set of micro-grids. We also account for potential competition at the centralized level, but only in a simplified way. Indeed we will make the assumption that any competitors to the GenCo have a fixed predetermined interaction with the micro-grids. In this work the GenCo can offer contracts to the micro-grid that detail the price of buying/selling electricity to the network. From a mathematical viewpoint we suggest a bilevel stochastic mixed integer program (BMIP) which in principle can account for a variety of instantiations thus covering any of the above management questions. In our numerical experiments however, we will focus on unit-commitment.

The first general methods for obtaining "optimistic" optimal solutions BMIPs can be traced back to Moore and Bard (1990) and Bard and Moore (1992), which could solve only small instances, with up to 10 general integer variables and 35 binary variables for the first level problem. While more recent works have shown significant improvements (e.g., Fischetti et al. (2016) based on intersection cuts), they are unable to handle the complex MILPs involved in unit-commitments or more complex power flow optimization problems. Our technical contribution therefore lies in suggesting a tractable reformulation of the bilevel problem that leads to a solution that is neither "optimistic" nor "pessimistic". This "heuristic" reformulation is tractable thanks to a preprocessing step that is not harder than solving the original management problem. We also compare the solution obtained from this reformulation with the "optimistic" solution that can only be obtained in reasonable times for small systems. Extensive numerical experiments confirm the interest of the here suggested reformulation. Essentially our reformulation takes inspiration, and to some degree is equivalent with, the value function approach originally designed by Outrata (1990). An advantage of this approach over the usual substitution of the KKT conditions is that we do obtain a solution of the original problem, albeit, not necessarily the "optimistic" solution.

The outline of this work is as follows. The assumptions of our model are described in Section A.2, where we present the ingredients of our model in a deterministic setting. Section A.3 deals with the suggested reformulation technique for the bilevel problem. In Section A.4, we show how our models can be adapted to handle uncertainty. We also show how our suggested reformulation can be adapted to this new setting. We assess our models and methodology in Section A.5 on a case study built from realistic data. Some of the details of the model underlying the case study are provided in A.7. Finally, we conclude the paper in Section A.6.

A.2 The problem

A.2.1 Description

In this paper we propose a stylized model wherein the current electrical system has transmuted into a likely potential future. In this future, several nearly independent subsystems, – micro-grids –, exist and interact with classical generation companies. We will make the assumption that micro-grids typically dispose of a set of generation assets mostly comprised of renewable intermittent sources (wind, solar). Remaining energetically independent of the rest of the system is then possible, up to a certain extent, by also disposing of a set of batteries. The remaining time, i.e., in case of production surplus or lack of generation, an interaction with the classical generators will provide useful back up to meet the total demand in energy. In our model we will focus on this last type of interaction from an abstract level and suggest several convenient reformulations so as to make the model tractable. This is non-trivial, since the formulated model will be a bilevel stochastic mixed integer optimization problem. Needless to say, the price to pay for tractability is a heuristic means of solving the problem, but by no means an unrealistic one. Indeed, our suggestion solution will be "feasible"[1] for the original bilevel problem, but neither be the "optimistic" nor "pessimistic" solution. From the viewpoint of practice, this is actually not unrealistic since both extremes (the "optimistic" and "pessimistic" views) are stylized anyway. Indeed there is no reason to believe that an independent actor pursuing his own goals (e.g., in the case of micro-grids, this could be to remain independent energetically speaking most of the time), will purposely select the most advantageous response for the leader, nor any reason to believe it would select the most adverse option. Having something in between can thus be assumed reasonably realistic.

Before providing a more mathematical description of our model, let us begin by introducing the key elements of it.

Parties involved We consider an extension of electricity production problems that involve two types of parties:

- **GenCos** are big producers of electricity in the network (they own large assets such as nuclear, thermal, hydro and other renewable energies). In order to avoid having to face an even tougher class of problems (an equilibrium problem with equilibrium constraints, or a coupled set of MPECs, e.g., Surowiec (2010)), we focus on the decision of a single operator that faces competing companies that have a fixed, perfectly known, policy. Hence, in what follows, GenCo refers to that single operator.
- Each **micro-grid** $q \in Q$ consists of a small subnetwork that has highly volatile generation capacities (solar, wind), and two types of demands to be attended, which we denote as hard and elastic demands. On the one hand, hard demands must be met

[1]Feasible here means that the given solution satisfies both upper and lower level constraints. Moreover at the given fixed upper level variables, the lower level variables are optimal for the lower level problem.

strictly at all times. On the other hand, elastic demands (heating up water, recharging electric cars, ...) can be shifted within a certain time window. This models the fact that environmentally aware users may be willing to postpone / anticipate an essential electrical consumption in order to reduce overall environmental burden (here translated through the system cost). Since this use is essential it will take place somewhere else within a given time window. Micro-grids are assumed to be relatively autonomous in terms of energy. However, due to the uncertain nature of their production, they need to buy or sell electricity from the GenCo.

Contract Micro-grids and GenCos interact through contracts that specify the costs of buying/selling electricity from/to the GenCo for each period of the time horizon T. Specifically, each contract $k \in K$ is specified by (i) the price of contract c_k paid by the micro-grid to the GenCo that proposes it, (ii) a linear cost function $x \mapsto f_{kt}x$ for buying the amount x of electricity during time period t, and (iii) a linear function $y \mapsto g_{kt}y$ for selling the amount y of electricity during time period t. We denote by K_0 the subset of contracts possibly proposed by the GenCo whose decisions are being optimized, while K also contains contracts of competing companies.

Objective On the one hand, the objective of the GenCo is to propose the least cost production schedule based on (i) the classical costs of unit-commitment and related problems (hydro optimization and nuclear maintenance scheduling, for instance) and (ii) the cost/benefit of buying/selling electricity to the micro-grids. On the other hand, the objective of each micro-grid is to minimize its total cost of buying/selling electricity to the GenCos, by choosing a contract offered by the GenCo or one of the competing companies (in the latter case, the GenCo does not produce, buy or earn anything for/from the micro-grid). These two conflicting objectives can be naturally modeled as a bilevel optimization problem, introduced in the next section.

Coupling among the different time periods We are given a time horizon $\mathcal{T} = \{1, \dots, T\}$ that is further partitioned into days: $\mathcal{T} = D_1 \cup \cdots \cup D_J$. Elastic loads couple all time periods of a given day. However, they do not couple time periods of different days as we may reasonably assume that the required load will be met during one full day. Nevertheless, time periods of different days may be coupled together in the presence of batteries with significant storage capacity. In this setting, the resulting multi-stage stochastic optimization problem is significantly harder, preventing us from decomposing the problem in time (by day). This is an immediate result of the temporal coupling of adjacent days through the large storage capacity batteries and the "strongly increasing complexity" of multi-stage stochastic programs with the number of stages (e.g., Shapiro (2006)).

A.2.2 A bilevel formulation

We describe below our mathematical model, represented as a bilevel multi-stage stochastic program. Let $x \in \mathbb{R}^{Q\times T}$ represent the electricity production of the GenCo. We are mainly interested here in the interaction between the GenCo and the micro-grids, so the value x_{qt} represents the electricity produced by the GenCo and fed into micro-grid q during period t. Similarly, we can define y_{qt} as the amount of electricity bought by the GenCo from micro-grid q during time period t. We denote by $F : \mathbb{R}^{Q\times 2T} \to \mathbb{R} \cup \{\infty\}$ the cost of producing $x - y$. Hence, one can think of the problem

$$\min_{x,y\geq 0} F(x,y) \tag{A.1}$$

as a compact abstract representation for the combination of unit-commitment, nuclear power plant maintenance planning, hydro power generation and other related problems, feeding the connected micro-grids with the power described by $x - y$. The mapping F can take the value ∞ for given vectors x, y if certain constraints cannot be met. For instance, whenever $x - y$ exceeds the generation capacity of the system. With this convention any constraints on generation (or other constraints) can be readily incorporated in the framework by adding the characteristic function of these constraints to F. Let us note that computing $F(0,0)$ amounts to solving the power generation problem without considering micro-grids. In particular, computing F is therefore as difficult as computing the least cost power generation schedule. We also remark that in view of the above discussion, assuming that the GenCo problem is feasible without micro-grids means that $F(0,0) < \infty$. We note that from a computational viewpoint it is better to incorporate constraints directly in the model so as to avoid having to deal with very large numerical values. This is what is done in the numerical experiments. Handling constraints by allowing F to take on the value ∞ is just convenient for the mathematical presentation.

In our model the vectors x and y indicate the power flow between the micro-grids and the (GenCo). These variables are related to the internal functioning of the micro-grids and in particular their internal constraints. To this end we introduce an abstract constraint set $\mathcal{M}_q$ for each $q \in Q$, modelling in particular the demand-power balance. These power flows induce a certain cost governed by a contract. In our model, the GenCo has to decide which contracts it proposes to the micro-grids. We introduce an additional binary variable Z_{qk} that is equal to 1 if contract k is offered to micro-grid q and zero otherwise. The index set K will denote the set of all contracts and $K_0 \subseteq K$, the set of contracts suggested by the GenCo, i.e., $K \setminus K_0$ is the set of contracts suggested by the competitors of the GenCo. We denote the set of contracts that the GenCo can offer as $\mathcal{Z} \subseteq \{0,1\}^{|Q|\times|K_0|}$. For instance, $\mathcal{Z}$ could contain all binary vectors, where the GenCo offers a fixed number of contracts N_q to each micro-grid $q \in Q$. In that case, we would have $\mathcal{Z} = \{Z \in \{0,1\}^{|Q|\times|K_0|} : \sum_{k\in K_0} Z_{qk} \geq N_q,\ \forall q \in Q\}$.

For each micro-grid $q \in Q$, the binary variable z_{qk} indicates if micro-grid q subscribes to contract k. Notice that micro-grid q interacts with the GenCo only if it subscribes to one of its contracts, that is, if $\sum_{k\in K_0} z_{qk} \geq 1$. We will introduce local versions of the variables

x, y, called $\tilde{x}$ and $\tilde{y}$ that describe locally the power status. In particular, $x = \tilde{x}$ and $y = \tilde{y}$ only if a contract with the GenCo is subscribed.

Summarizing, the deterministic bilevel problem below considers the following optimization variables:

- Z_{qk}: 1 if contract k is offered to micro-grid q (*leader*)
- z_{qk}: 1 if contract k is subscribed by micro-grid q (*follower*)
- $\tilde{x}_{qt}$: power consumed by micro-grid q during period t (*follower*)
- $\tilde{y}_{qt}$: power produced by micro-grid q during period t (*follower*)
- x_{qt}: power consumed by micro-grid q during period t from the *GenCo* (*follower*)
- y_{qt}: power produced by micro-grid q during period t for the *GenCo* (*follower*)

We are now in measure to formulate our bilevel program:

$$\min \quad F(x,y) - \sum_{q\in Q}\sum_{k\in K_0} z_{qk}(c_k + \sum_{t\in\mathcal{T}}(f_{kt}x_{qt} - g_{kt}y_{qt})) \tag{A.2a}$$

$$\text{s.t.} \quad Z \in \mathcal{Z} \tag{A.2b}$$

$$(x_q, y_q, z_q) \in \underset{z_{q\cdot}\in\{0,1\}^K, \tilde{x}_q, \tilde{y}_q}{\arg\min} \sum_{k\in K} z_{qk}(c_k + \sum_{t\in\mathcal{T}}(f_{kt}\tilde{x}_{qt} - g_{kt}\tilde{y}_{qt})), \quad \forall q \in Q \tag{A.3a}$$

$$\text{s.t.} \quad (\tilde{x}_q, \tilde{y}_q) \in \mathcal{M}_q, \tag{A.3b}$$

$$x_q = \tilde{x}_q \sum_{k\in K_0} z_{qk}, \tag{A.3c}$$

$$y_q = \tilde{y}_q \sum_{k\in K_0} z_{qk}, \tag{A.3d}$$

$$z_{qk} \leq Z_{qk}, \quad k \in K \tag{A.3e}$$

$$\sum_{k\in K} z_{qk} = 1, \tag{A.3f}$$

where the constraints (A.2b), (A.3b)-(A.3f) are as explained below. The objective function of the leader, (A.2a), minimizes the production cost already mentioned in (A.1) combined with transaction costs related to the micro-grids. Similarly, the objective functions of the micro-grids, (A.3a), amounts to minimizing their total transaction costs with respect to all GenCos, not only the one represented by the leader. Hence, (A.3a) involves all contracts in K while (A.2a) considers only the contracts in K_0. Constraints (A.2b) and (A.3b) restrain, respectively, the set of contracts offered by the GenCo and the feasible power consumption/production for the micro-grids. Constraints (A.3c) and (A.3d) model that $x_q = \tilde{x}_q$ and $y_q = \tilde{y}_q$ if and only if micro-grid q subscribes to one of the contracts offered by the GenCo. Constraints (A.3e) impose that only the contract offered by the GenCo can be subscribed by the micro-grid, while constraints (A.3f) forces each micro-grid to select exactly one contract.

A.3 MILP reformulations

In this section we will provide two reformulations of the bilevel problem (A.2)-(A.3). In order to understand these, it is to be observed that problem (A.2)-(A.3) is not "unequivocally" defined in the sense that it is not clear what is meant with min in (A.2a), unless (x_q, y_q, z_q) in (A.3a) is always unique. A classic way to remove this ambiguity is to consider a so called optimistic formulation wherein among all (x_q, y_q, z_q) in the lower level problem (A.3) optimal set, the most advantageous for the leader is chosen. The pessimistic formulation then corresponds to picking the most disadvantageous case. Obviously both notions coincide whenever the lower level problem has a unique solution. We refer to Dempe et al. (2015) for further information on this topic. It is generally recognized that the pessimistic formulation is significantly harder to analyze than the optimistic formulation. In section A.3.1 we will suggest a reformulation that is neither optimistic nor pessimistic, but somewhere in between. We also suggest a formulation leading to optimistic solutions in section A.3.2 for comparative reasons. In particular the numerical experiments carried out in section A.5 will show that results from both reformulations differ, hence implying that the arg min in (A.3a) is indeed not unique and justifying that care should be taken in the interpretation.

A.3.1 Heuristic reformulation

The bilinear optimization problem described in the previous section involves binary variables and non-linear constraints, all that being built on the top of the already difficult optimization problem (A.1). Hence, we address the problem heuristically rather than exactly and we show below how it is possible to exploit the somewhat simple linking constraints (A.3e) to provide a heuristic reformulation for the bilinear problem.

We propose below a one-level reformulation of the bilevel problem that may lead to solutions that are not solutions to the optimistic problem. The key aspect of our reformulation relies on pre-processing. Specifically, for each $k \in K$ and $q \in Q$, we solve the restricted follower problem where z_{qk} is equal to 1, namely:

$$\begin{aligned} \min_{\tilde{x}_q, \tilde{y}_q} \quad & c_k + \sum_{t \in \mathcal{T}} (f_{kt}\tilde{x}_{qkt} - g_{kt}\tilde{y}_{qkt}) \\ \text{s.t.} \quad & (\tilde{x}_{qk}, \tilde{y}_{qk}) \in \mathcal{M}_q, \end{aligned}$$

Let $(\overline{x}_{qk}, \overline{y}_{qk})$ be an optimal solution of the above problem and $\overline{C}_{qk}$ be its solution cost. Then, a heuristic solution to the bilevel problem (A.2)-(A.3), which ensures the optimality of the lower level, can be found by solving:

$$
\begin{aligned}
\min_{z,Z} \quad & F(x,y) - \sum_{q\in Q}\sum_{k\in K_0} \overline{C}_{qk} z_{qk} && \text{(A.4a)}\\
\text{s.t.} \quad & x_{qt} = \sum_{k\in K_0} \overline{x}_{qkt} z_{qk}, \quad \forall q \in Q, t \in \mathcal{T} && \text{(A.4b)}\\
& y_{qt} = \sum_{k\in K_0} \overline{y}_{qkt} z_{qk}, \quad \forall q \in Q, t \in \mathcal{T} && \text{(A.4c)}\\
& z_{qk} \le Z_{qk}, \quad \forall q \in Q, k \in K && \text{(A.4d)}\\
& \sum_{k\in K} z_{qk} = 1, \quad \forall q \in Q && \text{(A.4e)}\\
& z_{qk} \le 1 - Z_{q\ell}, \quad \forall k \in K, l \in K, q : \overline{C}_{qk} > \overline{C}_{ql} && \text{(A.4f)}\\
& Z \in \mathcal{Z} && \text{(A.4g)}\\
& z \in \{0,1\}^{Q\times K}
\end{aligned}
$$

The objective (A.4a) is obtained from (A.2a) by replacing the micro-grid cost for the GenCo by $\overline{C}_{qk} z_{qk}$. Similarly, (A.4b) and (A.4c) are obtained from (A.3c) and (A.3d) by replacing the variables $\tilde{x}$ and $\tilde{y}$ with their fixed values $\overline{x}$ and $\overline{y}$ computed in the pre-processing phase. It is exactly in these constraints that lies the heuristic aspect of our reformulation since the followers no longer choose the optimal solution that most benefits the leader, but instead take the one computed in the pre-processing phase, when (A.3a) admits several solutions. Finally, constraint (A.4f) ensures that the micro-grids choose the contracts that lead to the cheapest solutions. Indeed if $Z_{ql} = 1$ and the cheap solution $\bar{C}_{ql}$ is available, any costlier solution is ruled out by equation (A.4f).

A.3.2 Comparative exact formulation

To assess the quality of our heuristic approach, we have also devised an exact one-level reformulation for the problem where (A.4b) and (A.4c) are replaced by non-linear constraints involving the aforementioned decision variables $\tilde{x}$ and $\tilde{y}$ that are restricted by the sets $\mathcal{M}_q$ for each $q \in Q$. In addition, the formulation contains restrictions enforcing $\tilde{x}$ and $\tilde{y}$ to be optimal for the subproblems.

$$\min_{z,Z,\tilde{x},\tilde{y}} \quad F(x,y) - \sum_{q\in Q}\sum_{k\in K_0} \overline{C}_{qk} z_{qk} \tag{A.5a}$$

$$\text{s.t.} \quad \text{(A.4d)} - \text{(A.4g)}$$

$$x_{qt} = \sum_{k\in K_0} \tilde{x}_{qkt} z_{qk}, \quad \forall q \in Q, t \in \mathcal{T} \tag{A.5b}$$

$$y_{qt} = \sum_{k\in K_0} \tilde{y}_{qkt} z_{qk}, \quad \forall q \in Q, t \in \mathcal{T} \tag{A.5c}$$

$$\overline{C}_{qk} = \ c_k + \sum_{t\in\mathcal{T}} (f_{kt}\tilde{x}_{qkt} + g_{kt}\tilde{y}_{qkt}), \quad \forall q \in Q, \forall k \in K \tag{A.5d}$$

$$(\tilde{x}_{qk}, \tilde{y}_{qk}) \in \mathcal{M}_q \tag{A.5e}$$

$$z \in \{0,1\}^{Q\times K}$$

Constraints (A.5b) and (A.5c) contain bilinear terms that can be linearized using classical techniques. Specifically, let us introduce variables X_{qkt} and Y_{qkt}, respectively equal to products $\tilde{x}_{qkt} z_{qk}$ and $\tilde{y}_{qkt} z_{qk}$. These variables can be substituted in constraints (A.5b) and (A.5c), adding also the constraints

$$\begin{aligned}
&X_{qkt} \le \tilde{x}_{qkt} \\
&X_{qkt} \le M z_{qk} \\
&X_{qkt} \ge \tilde{x}_{qkt} - M(1 - z_{qk}) \\
&Y_{qkt} \le \tilde{y}_{qkt} \\
&Y_{qkt} \le M z_{qk} \\
&Y_{qkt} \ge \tilde{y}_{qkt} - M(1 - z_{qk}),
\end{aligned}$$

that models the products through linear constraints involving big-M coefficients. The difficulty of the resulting MILP is that big-M coefficients often lead to numerical instability and weak LP relaxations. In our numerical experiments we have observed no specific difficulties related to the big-M coefficients.

A.4 Stochastic extension of the model

Power generation problems that involve renewable intermittent energy like wind and solar are subject to uncertainty, since the output of the renewable power plants depends on the weather conditions. This power output is therefore only partially known and should in principle be considered uncertain. We will set up a model in this section that accounts for uncertainty in generation. The resulting bilevel stochastic optimization problem will again become tractable due to a reformulation akin to the one used in the deterministic case. Obviously the main impact of considering uncertainty is that the preprocessing step will become much harder.

A.4.1 Model

The most important assumption for our stochastic model below is how we set up the structure of information. The main assumption, making a reformulation as in the deterministic case feasible, is that we will restrict the choice of contracts to the first stage. This means that all contracts have to be decided prior to observing any uncertainty. The second choice involves the structure of information within the remaining stochastic program. Therein we will assume that decent weather forecasts are available for an upcoming day and hence that all information for a given day is known at the beginning of that day. The stochastic program therefore has as many stages as there are days. In what follows we will denote with ξ_j the uncertainty related to day $j \in \{1, \dots, J\}$. Let us summarize these assumptions:

- *First stage*
  - The GenCo chooses a set of contracts that are compatible with its objective.
  - Each micro-grid chooses a contract among the contracts offered by the GenCos.
- *Subsequent stages* The exact productions and demands are known for all entities for all time periods that belong to the current day. Hence, the GenCo can produce the electricity and the micro-grids manage their elastic loads and batteries according to the chosen contracts and generated power.

A second important assumption that we will make is that uncertainty ξ has finite support. It can therefore be represented by a finite set of scenarios Ξ and uncertainty can be assumed to be represented by a scenario tree. For any given scenario $\xi \in \Xi$, we define $x(\xi)$ and $y(\xi)$ as the scenario dependent power flows between the GenCo and micro-grids. Following e.g., Rockafellar (2017), we choose to not make explicit the non-anticipativity constraints which can be appended to the model through an additional abstract subspace to which the decisions should belong. Obviously on the scenario tree these conditions are readily enforced. As stated earlier, the optimization problem has as many stages as days and all information within a day is assumed to be known.

The introduction of uncertainty affects both the objective function and the constraints of the follower problem, as well as the objective function of the leader. Thanks to our specific measurability assumption concerning the days, uncertainty in constraints has an easy interpretation. In particular the abstract set $\mathcal{M}_q(\xi)$ (covering for instance power balance equations) is such that $\mathcal{M}_q(\xi) = \prod_{j=1}^{J} \mathcal{M}_q(\xi_j)$. Consequently, we can write $(\tilde{x}_q(\xi), \tilde{y}_q(\xi)) \in \mathcal{M}_q(\xi)$ a.s., which for a random realization at day j, simply means that $(\tilde{x}_q(\xi_j), \tilde{y}_q(\xi_j)) \in \mathcal{M}_q(\xi_j)$ as a plain deterministic constraint.

As uncertainty is also present in the objective function, this calls for a consideration of a tradeoff between risk and average performance (see e.g., Rockafellar and Uryasev (2013) for a thorough discussion on such matters). As is common in practice, we will adopt a Markowitz type of idea so that we will weigh a measure of risk and average performance. To this end we will endow ourselves with a convex weight λ, that will weigh the average performance and the risk measure for the leader (the micro-grids are assumed risk-neutral for now, we discuss how to relax this assumption later). We choose to use the CVaR_ϵ as a measure of risk as it preserves convexity and moreover allows for a linear reformulation due to the following characterization (e.g., Rockafellar and Uryasev (2000)):

$$\mathrm{CVaR}_\epsilon[X(\xi)] = \min_v \left\{ v + \frac{1}{1-\epsilon}\mathbb{E}[(X(\xi)-v)^+] \right\} \tag{A.7}$$

for any random variable X and probability level $\epsilon \in (0,1)$.

We can now present the following risk-averse version of the bilevel energy management problem:

$$\min \quad \lambda\mathbb{E}\left[F(x(\xi),y(\xi)) - \sum_{q\in Q}\sum_{k\in K_0}(c_k + \sum_{t\in\mathcal{T}}(f_{kt}x_{qt}(\xi) - g_{kt}y_{qt}(\xi)))z_{qk}\right] \tag{A.8a}$$

$$+ (1-\lambda)\mathrm{CVaR}_\epsilon\left[F(x(\xi),y(\xi)) - \sum_{q\in Q}\sum_{k\in K_0}(c_k + \sum_{t\in\mathcal{T}}(f_{kt}x_{qt}(\xi) - g_{kt}y_{qt}(\xi)))z_{qk}\right] \tag{A.8b}$$

$$\text{s.t.} \quad Z \in \mathcal{Z} \tag{A.8c}$$

$$(x_q(\xi), y_q(\xi), z_q) \in \underset{z_{q\cdot}\in\{0,1\}^K,\tilde{x}_q,\tilde{y}_q}{\arg\min} \quad \mathbb{E}\left[\sum_{k\in K}(c_k + \sum_{t\in\mathcal{T}}(f_{kt}x_{qt}(\xi) - g_{kt}y_{qt}(\xi)))z_{qk}\right], \quad q\in Q \tag{A.9a}$$

$$\text{s.t.} \quad (\tilde{x}_q(\xi), \tilde{y}_q(\xi)) \in \mathcal{M}_q(\xi) a.s., \quad \xi\in\Xi \tag{A.9b}$$

$$x_q(\xi) = \tilde{x}_q(\xi)\sum_{k\in K_0} z_{qk}, \quad \xi\in\Xi \tag{A.9c}$$

$$y_q(\xi) = \tilde{y}_q(\xi)\sum_{k\in K_0} z_{qk}, \quad \xi\in\Xi \tag{A.9d}$$

$$z_{qk} \le Z_{qk}, \quad k\in K_0, \xi\in\Xi \tag{A.9e}$$

$$\sum_{k\in K} z_{qk} = 1, \quad \xi\in\Xi \tag{A.9f}$$

A.4.2 Heuristic reformulation

As in the deterministic case, the problem (A.8)-(A.9) can be solved heuristically by solving follower problems in a pre-processing phase. For each $k \in K$ and $q \in Q$, we solve the restricted follower problem where z_{qk} is equal to 1

$$\min_{\tilde{x}_q(\xi),\tilde{y}_q(\xi)} \quad c_k + \mathbb{E}\left[\sum_{t\in\mathcal{T}}(f_{kt}\tilde{x}_{qt}(\xi) - g_{kt}\tilde{y}_{qt}(\xi))\right] \tag{A.10a}$$

$$\text{s.t.} \quad (\tilde{x}_q(\xi), \tilde{y}_q(\xi)) \in \mathcal{M}_q(\xi) a.s., \quad \forall\xi\in\Xi, \tag{A.10b}$$

under non-anticipativity constraints. Let $(\overline{x}_{qk}, \overline{y}_{qk})$ be an optimal solution and $\overline{\mathbb{C}}_{qk}$ be its optimal value, namely,

$$\overline{\mathbb{C}}_{qk} = c_k + \mathbb{E}\left[\sum_{t\in\mathcal{T}}(f_{kt}\overline{x}_{qt}(\xi) + g_{kt}\overline{y}_{qt}(\xi))\right].$$

Then, a heuristic solution to the stochastic bilevel problem (A.8)-(A.9), which ensures the optimality of the lower level, can be found by solving:

$$\begin{aligned}
\min_{z,Z} \quad & \lambda\mathbb{E}\left[F(x(\xi),y(\xi)) - \sum_{q\in Q}\sum_{k\in K_0}(c_k + \sum_{t\in\mathcal{T}}(f_{kt}x_{qt}(\xi) - g_{kt}y_{qt}(\xi)))z_{qk}\right] \\
& + (1-\lambda)\mathrm{CVaR}_\epsilon\left[F(x(\xi),y(\xi)) - \sum_{q\in Q}\sum_{k\in K_0}(c_k + \sum_{t\in\mathcal{T}}(f_{kt}x_{qt}(\xi) - g_{kt}y_{qt}(\xi)))z_{qk}\right] && \text{(A.11a)} \\
\text{s.t.} \quad & z_{qk} \le Z_{qk}, \quad \forall q\in Q, k\in K_0 && \text{(A.11b)} \\
& \sum_{k\in K} z_{qk} = 1, \quad \forall q \in Q && \text{(A.11c)} \\
& x_{qt}(\xi) = \sum_{k\in K_0}\overline{x}_{qkt}(\xi)z_{qk}, \quad \forall q\in Q, t\in\mathcal{T}, \xi\in\Xi && \text{(A.11d)} \\
& y_{qt}(\xi) = \sum_{k\in K_0}\overline{y}_{qkt}(\xi)z_{qk}, \quad \forall q\in Q, t\in\mathcal{T}, \xi\in\Xi && \text{(A.11e)} \\
& z_{qk} \le 1 - Z_{\ell q}, \quad \forall k,q : \overline{\mathbb{C}}_{qk} > \overline{\mathbb{C}}_{ql} && \text{(A.11f)} \\
& Z \in \mathcal{Z} && \text{(A.11g)} \\
& z \in \{0,1\}^{Q\times K}. && \text{(A.11h)}
\end{aligned}$$

The above program can be further reformulated by replacing CVaR with its characterization (A.7) and by introducing the optimization variables $w(\xi)$ to linearize the function $[\cdot]^+$. This yields the regular stochastic program:

$$\begin{aligned}
\min_{z,Z,w(\xi)} \quad & \mathbb{E}\left[\lambda F(x(\xi),y(\xi)) - \lambda\sum_{q\in Q}\sum_{k\in K_0}(c_k + \sum_{t\in\mathcal{T}}(f_{kt}x_{qt}(\xi) - g_{kt}y_{qt}(\xi)))z_{qk} + (1-\lambda)\left(v + \frac{w(\xi)}{1-\epsilon}\right)\right] \\
\text{s.t.} \quad & \text{(A.11b)} - \text{(A.11h)} \\
& w(\xi) \ge F(x(\xi),y(\xi)) - \sum_{q\in Q}\sum_{k\in K_0}(c_k + \sum_{t\in\mathcal{T}}(f_{kt}x_{qt}(\xi) - g_{kt}y_{qt}(\xi)))z_{qk} - v, \quad \forall\xi\in\Xi \\
& w(\xi) \ge 0, \quad \forall \xi\in\Xi.
\end{aligned}$$

We mention that a similar MILP reformulation can be obtained whenever the micro-grids are also assumed to be risk-averse. For instance, we could consider the same Markowitz type model and endow ourselves with a set of convex multipliers $\{\lambda^q\}_{q\in Q}$ weighting the expectation and CVaR_ϵ for each micro-grid. The resulting counterpart of (A.10) is more demanding computationally due to the difficulty of decomposing the problem per scenario.

This difficulty is well-known in risk-averse stochastic optimization, see for instance the recent Rockafellar (2017). As this goes beyond the scope of this paper, we disregard this extension in what follows.

A.4.3 Comparative reformulation

As in the deterministic case, the above heuristic reformulation can be made exact by replacing constraints (A.11d) and (A.11e) with the bilinear constraints

$$x_{qt}(\xi) = \sum_{k \in K_0} \tilde{x}_{qkt}(\xi) z_{qk}, \quad \forall q \in Q, t \in \mathcal{T}, \xi \in \Xi \tag{A.13}$$

$$y_{qt}(\xi) = \sum_{k \in K_0} \tilde{y}_{qkt}(\xi) z_{qk} \forall q \in Q, t \in \mathcal{T}, \xi \in \Xi \tag{A.14}$$

and adding constraints

$$\overline{\mathbb{C}}_{qk} = c_k + \mathbb{E}\left[\sum_{t \in \mathcal{T}} (f_{kt} \tilde{x}_{qt}(\xi) + g_{kt} \tilde{y}_{qt}(\xi))\right], \quad \forall q \in Q, k \in K \tag{A.15}$$

$$(\tilde{x}_{qk}(\xi), \tilde{y}_{qk}(\xi)) \in \mathcal{M}_q(\xi), \quad \forall q \in Q, k \in K, \xi \in \Xi. \tag{A.16}$$

Moreover optimization is now to be carried out over $(z, Z, \tilde{x}(\xi), \tilde{y}(\xi))$. Evidently the resulting problem is significantly harder!

A.5 Case study based on thermal power unit-commitment

We assess below our heuristic reformulation on a power generation problem where F represents a thermal unit-commitment problem, following closely Carrión and Arroyo (2006). The latter minimizes the total operation cost, which is defined as the sum of the production cost, the startup cost, and the shutdown cost. The production cost is piece-wise linear, while the startup cost is piece-wise linear approximation of an exponential function of the off-line time prior to the startup. Since the MILP model is rather technical, and because computing $F(x, y)$ is not the focus of this work, we refer the interested reader to, e.g., Carrión and Arroyo (2006) for the details about the mixed-integer linear program behind F. We also refer to Tahanan et al. (2015) and the many references therein for further alternatives to the model considered here. In addition, we detail the constraints used in our micro-grid model in A.7.

A.5.1 Data

A.5.1.1 General system

The time horizon is one week starting on Monday at 12 am, each day is split in periods of one hour, i.e., $|\mathcal{T}| = 168$. The data used for the unit-commitment part is provided by

Carrión and Arroyo (2006), 50 generators are used giving a daily fixed load of 135,5 GWh.

A.5.1.2 Considered contracts

Several types of contracts are proposed combining the possibility of lower prices during nighttime and/or the possibility of lower prices during the weekend. This leads to a total of four types of contracts. The competitors offer one contract of each type to each micro-grid. The GenCo generates four possible contracts of each type, thus $|K_0| = 16$ and $|K| = 20$. For all contracts, the average cost of 1kW is about 0,10 EUR[2].

A.5.1.3 Description of the micro-grids

In what follows we will define devices as an abstract group of objects fulfilling a purpose of consumption, production or storage under time constraints within a micro-grid. The devices considered, their consumption and the time periods where they are used fit numbers reported in a recent study of electricity demand/offer equilibrium in France[2].

- Consumption devices are devices appearing in a common household, which we split into three further sub-classes:
  - Constant: these devices consume power constantly. They represent 30% of the daily consumption.
  - Comfort: devices that consume power during a predetermined period each day, representing 30% of the daily consumption.
  - Elastic: devices for which predetermined windows of usage are defined for each day as well as a total daily load. They represent 40% of the daily consumption.

  The periods where comfort devices are used and elastic devices can be used as well as their consumptions are randomly generated such that the graph of the total consumption of a micro-grid fits a classical duck curve for each day. The average consumption of 200 devices over one week is 1GWh, which corresponds to the consumption of 5000 common (European) households. Consumption of elastic devices can be reorganized to reduce costs by taking advantage of low hourly prices, production devices and storage devices. Figure A.1 illustrates the hourly demand of the micro-grid composed of 10000 devices.

- Production devices are equally split into solar panels and wind turbines. The production capacity of each production device is generated randomly. For each micro-grid, their production capacity is sufficient to satisfy on average 50% of the consumption of micro-grids.

- Storage devices are of a single type, their capacity is sufficient to store on average 30% of the daily electricity load of a micro-grid. We consider that during the storage process

[2]http://www.rte-france.com/fr/document/bilan-previsionnel-de-l-equilibre-offredemande-2016

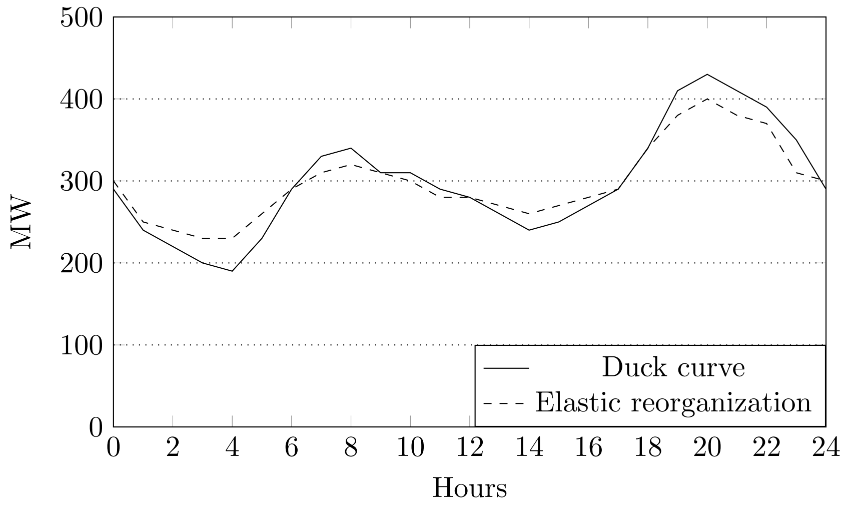


FIGURE A.1: Duck curve example for a micro-grid with 10000 devices

there is a total loss of 10% of energy due to physical constraints which corresponds to the storage efficiency of state-of-the-art domestic batteries such as Tesla's Powerwall[3].

In the largest instance, 90,000 devices are considered, corresponding to over $2,000,000$ households. The total consumption of devices in this instance is on average 64 GWh per day. As on average 50% of the daily load is supplied by production devices, about 32 GWh must be provided from the global network each day.
The penetration of the MGs in the total load of the GenCo is defined as

$$penetration = \frac{load\ of\ the\ MGs}{total\ load\ for\ the\ GenCo}$$

If all MG considered choose a contract from the GenCo, the penetration rate is about 19%.

A.5.1.4 Uncertainty

Four simple independent scenarios are considered based on the possibility of good sunshine and good wind that influence production of micro-grids. Weather conditions are considered as identical each day of the week in a scenario. In bad conditions (no sunshine or no wind), a production device produces only 50% of its maximum production capacity. Notice that this implies that the multi-stage structure of the problem is simplified to a two-stage problem for which no non-anticipativity constraints are required. The value of ϵ used in CVaR is equal to 0.1, representing a high aversion to risk. The full instance details are available from the authors upon request.

[3]https://www.tesla.com/fr_BE/powerwall?redirect=no

# Devices	Avg. Obj	σ	time(s)
100	34.99	2.25	0.15
200	37.41	3.67	0.32
1000	207.51	14.97	1.91
2000	332.93	21.32	3.95
5000	653.61	37.36	14.57
10000	957.65	40.56	24.01

TABLE A.1: Average computation time and objective value for the micro-grid power planning preprocessing step.

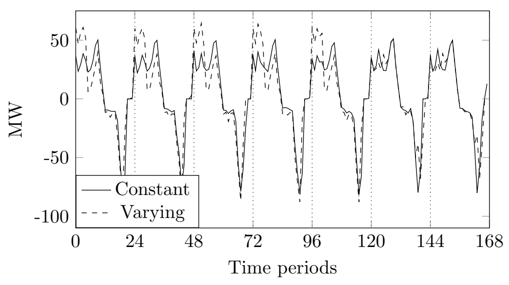
A.5.2 Numerical results

All the methods were implemented using Julia 0.4.6 and ILOG CPLEX 12.7 as MIP solver. Tests were made on a 4-core i7 2.30 GHz processor with 16Go of RAM memory. Maximum computation time is set to 3600s.

Table A.1 presents the average computation times for solving the follower problems. Times and objectives are averaged over the 20 possible contracts for the 5 instances of micro-grids of each size. The objectives are the average consumption costs for micro-grids of a given size which is also the average income for a GenCo supplying those micro-grids. They are provided in euros. The standard deviation (σ) of the objective value is also provided. These results show that solving the restricted follower problem can be done fairly quickly. The objective value for all contracts is positive and for all instances considered, there exists at least one contract of the GenCo that is cheaper and one that is more expensive than those of the competitors. Each micro-grid is financially interesting for the GenCo if the additional production costs are not too high. In terms of size, the formulation of the follower problem on a micro-grid with 100 devices and a single contract available contains 59136 variables and 35390 constraints.
Figure A.2 illustrates the optimal demand strategies over time of two micro-grid of 5000 devices for two contracts. The left graph represents a micro-grid composed of households and the right one a micro-grid composed of offices. One contract considers constant buying and selling prices over time (full line) and the other considers smaller prices during nighttime and weekends (dashed line). The different demand strategies will influence the production of the GenCo if the two contracts are proposed.

Notice that the curves have a different aspect than a classical duck curve illustrated in Figure A.1. This is due to production devices as solar panel that can be active only during daytime. As a result, the demand is negative almost all daytime and reaches its biggest values during nighttime. The right hand side graph is mostly flat during the weekend as the micro-grid considered is composed of offices with very low consumption during this period.

Table A.2 presents results solving small instances with the exact (*Exact* optimistic) and the heuristic (*Heuristic*) bilevel reformulations. Here, the value of λ is set to 1 in order to optimize the expectation. The first two columns report the number of devices in the



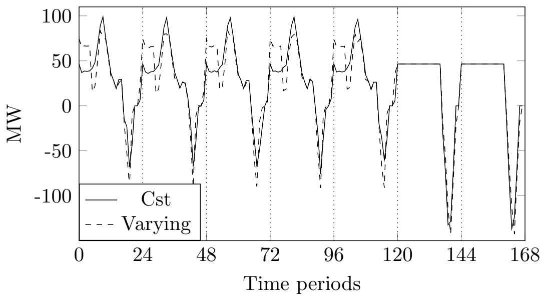


FIGURE A.2: Micro-grid demand management

considered micro-grids and the number of micro-grids of each size. Hence, the first line considers one micro-grid with 100 devices and the last line considers five micro-grids with 100 devices plus five micro-grids with 200 devices.

It has been observed in the results that for each instance, both approaches (*Heuristic* and *Exact*) select the same contracts for each micro-grid.

The difference between optimal values lies in the extra production cost there is for the GenCo to satisfy the demand of the micro-grids who subscribed a contract. Specifically, in *Exact*, the planning of elastic devices of those micro-grids can be reorganized to reduce the cost of the UC problem, preserving optimal cost for the micro-grids. This observation shows that the bilevel problem (A.8)-(A.9) (or (A.2)-(A.3)) is indeed ambiguous in so much that the argmin of problem (A.9) is not unique. Consequently the pessimistic and optimistic versions of the model differ and our heuristic reformulation offers something in between. However, this gap, as reported in the column $\Delta(F - UC)$ is not very large. Indeed, let us denote by (x_{opt}, y_{opt}) and (x_{heur}, y_{heur}) the power generation returned by *Exact* and *Heuristic*, respectively. Columns $F(x_{opt}, y_{opt}) - F(0,0)$ and $F(x_{heur}, y_{heur}) - F(0,0)$ correspond to GenCo's extra production costs due to the consumption of the micro-grids in the solutions of *Exact* and *Heuristic* respectively. The value

$$\Delta(F - UC) = \frac{F(x_{heur}, y_{heur}) - F(x_{opt}, y_{opt})}{F(x_{opt}, y_{opt}) - F(0,0)}$$

is the relative gap between the costs of extra production in *Heuristic* and *Exact*. The reported computation times of *Heuristic* do not include the preprocessing times.

Computation times of *Heuristic* (without considering preprocessing) are quite stable while those of *Exact* tend to grow very quickly and the last instance considering 10 micro-grids cannot be solved in one hour. Concerning the instance with a single micro-grid with 200 devices, the extra production cost is 0 because the micro-grid chose a contract of another company. The gap on extra production cost is very low in all tests. The *Heuristic*/*Exact* formulation of the instance with five micro-grids of 100 devices contains 200/300 binary variables, 1680/37255 continuous variables and 1975/279725 constraints. As no variables are related to devices in the *Heuristic* formulation, the number of devices does not impact its the size.

Test set		*Exact*		*Heuristic*		$\Delta(F - UC)$ (%)
# Devices	nb	$F(x_{opt}, y_{opt}) - F(0,0)$	time(s)	$F(x_{heur}, y_{heur}) - F(0,0)$	time(s)	
100	1	8.134968	76.26	8.146005	26.9	0.136
100	2	29.413995	202.3	29.438447	28.63	0.083
100	3	67.440871	312.46	67.471015	28.07	0.045
100	4	52.290345	678.9	52.334735	26.67	0.085
100	5	24.543927	690.63	24.588317	27.22	0.181
200	1	0	135.01	0	21.16	-
200	2	71.09899	242.34	71.1142	25.88	0.021
200	3	128.69007	635.02	128.717851	28.05	0.022
200	4	188.672761	1684	188.715209	25.85	0.022
200	5	171.455653	1919.43	171.498219	25.95	0.025
100-200	1	60.079001	223.74	60.090038	25.7	0.018
100-200	2	100.510032	893.82	100.550933	25.93	0.041
100-200	3	196.12615	1697.23	196.184324	27.19	0.030
100-200	4	240.957646	3520.43	241.045643	27.38	0.037
100-200	5	-	> 3600	196.081488	28.7	-

Table A.2: Comparing *Exact* and *Heuristic* on small instances.

Table A.3 reports the results of *Heuristic* on larger instances. We consider three values for λ: 1, 0.5 and 0. When set to 1 (resp. 0), the formulation optimizes the expectation (resp. CVaR_ϵ). We have chosen a value ϵ of 0.1, representing a high aversion to risk. In each instance, five micro-grids of each size are used. Columns E report the expectation of the optimal solutions, columns CVaR report their conditional value-at-risk, both are given in thousands of €. Columns sold report the number of contracts of the GenCo chosen by the MG in the optimal solution found. These values are reported only for $\lambda = 1$ as they were identical for other values of λ after rounding. The first line considers only the original UC problem while the last line considers 20 micro-grids (5 of each size) with a maximum possible penetration of 19%. Notice the negative solution cost of the last line, which is obtained from a revenue from the micro-grids that is higher than the production cost. Computation times tend to grow slowly when adding micro-grids and stay close to the computation time of the original UC problem when $\lambda = 1$. For $\lambda = 0.5$, the expectation is similar, the CVaR decreases significantly for some instances and the computation time increases of about 20%. With $\lambda = 0$, results for the expectation and CVaR are similar than for $\lambda = 0.5$ and computation times are more than doubled for most instances in comparaison of $\lambda = 1$.

Our formulation does not increase much the difficulty of the UC problem on the tested instances when minimizing the expectation. Integrating CVaR in the objective function with $\lambda = 0.5$ seems interesting as it is not very time consuming and reduces CVaR almost to its best attainable value.

The number of contracts sold per instance reflects the quadratic aspect of production costs considered in the formulation for the UC Carrión and Arroyo (2006). Summing the numbers of contracts sold in instances where all micro-grids have the same size (lines 2 to 5 of Table A.3), a total of 19 contracts are sold to the 20 micro-grids. When solving the problem with all 20 micro-grids together (last line of the table), only 10 contracts are sold. In this solution, respectively 2, 1, 4 and 3 contracts are sold to the MGs of size 1000, 2000, 5000 and 10,000. The results of this last instance let us suppose the GenCo cannot afford having

Test set	$\lambda = 1$					$\lambda = 0.5$			$\lambda = 0$		
	E	CVaR	time(s)	sold	penetration	E	CVaR	time(s)	E	CVaR	time(s)
UC	3599	-	20.98	0	0	3599	-	20.98	3599	-	20.98
1000	3221	3580	26.31	5	1.2%	3221	3250	30.09	3221	3250	47.41
2000	2973	3304	22.98	4	2.1%	2973	3018	26.37	2973	3018	46.71
5000	1373	1526	24.38	5	6.0%	1374	1510	27.33	1375	1509	46.54
10000	-421	768	24.98	5	10.9%	-421	-144	29.87	-420	-145	56.54
1000-2000	2595	2884	25.00	9	3.3%	2595	2670	28.09	2595	2670	50.53
1000-2000-5000	378	960	25.61	14	9.1%	379	589	30.24	379	589	50.63
1000-2000-5000-10000	-2055	145	32.42	10	12.6%	-2055	-1768	44.19	-2053	-1769	82.29

TABLE A.3: Solution of *Heuristic* for larger instances.

a penetration rate higher than about 13% with the contracts proposed by the competitors. Even the small MG with 1000 devices go to competitors illustrating that the GenCo can only propose contracts more expensive than those of competitors. If many micro-grids must be supplied with electricity, the average production costs increases for the GenCo in the UC problem. At some point, the GenCo cannot afford proposing low price contracts to additional micro-grids, otherwise it would produce at a loss. Our approach prevents the GenCo from falling in this situation by proposing contracts that are more expensive than those of the competitors to micro-grids that are not financially interesting.

For each instance, the same MGs choose a contract of the GenCo for each value of λ considered but the contracts sold are not all identical. Table A.4 illustrates the solution of the instance with 5 micro-grids of size 5000. The contracts are represented in the columns and are grouped by type. Constant price, daytime-nighttime price, week-weekend prices and daytime-nighttime plus week-weekend prices. A blank dot appears when a contract is proposed to a micro-grid by the GenCo, a black dot appears when a micro-grid selects a contract. The five micro-grids select a contract from the GenCo for all values of λ but the contracts proposed and chosen for the fourth and fifth micro-grid vary. Integrating CVaR can discredit contracts that present a high risk in some scenarios.

A.6 Conclusions and perspectives

We have addressed a variant of power generation optimization problems where the GenCo interacts with micro-grids by buying or selling power to them. We have modeled the problem as a bilevel stochastic optimization problem, built on the top of the already difficult power generation optimization problem that the GenCo must solve to produce its energy. For realistic size data, solving the bilevel problem exactly is out of reach so that we have focused on a heuristic reformulation that produces a solution to the bilevel problem that is neither “optimistic” nor “pessimistic”. Although our numerical results exhibit that the resulting solution is indeed different from the “optimistic” solution, they also indicated that the gap is small (roughly 0.05 - 0.1 % variation on the upper level objective function). Interestingly enough, we could link this discrepancy between the optimistic bilevel solution and our solution to the existence of load-shifting devices within the micro-grid that are piloted differently in the “optimistic” situation.

Test set		Cst price				DN price				WWE price				DN and WWE price			
MG	λ	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	○				○		○		○	○	○	○	○		○	●
	0.5	○				○		○		○	○	○	○	○		○	●
	0	○				○		○		○	○	○	○	○		○	●
2	1	○				○		○	●		○	○	○			○	
	0.5	○				○		○	●		○	○	○			○	
	0	○				○		○	●		○	○	○			○	
3	1	○				○		○		○	○	○	○	○		○	●
	0.5	○				○		○		○	○	○	○	○		○	●
	0	○				○		○		○	○	○	○	○		○	●
4	1	○				○		○	●	○	○	○	○	○		○	○
	0.5	○				○		○	●	○	○	○	○	○		○	○
	0	○				○		○		○	○	○	○	○		○	●
5	1	○				○		○	○	●	○	○	○	○	○	○	○
	0.5	○	●			○		○	○		○	○	○	○	○	○	○
	0	○	●			○		○	○		○	○	○	○	○	○	○

Table A.4: Contracts proposed and selected to five MG with $|D| = 5000$.

We have assessed our reformulation on a case-study based on a thermal unit-commitment problem using realistic data. Our numerical results have confirmed the tractability of the heuristic approach, since the solution time of our reformulation is at most 50% higher than the one required for solving the thermal unit-commitment problem. In contrast, the solution of an exact reformulation of the bilevel problem increases the solution times by more than 10 000% in the presence of only 8 micro-grids. Our results also confirm the quality of the approximation provided by our heuristic, which provides a solution very close to the solution of the exact "optimistic" model. An interesting venue for future research would seek to extend the kind of reformulations proposed in this paper to other bilevel optimization problems with integer variables, which is a class of optimization problems that are notoriously difficult to solve exactly.

A.7 Micro-grids

For simplicity, this specific study considers the bus model for power flows, that is, the power network is not taken into consideration. We define two types of unit components in the micro-grid, which are called devices in the sequel. *Storage* devices typically represent batteries, whose status may switch between online (connected to the grid) and offline during the time horizon. During its offline periods, a storage device can be unloaded: for example, an electric vehicle is loaded during the night. During this time period, it can be used to store and serve power, but it must be fully loaded at the end of the night. During the day, the vehicle is used and its battery is emptied so that its storage level is low when it is back online. In our setting, we associate with each storage device a set of online time intervals. At the beginning of such a time interval, the charge level is a stochastic input parameter (for example, it depends on how much the vehicle has been used during the day). The required charge level at the end of the online period is modeled through more general parameters giving minimum and maximum acceptable charge levels for each online time step. We also assume that each storage device has limited capacity, charging and discharging speed and a power loss factor that is the proportion of power stored to the power consumed during the charge.

The *regular* devices come with stochastic consumptions and productions of power during each time step. Some of their consumption can be partially delayed (elastic demand). We model this feature by defining a set of time intervals for each device (for example, a water heater must heat the water during the night). During each of them, the required total power consumption is known, as a stochastic input data. The maximum power consumption of devices is limited during each time step.

The decisions to be taken in the micro-grid problem are, first, to choose a contract among those proposed by the GenCos. Then, for each regular device and each time step, the amount of elastic power consumption must be determined. For each *storage* device, the amounts of power consumed (to charge) and released must be fixed. We provide below a model for the problems faced by each one of the micro-grids.

Parameters Scenario-independent parameters:

- K: set of available contracts, defined by c_k, f_k and g_k. We assume that $\forall t \in \mathcal{T}$, $f_{kt} \geq g_{kt}$.
- D: set of non-storage devices. For each device $d \in D$:
  - Θ^d_D: set containing sets of time periods defining elastic consumption slots
- S: set of storage devices. For all $d \in S$:
  - $\bar{s}^d$: capacity of storage device d
  - $\bar{\ell}^d$: maximum power used to reload d during one time period
  - $\bar{u}^d$: maximum power released by d during one time period
  - α^d: power loss factor when charging d
  - Θ^d_S: set of time intervals when d is online (can be charged or discharged). We note, for all $\theta \in \Theta^d_S$, $\theta = [t^-(\theta), t^+(\theta)]$.
  - $\underline{S}^d_t, \bar{S}^d_t$: minimum and maximum charge level for d at time t.

Scenario-dependent parameters: For all scenarios $\xi \in \Xi$:

- For each device $d \in D$:
  - $\forall t \in \mathcal{T}$:
    - $b^d_t(\xi)$: power production of d during period t in scenario ξ
    - $r^d_t(\xi)$: power consumption of d during period t in scenario ξ

    * $\bar{w}_t^d(\xi)$: maximum possible elastic consumption of d during t
  - $\forall \theta \in \Theta_D^d$
    * $e_\theta^d(\xi)$: total elastic power demand of d during θ in scenario ξ
  - For each storage device $d \in S$ and online interval $\theta \in \Theta_S^d$: $I_\theta^d(\xi)$ is the initial stock level when d is plugged in.

Decision variables

- Stage 0:
  - $\forall k \in K$: $z_k = 1$ if contract k is chosen by the micro-grid, 0 otherwise
- Stage $t, t \in \mathcal{T}$:
  - $x_t(\xi)$: power consumed by the micro-grid during period t
  - $y_t(\xi)$: power produced by the micro-grid during period t
    * $\forall d \in D$, $w_t^d(\xi)$: elastic power consumed by d during t
    * For all $d \in S$:
      · $s_t^d(\xi)$: power stock in d at the end of t
      · $\ell_t^d(\xi)$: power consumed to charge device d during t
      · $u_t^d(\xi)$: power released by discharging device d during t

Formulation

$$\min \quad \mathbb{E}\left[\sum_{k\in K}\sum_{t\in\mathcal{T}} \left(f_{kt}x_t(\xi) + g_{kt}y_t(\xi) + c_k\right) z_k\right] \tag{A.17a}$$

$$\text{s.t.} \quad \sum_{k\in K} z_k = 1 \tag{A.17b}$$

$$x_t(\xi) - y_t(\xi) = \sum_{d\in D}\left(r_t^d(\xi) + w_t^d(\xi) - b_t^d(\xi)\right) + \sum_{d\in S}\left(\ell_t^d(\xi) - u_t^d(\xi)\right) \quad \forall t, \xi \tag{A.17c}$$

$$s_t^d(\xi) = s_{t-1}^d(\xi) + \alpha^d \ell_t^d(\xi) - u_t^d(\xi) \quad \forall d \in S, \theta \in \Theta_S^d, t \in \theta - \{t^-(\theta)\}, \xi \tag{A.17d}$$

$$s_{t^-(\theta)}^d(\xi) = I_\theta^d(\xi) + \alpha^d \ell_{t^-(\theta)}^d(\xi) - u_{t^-(\theta)}^d(\xi) \quad \forall d \in S, \theta \in \Theta_S^d, \xi \tag{A.17e}$$

$$\sum_{t\in\theta} w_t^d(\xi) = e_\theta^d(\xi) \quad \forall d \in D, \theta \in \Theta_D^d, \xi \tag{A.17f}$$

$$w_t^d(\xi) \le \bar{w}^d \quad \forall d \in D, t, \xi \tag{A.17g}$$

$$\underline{S}_t^d \le s_t^d(\xi) \le \bar{S}_t^d \quad \forall d \in S, t, \xi \tag{A.17h}$$

$$\ell_t^d \le \bar{\ell}^d \quad \forall d \in S, t \tag{A.17i}$$

$$u_t^d \le \bar{u}^d \quad \forall d \in S, t \tag{A.17j}$$

$$\ell_t^d(\xi) = u_t^d(\xi) = 0 \quad \forall d \in S, \theta \notin \Theta_S^d, t \in \theta \tag{A.17k}$$

$$z_k \in \{0,1\} \quad \forall k \in K \tag{A.17l}$$

$$x, y, w, r, s, l, u \ge 0 \tag{A.17m}$$

The objective of the problem (A.17a) is to minimize the total cost for the micro-grid, which is composed of the fixed cost of the contract, the cost of buying power from the GenCos minus the income obtained from selling the over-production. Constraint (A.17b) states that exactly one contract must be chosen. Constraints (A.17c) ensure that the power flow into/out of the micro-grid is equal to its production/consumption during each time step. In the right-hand-side, the summation over D (resp. S) represents the total consumption/production of regular (resp. storage) devices. Constraints (A.17d) and (A.17e) define the level of power stock for each device and time step. Constraints (A.17f) fix the correct total amount of power that must be consumed by a device during an elastic consumption interval. The instantaneous power consumed by a device is limited by constraints (A.17g). The acceptable stock levels are bound by constraints (A.17h). Constraints (A.17i) and (A.17j) define maximum charging and discharging speeds for the storage devices, while constraints (A.17k) are just a way to state that an offline device cannot be charged or discharged (the corresponding variables may as well be omitted in the model). The domains of the variables are given in constraints (A.17l) and (A.17m).

Bibliography

L. Adam, R. Henrion, and J. Outrata. On M-stationarity conditions in MPECs and the associated qualification conditions. *Mathematical Programming*, 168(1):229–259, Mar. 2018.

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

U. Al-Turki, C. Fedjki, and A. Andijani. Tabu search for a class of single-machine scheduling problems. *Computers & Operations Research*, 28(12):1223–1230, 2001.

S. M. Al-Yakoob and H. D. Sherali. A Column Generation Approach for an Employee Scheduling Problem with Multiple Shifts and Work Locations. *The Journal of the Operational Research Society*, 59(1):34–43, 2008.

H. K. Alfares. Survey, Categorization, and Comparison of Recent Tour Scheduling Literature. *Annals of Operations Research*, 127(1-4):145–175, Mar. 2004.

G. E. Asimakopoulou, A. L. Dimeas, and N. D. Hatziargyriou. Leader-Follower Strategies for Energy Management of Multi-Microgrids. *IEEE Transactions on Smart Grid*, 4(4): 1909–1916, Dec. 2013.

D. Aussel, P. Bendotti, and M. Pištěk. Nash equilibrium in a pay-as-bid electricity market: Part 1 existence and characterization. *Optimization*, 66(6):1013–1025, June 2017a.

D. Aussel, P. Bendotti, and M. Pištěk. Nash equilibrium in a pay-as-bid electricity market Part 2 - best response of a producer. *Optimization*, 66(6):1027–1053, June 2017b.

T. Aykin. Optimal Shift Scheduling with Multiple Break Windows. *Management Science*, 42(4):591–602, Apr. 1996.

T. Aykin. A composite branch and cut algorithm for optimal shift scheduling with multiple breaks and break windows. *Journal of the Operational Research Society*, 49(6):603–615, June 1998.

J. Bailey. Integrated days off and shift personnel scheduling. *Computers & Industrial Engineering*, 9(4):395–404, Jan. 1985.

K. R. Baker. Workforce Allocation in Cyclical Scheduling Problems: A Survey. *Operational Research Quarterly (1970-1977)*, 27(1):155–167, 1976.

E. Balas and M. C. Carrera. A Dynamic Subgradient-Based Branch-and-Bound Procedure for Set Covering. *Operations Research*, 44(6):875–890, Dec. 1996.

E. Balas and M. Padberg. Set Partitioning: A survey. *SIAM Review*, 18(4):710–760, Oct. 1976.

R. Baldacci, N. Christofides, and A. Mingozzi. An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. *Mathematical Programming*, 115(2):351–385, Oct. 2008.

R. Baldacci, E. Bartolini, A. Mingozzi, and R. Roberti. An exact solution framework for a broad class of vehicle routing problems. *Computational Management Science*, 7(3): 229–268, July 2010.

R. Baldacci, E. Bartolini, and A. Mingozzi. An Exact Algorithm for the Pickup and Delivery Problem with Time Windows. *Operations Research*, 59(2):414–426, Apr. 2011a.

R. Baldacci, E. Bartolini, A. Mingozzi, and A. Valletta. An Exact Algorithm for the Period Routing Problem. *Operations Research*, 59(1):228–241, Feb. 2011b.

R. Baldacci, S. U. Ngueveu, and R. Wolfler Calvo. The Vehicle Routing Problem with Transhipment Facilities. *Transportation Science*, 51(2):592–606, Dec. 2016.

R. Baldacci, A. Hill, E. A. Hoshino, and A. Lim. Pricing strategies for capacitated ring-star problems based on dynamic programming algorithms. *European Journal of Operational Research*, 262(3):879–893, Nov. 2017.

M. L. Balinski and R. E. Quandt. On an Integer Program for a Delivery Problem. *Operations Research*, 12(2):300–304, Apr. 1964.

F. Barahona and R. Anbil. On some difficult linear programs coming from set partitioning. *Discrete Applied Mathematics*, 118(12):3–11, Apr. 2002.

J. F. Bard and J. T. Moore. An algorithm for the discrete bilevel programming problem. *Naval Research Logistics (NRL)*, 39(3):419–435, Apr. 1992.

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, June 1998.

S. E. Bechtold and L. W. Jacobs. Implicit Modeling of Flexible Break Assignments in Optimal Shift Scheduling. *Management Science*, 36(11):1339–1351, Nov. 1990.

S. E. Bechtold and L. W. Jacobs. The equivalence of general set-covering and implicit integer programming formulations for shift scheduling. *Naval Research Logistics (NRL)*, 43(2): 233–249, 1996.

M. Boschetti and V. Maniezzo. A set covering based matheuristic for a real-world city logistics problem. *International Transactions in Operational Research*, 22(1):169–195, 2015.

M. A. Boschetti, A. Mingozzi, and S. Ricciardelli. A dual ascent procedure for the set partitioning problem. *Discrete Optimization*, 5(4):735–747, 2008.

M. A. Boschetti, V. Maniezzo, M. Roffilli, and A. Bolufé Röhler. Matheuristics: Optimization, Simulation and Control. In M. J. Blesa, C. Blum, L. Di Gaspero, A. Roli, M. Sampels, and A. Schaerf, editors, *Hybrid Metaheuristics*, Lecture Notes in Computer Science, pages 171–177. Springer Berlin Heidelberg, 2009.

V. Boyer, B. Gendron, and L.-M. Rousseau. A branch-and-price algorithm for the multi-activity multi-task shift scheduling problem. *Journal of Scheduling*, 17(2):185–197, June 2013.

G. Brønmo, B. Nygreen, and J. Lysgaard. Column generation approaches to ship scheduling with flexible cargo sizes. *European Journal of Operational Research*, 200(1):139–150, Jan. 2010.

J. O. Brunner and J. F. Bard. Flexible weekly tour scheduling for postal service workers using a branch and price. *Journal of Scheduling*, 16(1):129–149, Feb. 2013.

M. J. Brusco and L. W. Jacobs. A Simulated Annealing Approach to the Solution of Flexible Labour Scheduling Problems. *The Journal of the Operational Research Society*, 44(12): 1191–1200, 1993.

M. J. Brusco and L. W. Jacobs. Optimal Models for Meal-Break and Start-Time Flexibility in Continuous Tour Scheduling. *Management Science*, 46(12):1630–1641, 2000.

E. K. Burke, P. D. Causmaecker, S. Petrovic, and G. V. Berghe. Metaheuristics for handling time interval coverage constraints in nurse scheduling. *Applied Artificial Intelligence*, 20 (9):743–766, 2006.

V. Cacchiani, V. C. Hemmelmayr, and F. Tricoire. A set-covering based heuristic algorithm for the periodic vehicle routing problem. *Discrete Applied Mathematics*, 163(Part 1): 53–64, Jan. 2014.

M. Carrión and J. M. Arroyo. A computationally efficient mixed-integer linear formulation for the thermal unit commitment problem. *IEEE Transactions on Power Systems*, 21(3): 1371–1378, Aug. 2006.

S. Ceria, P. Nobili, and A. Sassano. A lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81(2):215–228, 1998a.

S. Ceria, P. Nobili, and A. Sassano. A Lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, 81(2):215–228, Apr. 1998b.

C. Cervilla, J. Villar, and F. A. Campos. Bi-level optimization of electricity tariffs and PV distributed generation investments. In *2015 12th International Conference on the European Energy Market (EEM)*, pages 1–5, May 2015.

M. Chiarandini, A. Schaerf, and F. Tiozzo. Solving employee timetabling problems with flexible workload using tabu search. In *Proceedings of the 3 rd Int. Conf. on the Practice and Theory of Automated Timetabling (E. Burke & W. Erben, Eds.) pp*, pages 298–302. Citeseer, 2000.

M.-C. Côté, B. Gendron, C.-G. Quimper, and L.-M. Rousseau. Formal languages for integer programming modeling of shift scheduling problems. *Constraints*, 16(1):54–76, Jan. 2011a.

M.-C. Côté, B. Gendron, and L.-M. Rousseau. Grammar-Based Integer Programming Models for Multiactivity Shift Scheduling. *Management Science*, 57(1):151–163, 2011b.

M.-C. Côté, B. Gendron, and L.-M. Rousseau. Grammar-Based Column Generation for Personalized Multi-Activity Shift Scheduling. *INFORMS Journal on Computing*, 25(3): 461–474, July 2013.

S. Dahmen and M. Rekik. Solving Multi-activity Multi-day Shift Scheduling Problems with a Hybrid Heuristic. *J. of Scheduling*, 18(2):207–223, Apr. 2015.

S. Dahmen, M. Rekik, and F. Soumis. An implicit model for multi-activity shift scheduling problems. *Journal of Scheduling*, 21(3):285–304, June 2018.

G. B. Dantzig. Letter to the EditorA Comment on Edie's "Traffic Delays at Toll Booths". *Journal of the Operations Research Society of America*, 2(3):339–341, Aug. 1954.

G. B. Dantzig and P. Wolfe. Decomposition Principle for Linear Programs. *Operations Research*, 8(1):101–111, 1960.

K. Darby-Dowman and G. Mitra. An extension of set partitioning with application to scheduling problems. *European Journal of Operational Research*, 21(2):200–205, Aug. 1985.

P. De Bruecker, J. Van den Bergh, J. Belien, and E. Demeulemeester. A tabu search heuristic for building aircraft maintenance personnel rosters. *Available at SSRN 2464033*, 2014.

V. L. de Matos, D. P. Morton, and E. C. Finardi. Assessing policy quality in a multi-stage stochastic program for long-term hydrothermal scheduling. *Annals of Operations Research*, 253(2):713–731, June 2017.

S. Demassey, G. Pesant, and L.-M. Rousseau. Constraint Programming Based Column Generation for Employee Timetabling. pages 140–154. Springer, Berlin, Heidelberg, May 2005.

S. Demassey, G. Pesant, and L.-M. Rousseau. A Cost-Regular Based Hybrid Column Generation Approach. *Constraints*, 11(4):315–333, Oct. 2006.

S. Dempe and J. Dutta. Is bilevel programming a special case of a mathematical program with complementarity constraints? *Mathematical Programming*, 131(1):37–48, Feb. 2012.

S. Dempe, V. Kalashnikov, G. A. Prez-Valds, and N. Kalashnykova. *Bilevel Programming Problems: Theory, Algorithms and Applications to Energy Networks*. Energy Systems. Springer-Verlag, Berlin Heidelberg, 2015.

G. Desaulniers, J. Desrosiers, Y. Dumas, S. Marc, B. Rioux, M. M. Solomon, and F. Soumis. Crew pairing at Air France. *European Journal of Operational Research*, 97(2):245–259, Mar. 1997.

G. Desaulniers, J. Desrosiers, and M. M. Solomon. Accelerating Strategies in Column Generation Methods for Vehicle Routing and Crew Scheduling Problems. In C. C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, Operations Research/Computer Science Interfaces Series, pages 309–324. Springer US, Boston, MA, 2002.

J. Desrosiers and M. E. Lübbecke. A Primer in Column Generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 1–32. Springer US, Boston, MA, 2005.

F. F. Easton and D. F. Rossin. Sufficient Working Subsets for the Tour Scheduling Problem. *Management Science*, 37(11):1441–1451, 1991.

L. C. Edie. Traffic Delays at Toll Booths. *Journal of the Operations Research Society of America*, 2(2):107–138, 1954.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier. An Annotated Bibliography of Personnel Scheduling and Rostering. *Annals of Operations Research*, 127(1-4): 21–144, Mar. 2004a.

A. T. Ernst, H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: A review of applications, methods and models. *European Journal of Operational Research*, 153(1):3–27, Feb. 2004b.

F. Facchinei and J.-S. Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems*. Springer Series in Operations Research and Financial Engineering, Finite-Dimensional Variational Inequalities and Complementarity Problems: Volume I. Springer-Verlag, New York, 2003a.

F. Facchinei and J.-S. Pang. *Finite-Dimensional Variational Inequalities and Complementarity Problems*. Springer Series in Operations Research and Financial Engineering, Finite-Dimensional Variational Inequalities and Complementarity Problems: Volume II. Springer-Verlag, New York, 2003b.

R. Fernández-Blanco, J. M. Arroyo, N. Alguacil, and X. Guan. Incorporating Price-Responsive Demand in Energy Scheduling Based on Consumer Payment Minimization. *IEEE Transactions on Smart Grid*, 7(2):817–826, Mar. 2016.

M. Fischetti, I. Ljubić, M. Monaci, and M. Sinnl. Intersection Cuts for Bilevel Optimization. In Q. Louveaux and M. Skutella, editors, *Integer Programming and Combinatorial Optimization*, pages 77–88. Springer International Publishing, 2016.

M. L. Fisher and P. Kedia. Optimal Solution of Set Covering/Partitioning Problems Using Dual Heuristics. *Management Science*, 36(6):674–688, June 1990.

F. Furini, E. Malaguti, S. Martin, and I.-C. Ternier. ILP Models and Column Generation for the Minimum Sum Coloring Problem. *Electronic Notes in Discrete Mathematics*, 64: 215–224, Feb. 2018.

M. Gamache, F. Soumis, G. Marquis, and J. Desrosiers. A Column Generation Approach for Large-Scale Aircrew Rostering Problems. *Operations Research*, 47(2):247–263, 1999.

L. D. Gaspero, J. Grtner, N. Musliu, A. Schaerf, W. Schafhauser, and W. Slany. A Hybrid LS-CP Solver for the Shifts and Breaks Design Problem. In *Hybrid Metaheuristics*, pages 46–61. Springer, Berlin, Heidelberg, Oct. 2010.

L. D. Gaspero, J. Grtner, N. Musliu, A. Schaerf, W. Schafhauser, and W. Slany. Automated Shift Design and Break Scheduling. In *Automated Scheduling and Planning*, Studies in Computational Intelligence, pages 109–127. Springer, Berlin, Heidelberg, 2013.

M. Gérard, F. Clautiaux, and R. Sadykov. Column generation based approaches for a tour scheduling problem with a multi-skill heterogeneous workforce. *European Journal of Operational Research*, 252(3):1019–1030, Aug. 2016.

F. Glover. Tabu search-part i. *ORSA Journal on computing*, 1(3):190–206, 1989.

S. Hao and F. Zhuang. New models for integrated short-term forward electricity markets. *IEEE Transactions on Power Systems*, 18(2):478–485, May 2003.

A. Haurie, R. Loulou, and G. Savard. A two-player game model of power cogeneration in New England. *IEEE Transactions on Automatic Control*, 37(9):1451–1456, Sept. 1992.

S. Held, W. Cook, and E. C. Sewell. Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, 4(4):363–381, Dec. 2012.

R. Henrion, J. Outrata, and T. Surowiec. Analysis of M-stationary points to an EPEC modeling oligopolistic competition in an electricity spot market. *ESAIM: Control, Optimisation and Calculus of Variations*, 18(2):295–317, Apr. 2012.

N. A. Hernández-Leandro, V. Boyer, M. A. Salazar-Aguilar, and L.-M. Rousseau. A matheuristic based on Lagrangian relaxation for the multi-activity shift scheduling problem. *European Journal of Operational Research*, July 2018.

B. Heymann and A. Jofré. Mechanism Design and Auctions for Electricity Network. May 2016.

B. F. Hobbs and S. K. Nelson. A nonlinear bilevel model for analysis of electric utility demand-side planning issues. *Annals of Operations Research*, 34(1):255–274, Dec. 1992.

J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, And Computation*. Addison-Wesley, Boston, 3rd edizione edition, 2006.

D. Huisman, R. Jans, M. Peeters, and A. P. M. Wagelmans. Combining Column Generation and Lagrangian Relaxation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 247–270. Springer US, 2005.

S. Irnich and G. Desaulniers. Shortest Path Problems with Resource Constraints. In *Column Generation*, pages 33–65. Springer, Boston, MA, 2005.

L. W. Jacobs and M. J. Brusco. Overlapping Start-Time Bands in Implicit Tour Scheduling. *Management Science*, 42(9):1247–1259, Sept. 1996.

Y. Jin, J.-P. Hamiez, and J.-K. Hao. Algorithms for the minimum sum coloring problem: a review. *Artificial Intelligence Review*, 47(3):367–394, Mar. 2017.

E. G. Kardakos, C. K. Simoglou, and A. G. Bakirtzis. Optimal Offering Strategy of a Virtual Power Plant: A Stochastic Bi-Level Approach. *IEEE Transactions on Smart Grid*, 7(2): 794–806, Mar. 2016.

L. Lozano and A. L. Medaglia. On an exact method for the constrained shortest path problem. *Computers & Operations Research*, 40(1):378–384, Jan. 2013.

M. E. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, 53(6):1007–1023, 2005.

R. M. Lusby, J. Larsen, M. Ehrgott, and D. Ryan. Railway track allocation: models and methods. *OR Spectrum*, 33(4):843–883, Oct. 2011.

E. Malaguti, M. Monaci, and P. Toth. An exact approach for the Vertex Coloring Problem. *Discrete Optimization*, 8(2):174–190, May 2011.

A. Mehrotra, K. E. Murphy, and M. A. Trick. Optimal shift scheduling: A branch-and-price approach. *Naval Research Logistics (NRL)*, 47(3):185–200, Apr. 2000.

A. Meisels and A. Schaerf. Modelling and Solving Employee Timetabling Problems. *Annals of Mathematics and Artificial Intelligence*, 39(1-2):41–59, Sept. 2003.

A. Mingozzi, V. Maniezzo, S. Ricciardelli, and L. Bianco. An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation. *Management Science*, 44(5):714–729, May 1998.

A. Mohsenian-Rad, V. W. S. Wong, J. Jatskevich, R. Schober, and A. Leon-Garcia. Autonomous Demand-Side Management Based on Game-Theoretic Energy Consumption Scheduling for the Future Smart Grid. *IEEE Transactions on Smart Grid*, 1(3):320–331, Dec. 2010.

J. A. Momoh and L. Mili. *Economic Market Design and Planning for Electric Power Systems.* Wiley, Nov. 2009.

J. T. Moore and J. F. Bard. The Mixed Integer Linear Bilevel Programming Problem. *Operations Research*, 38(5):911–921, Oct. 1990.

N. Musliu, A. Schaerf, and W. Slany. Local search for shift design. *European Journal of Operational Research*, 153(1):51–64, Feb. 2004.

İ. Muter, Ş. İ. Birbil, and G. Şahin. Combination of Metaheuristic and Exact Algorithms for Solving Set Covering-Type Optimization Problems. *INFORMS Journal on Computing*, 22(4):603–619, Mar. 2010.

P. J. Neame. *Nonsmooth Dual Methods in Integer Programming.* PhD thesis, University of Melbourne, Department of Mathematics and Statistics, 1999.

J. V. Outrata. On the numerical solution of a class of Stackelberg problems. *Z. Operations Research*, 34(4):255–277, July 1990.

J.-S. Pang and M. Fukushima. Quasi-variational inequalities, generalized Nash equilibria, and multi-leader-follower games. *Computational Management Science*, 2(1):21–56, Jan. 2005.

J.-S. Pang and M. Fukushima. Quasi-variational inequalities, generalized Nash equilibria, and multi-leader-follower games. *Computational Management Science*, 6(3):373–375, Aug. 2009.

J.-S. Pang, M. Razaviyayn, and A. Alvarado. Computing B-Stationary Points of Nonsmooth DC Programs. *Mathematics of Operations Research*, 42(1):95–118, Oct. 2016.

G. Pesant. A Regular Language Membership Constraint for Finite Sequences of Variables. pages 482–495. Springer, Berlin, Heidelberg, Sept. 2004.

A. Pessoa, R. Sadykov, E. Uchoa, and F. Vanderbeck. Automation and combination of linear-programming based stabilization techniques in column generation, May 2017.

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, Dec. 2009.

C.-G. Quimper and L.-M. Rousseau. A large neighbourhood search approach to the multi-activity shift scheduling problem. *Journal of Heuristics*, 16(3):373–392, Apr. 2009.

M. S. Rasmussen and J. Larsen. *Optimisation-Based Solution Methods for Set Partitioning Models.* PhD thesis, Technical University of Denmark, Department of Informatics and Mathematical Modeling, 2011.

M. Rekik, J.-F. Cordeau, and F. Soumis. Implicit shift scheduling with multiple breaks and work stretch duration restrictions. *Journal of Scheduling*, 13(1):49–75, Feb. 2010.

M. I. Restrepo, L. Lozano, and A. L. Medaglia. Constrained network-based column generation for the multi-activity shift scheduling problem. *International Journal of Production Economics*, 140(1):466–472, Nov. 2012.

M. I. Restrepo, B. Gendron, and L.-M. Rousseau. Branch-and-Price for Personalized Multiactivity Tour Scheduling. *INFORMS Journal on Computing*, 28(2):334–350, Apr. 2016.

M. I. Restrepo, B. Gendron, and L.-M. Rousseau. Combining Benders decomposition and column generation for multi-activity tour scheduling. *Computers & Operations Research*, 93:151–165, May 2018.

N. J. Rezanova and D. M. Ryan. The train driver recovery problemA set partitioning based model and solution method. *Computers & Operations Research*, 37(5):845–856, May 2010.

G. Righini and M. Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, Sept. 2006.

R. T. Rockafellar. Solving Stochastic Programming Problems with Risk Measures by Progressive Hedging. *Set-Valued and Variational Analysis*, July 2017.

R. T. Rockafellar and S. Uryasev. Optimization of Conditional Value-at-Risk. *Journal of Risk*, 2:21–41, 2000.

R. T. Rockafellar and S. Uryasev. The fundamental risk quadrangle in risk management, optimization and statistical estimation. *Surveys in Operations Research and Management Science*, 18(1):33–53, Oct. 2013.

M. Rönnqvist. A method for the cutting stock problem with different qualities. *European Journal of Operational Research*, 83(1):57–68, May 1995.

S. Ropke and D. Pisinger. An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows. *Transportation Science*, 40(4):455–472, Nov. 2006.

F. Rossi and S. Smriglio. A set packing model for the ground holding problem in congested networks. *European Journal of Operational Research*, 131(2):400–416, June 2001.

R. Sadykov, F. Vanderbeck, A. Pessoa, I. Tahiri, and E. Uchoa. Primal Heuristics for Branch-and-Price: the assets of diving methods. *INFORMS Journal on Computing*, Jan. 2018.

A. Shapiro. On complexity of multistage stochastic programs. *Operations Research Letters*, 34(1):1–8, Jan. 2006.

P. Shaw. Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems. pages 417–431. Springer, Berlin, Heidelberg, Oct. 1998.

T. Surowiec. *Explicit stationarity conditions and solution characterization for equilibrium problems with equilibrium constraints.* PhD thesis, Humboldt-Universität zu Berlin, Mar. 2010.

M. Tahanan, W. van Ackooij, F. Antonio, and F. Lacalandra. Large-scale Unit Commitment under uncertainty: a literature survey. *4OR*, 13:115–171, Jan. 2015.

R. Taktak and C. D'Ambrosio. An overview on mathematical programming approaches for the deterministic unit commitment problem in hydro valleys. *Energy Systems*, 8(1):57–79, Feb. 2017.

G. M. Thompson. Improved Implicit Optimal Modeling of the Labor Shift Scheduling Problem. *Management Science*, 41(4):595–607, Apr. 1995.

G. M. Thompson. Labor scheduling: A commentary. *The Cornell Hotel and Restaurant Administration Quarterly*, 44(5):149–155, Oct. 2003.

N. Touati-Moungla, L. Létocart, and A. Nagih. Solutions diversification in a column generation algorithm. *Algorithmic Operations Research*, 5(2):86–95, Dec. 2010.

W. van Ackooij and W. de Oliveira. DC programming techniques with inexact subproblems' solution for general DC programs. *Submitted manuscript*, pages 1–27, 2017.

J. Van den Bergh, J. Belin, P. De Bruecker, E. Demeulemeester, and L. De Boeck. Personnel scheduling: A literature review. *European Journal of Operational Research*, 226(3):367–385, May 2013.

F. Vanderbeck. Implementing Mixed Integer Column Generation. In G. Desaulniers, J. Desrosiers, and M. M. Solomon, editors, *Column Generation*, pages 331–358. Springer US, Boston, MA, 2005.

F. Vanderbeck and L. A. Wolsey. An exact algorithm for IP column generation. *Operations Research Letters*, 19(4):151–159, Oct. 1996.

R. R. Vemuganti. Applications of Set Covering, Set Packing and Set Partitioning Models: A Survey. In D.-Z. Du and P. M. Pardalos, editors, *Handbook of Combinatorial Optimization*, pages 573–746. Springer US, Boston, MA, 1998.

P. J. Zwaneveld, L. G. Kroon, H. E. Romeijn, M. Salomon, S. Dauzère-Pérès, S. P. M. Van Hoesel, and H. W. Ambergen. Routing Trains Through Railway Stations: Model Formulation and Algorithms. *Transportation Science*, 30(3):181–194, Aug. 1996.

Acknowledgements

I will always be grateful to Roberto for his daily commitment and dedication as supervisor during these three years. Sei stato un padre scientifico essenziale per tutto il tempo.

My sincerest thanks go to François Clautiaux and Bernard Gendron for having agreed to be the referees of my thesis. I also want to thank Sophie Demassey, Safia Kedad-Sidhoum and Louis-Martin Rousseau who have accepted to be part of the jury despite their busy agenda. A special thank goes to Louis-Martin Rousseau, for all the constructive and valuable advices given during the internship at École Polytechnique de Montréal (il Paese dei Balocchi).

I want to thank my co-supervisor Lucas, and Nora and Mahuna for being not only my *co-encadrants*, but also good colleagues. Nora, from the moment you called me *Ania* by mistake, I realize that I could really count on you. Mahuna, from your being a dreamer I learned that I always have to try. I also thank my officemate Thomas for his good mood and all the breaks with the Arpeggio and Ristretto coffees. A special thank goes to Horizontal Software for giving me the opportunity of doing my PhD, and for pushing me to face new challenges and to go further and further under the best conditions.

I would like to thank the members of LIPN with whom I share many unforgettable moments. In particular, I am grateful to the *ritals*: Enrico for being the quietest flatmate at Scotti-Pan-Bettiol place; Emiliano for your insatiable curiosity and the fermented shark; Marcos for being so Brazilian and Italian at the same time; Michele Barbato for being my model as PhD student and for the wise words you told me about the feelings on the last months of PhD; Stefano, Farideh, Laura, Desirée and my officemates at LIPN (Antoine, Gaël, Rado, Tsinjo, Ivan and all the others).

A special thank goes to Anna, Giulia and Valentina: even though many kilometers and time zones divide us, you are always there. Aspettando la prossima Tennent's ai Navigli. I also thank my friends Lara, Elena and Letizia. Leti, nessuno mi ha capita più di te e Yoda quest'estate.

Ringrazio la mia famiglia per esserci sempre stata e per esserci sempre. Luciana per una vita passata ad assicurarti la nostra felicità, Romano per essere senza dubbio l'uomo della mia vita, Andrea perché avere un fratello maggiore mi fa sentire intoccabile, e Gaia perché finalmente ho anche una sorella. Ringrazio Simonetta e Roberto per farmi sentire sempre come una figlia.

And last but definitely not least, I thank Andrea for supporting and standing me during all these years. With your constant optimism, you taught me that you make your own luck (even though I still think yours is pretty special).