New methods for the multi-skills project scheduling problem

Carlos Eduardo Montoya Casas

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

ECOLE DES MINES DE NANTES

Thèse de Doctorat

Carlos Montoya

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'École des mines de Nantes
sous le label de l'Université de Nantes Angers Le Mans

Discipline : Informatique
Spécialité : Recherche Opérationnelle
Laboratoire : IRCCyN

Soutenue le 13 decembre 2012

École doctorale : 503 (STIM)
Thèse n° : 2012 EMNA 0077

New methods for the Multi-Skill Project Scheduling Problem

JURY

Rapporteurs :	**M. Christian ARTIGUES**, Directeur de Recherche, LAAS, CNRS, Toulouse **M. Jacques CARLIER**, Professeur, Université de Technologie de Compiègne
Examinateurs :	**M. Pierre DEJAX**, Professeur, Ecole des Mines de Nantes **M. Boris DETIENNE**, Maître de Conférences, Université d'Avignon **M. Emmanuel NÉRON**, Professeur, Université François-Rabelais de Tours
Directrice de thèse :	**M^{me} Christelle JUSSIEN-GUÉRET**, Professeur, Université de Angers
Co-encadrants :	**M^{me} Odile BELLENGUEZ-MORINEAU**, Chargée de Recherche, Ecole des Mines de Nantes **M. David RIVREAU**, Enseignant-Chercheur, Université Catholique de l'Ouest

Acknowledgements

I would like to express my thanks to all the people who have been very helpful to me during the whole phD experience. I have to outline also that the work done during my phD took place in the Ecole des Mines Nantes and the IRCCyN.

In the first place I would like to thank Christelle Jussien-Guéret who was the first person to consider me as a potential phD student. Thereafter, thanks to that initial support I was able to start this amazing journey, in which I had the opportunity to experience a new life experience in a different country and culture.

I would like also to thank my supervisors Odile Bellenguez-Morineau and David Rivreau for their valuable suggestions, feedback, patience, guidance and support all along this last three years. I would like to thank as well Eric Pinson, for his valuable help and contribution in the work done during my phD. Thanks is owed also to the project LigéRO for funding my phD thesis.

Subsequently, I want to thank Christian Artigues and Jacques Carlier for accepting to be the reviewers of my thesis and therefore for their remarks and suggestions allowing me to enforce certain perspectives that could lead to interesting findings in my research subject.

I want to thank as well Emmanuel Néron, Boris Detienne and Pierre Dejax for participating as members of the jury of my thesis and for their feedback and perspectives on the work done during my phD.

Finally I would like to thank all the good friends that I have the opportunity to met and cross by during this last three years, allowing me to have a lot of nice experiences and memories that I will always keep and treasure.

Contents

List of Tables

List of Figures

1

Introduction

Resource management plays an important role for enhancing the competitiveness of a company. Particularly, the correct management of human resources represents an important issue for organizations. Furthermore, defining the schedule and assignment of tasks among the available resources are important duties that take place in a normal daily basis in any organization. In the most of planning and scheduling tasks the hardest constraints are caused by restricted resources. This is why resource allocation is an important component of many real-life planning and scheduling tasks. From algorithmic point of view it can be considered either as a part of the planning or a part of the scheduling. A mixed approach is also used. Thereby, the methods used to perform such duties must be oriented in giving a good use to resources considering their capacity, cost and availability. There are several scheduling environments that deal with the aspects previously mentioned, thus, in this thesis we focus in a particular one, which involves the management of scarce resources in a Project Scheduling environment. Moreover, these issues are addressed by the Resource Constrained Project Scheduling Problem (RCPSP) [22] which is a classical scheduling problem that received major attention in the last years.

The RCPSP deals with a given number of activities that have to be scheduled on a set of resources. It also takes into account precedence relation between activities and limited resource availability. The interest in extending the practical applications of the RCPSP encouraged researchers to work on different extensions that capture several variants and features related to specific real life situations [22, 83, 93, 2, 64]. Different optimization criteria have been addressed for the RCPSP, from which minimization of the makespan is among the most popular ones. Although, other time-based objectives based on lateness, tardiness, and earliness present a particular importance as well. Additionally, there are other objectives based on costs assignment related to resources and/or activities and/or time. Also, the concept of cash flow has been taken into account for maximizing the net present value of a project [22].

In this context, it is important to notice that scheduling the activities of a project is partially conditioned by the constraints and specifications of the available resources. Subsequently, we can define such constraints and specifications by identifying the different types of resources that are normally considered [64] in a project:

- Renewable resources: Such a type of resources can be used whenever they are available in each period with its full capacity.(e.g staff members, machines and equipment).
- Non-renewable resources: This type of resources is available only in a given amount for the entire project duration (such as money). A good example would be a predefined budget for the project.
- Doubly constrained resources: They are limited both for each period and for the whole project. Money, can be an example if both the budget and the per-period cash flow of the project are limited.
- Partially renewable resources: They allow to define capacity limitations over arbitrary subsets of periods. This type of resources permits for example to represent workers for which their working hours are limited by the contract duration.

Furthermore, when dealing with human resources, we can also identify that particularly in service firms like health care enterprises, call centers and also in different manufacturing environments the utilization of staff members with multiple skills is commonly required. Thus, in order to consider such a feature, we also have to distinguish the resources that are able to fulfill a unique kind of requirement (mono-skilled) from the ones that are capable to satisfy several ones (multi-skilled) [65]. This concept can also be extended to a scheduling environment in which the resources are multi-purpose machines, that can be able to perform several types of tasks [23].

Thereafter, given its practical importance, the notion of skills has been addressed by several authors. More precisely, in the field of personnel scheduling and assignment there are several problems that deal with multi-skilled resources [45]. In this thesis we aim at considering the assignment of several resources with different skills to the same task (activity) under a Project Scheduling environment. More precisely, besides using multi-skilled resources, we consider that a given activity might have several skill requirements.

Furthermore, we focus on one particular extension of the RCPSP, which is known as the Multi-Skill Project Scheduling Problem (MSPSP). This problem was originally proposed by Néron and Baptista [93]. It mixes both the classical RCPSP and the Multi-Purpose Machine model. The aim is to find a schedule that minimizes the makespan. Practical applications can be related to call centers, construction of buildings, production and software development planning. A specific example of a real life application, with similar features to the MSPSP can be found in the work of Cordeau et al. [31]. They proposed a solution method for the technician and task scheduling problem arising in a large telecommunications company. Valls et al. [112] deal also with a real-life problem that comes up in the daily management of service centers.

In this work, we intend to propose several methods for solving the MSPSP. We give a particular importance to exact methods with the purpose of obtaining optimal solutions for small and medium sized instances. Best results obtained so far were achieved by Bellenguez-Morineau and Néron [17] by means of a heuristic approach. In addition, these last mentioned authors also founded strong lower bounds for the different available instances, obtaining that there are still several small and medium sized instances for which optimality is still to be proven.

This thesis is organized as follows: Chapter 2 presents the detailed description of the Multi-Skill Project Scheduling Problem, followed by an example based on a real life application. Afterwards, we propose a review of related problems for identifying and understanding differences and similarities. Thereafter, we introduce the previous work done on the MSPSP and other approaches for problems with similar features to the ones treated in this thesis. Hence, we explain and introduce all the instances that are used as benchmark for the MSPSP. More precisely, we introduce five different integer linear programming (ILP) models, which help us to represent the MSPSP from different perspectives. Thereby, we discuss their efficiency and limits, based on several computational experiments carried out on instances taken from the literature.

Subsequently, in chapter 3 we proposed different procedures for obtaining makespan lower bounds based on a Column Generation (CG) approach. Hence, we introduce several mathematical formulations and different resolution techniques. Thereby, we present the respective results, and then, we compare the obtained makespan lower bounds with the linear relaxations values of the ILP models.

Thereby, in chapter 4 we introduce both, exact and heuristic approaches for solving the MSPSP, based on the utilization of the Column Generation (CG). Therefore, initially, with the purpose of obtaining an integer optimal solution, we developed a Branch-and-Price (B&P) procedure. Subsequently, we describe the different branching and node filtering techniques, along with the methods we used for getting a makespan upper bound. Thereafter, we present the respective results and conclusions. Furthermore, with the purpose of solving big size instances, we introduce a recovering beam search (RBS) approach that exploits the structure of the branch and price procedures discussed in a previous section and integrates the resolution of different mathematical models. Hence, we introduce and analyze the obtained results.

Finally, in chapter 5 we explore a two-phase procedure, in which new constraints (cuts) are generated iteratively until ensuring an optimal schedule. The first phase involves the definition of the starting times of the activities of the project. Subsequently, in the second phase we focus on finding a feasible assignment of workers that allows the execution of the activities according to the starting times defined in the first phase. Thereby, we present and analyze the obtained results.

2

Problem definition and formulation

This chapter presents the detailed description of the Multi-Skill Project Scheduling Problem (MSPSP). All the related notations and specifications of the problem are explained, then, we present an example based on a real life application. We also describe the features of the expected solution after doing the schedule of the project presented in the example. Afterwards, we introduce the literature review of related problems in order to identify and understand differences and similarities. Thereby, we introduce the previous work done on the MSPSP and other approaches for solving problems that deal with the scheduling and assignment of multi-skilled resources.

2.1 Problem presentation

The Multi-Skill Project Scheduling Problem (MSPSP) is a known project scheduling problem, which is mainly composed by three elements: Activities, resources and skills. These elements are detailled below:

Activities

A project is composed by a set of activities $A = \{A_0, \ldots, A_N\}$. Within this set, we also define two dummy activities A_0 and A_N to represent respectively the beginning and termination of the project. Activities are submitted to precedence relations, which implies that certain activities have to be finished before others can start [64]. Thus, each activity A_i has a corresponding set of successors E_i^+ and predecessors E_i^-. This is handled by depicting the project as a directed graph (G) where an activity is represented by a node and the precedence relation between two activities is represented by a directed arc (A_i, A_j) where $A_j \in E_i^+$ [83]. This arc also represents the minimal duration time between the starting time of A_i and the beginning of any of its direct successors. Thereafter, such a duration time is denoted as p_i, which also corresponds to the processing time of A_i.

All the activities must be scheduled in order that the total duration time of the project (makespan) is minimized. Given that the starting time of an activity A_i is denoted by t_i, its completion time will be given by $t_i + p_i$, since preemption is not allowed.

Resources and skills

In a classical MSPSP context, the resources we focus on for performing all the activities of a project are staff members. Thus, we can ensure that we deal only with a renewable type of resources. Due to the nature of the treated problem, we consider a set W of M workers and a set S of K skills. Each single resource W_m ($W_m \in W$) is able to carry out a given subset of skills. The distribution of skills in the workforce is denoted by a parameter $MS_{m,k}$ which takes the value of one if worker W_m masters skill S_k, or zero otherwise.

Furthermore, a given number of workers must be assigned to each one of the required skills to perform a given activity. This implies that a single activity might need the utilization of several skills. The constraints linked to the notion of skills are described as follows [93]:

- A total number of $b_{i,k}$ workers that master each skill S_k must be assigned to activity A_i. If S_k is not required by A_i, we set $b_{i,k} = 0$.
- A worker W_m can be assigned only for using a skill he masters, i.e $MS_{m,k} = 1$.
- A worker W_m can only use one skill on one activity at a given time t.
- All the workers assigned to a specific activity A_i must work simultaneously during its whole processing time ($[t_i, t_i + p_i[$).

Thereby, we present all the features described above with an example, which is based on a real life application for the development of software products. Here we consider a small project of three activities, four workers and three skills. Table 2.1 shows the description of each skill. Table 2.2 gives the skills requirement of each activity. Table 2.3 shows the description of the staff members in terms of the skills. Figure 2.1 presents the graph of the project with the processing times of the activities. Figure 2.2 shows a Gantt chart of a feasible solution that fulfills all the constraints.

S_0	Programmer
S_1	Data Base Designer
S_2	Webmaster

Table 2.1: Required skills description

From this table it can be noticed that W_0 masters S_1 and S_2, while W_2 only masters S_0.

This table shows for instance that A_3 requires two workers that masters S_0 and one worker S_1, while A_4 only requires one worker that masters S_1.

	S_0	S_1	S_2
A_1	-	1	1
A_2	1	-	1
A_3	2	1	-
A_4	-	1	-

Table 2.2: Number of workers required per activity ($b_{i,k}$) per skill

	S_0	S_1	S_2
W_0	-	1	1
W_1	1	-	1
W_2	1	-	-
W_3	1	-	1

Table 2.3: List of skills mastered per worker ($MS_{m,k}$)

From this table we can see that W_0 masters S_1 and S_2, while W_2 only masters S_0.

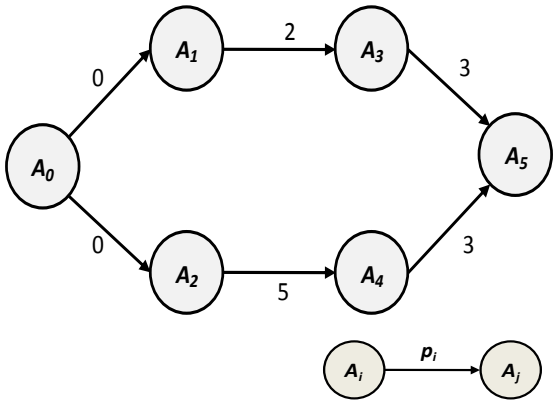

Figure 2.1: Precedence relations Graph G

Notice that A_0 and A_5 are additional dummy activities which represent the beginning and termination of the project respectively. Arcs weight represents the processing times of each activity.

It is important to notice from figure 2.2, that once A_1 and A_2 are scheduled at time zero, A_3 cannot start at time two, due to the fact that at that moment there are not enough available workers that masters S_0 to fulfill all the requirements of such an activity. In addition, the fact that A_3 is scheduled at time five implies that W_0, who is the only one that masters S_1, will not be available until time eight. Thus A_4 can start only until that moment.

Finally, according to the known fact that the RCPSP is already an $\mathcal{NP}$-Hard optimization problem, thus the MSPSP is also $\mathcal{NP}$-Hard in the strong sense [2]. Thereby, these two problems are equivalent if each worker master only one skill.

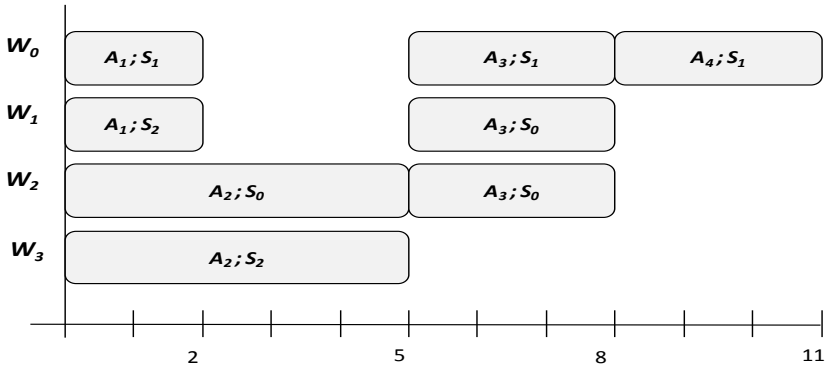

Figure 2.2: Gantt chart of an optimal solution

2.2 Related Problems

In this section, we try to put in context the features of our problem in comparison to other related problems of the literature. Afterwards, we discuss some of the previous work done specifically on the MSPSP.

2.3 Job Shop Scheduling Problem

The Resource constrained Project Scheduling Problems are considered as a generalization of the JSSP, thus we start by giving an overview of its most important features. The Job Shop Scheduling Problem (JSSP) is a well-known disjunctive scheduling problem which can be described as follows:

- The problem consists of a set J of n jobs and set M of m machines.
- Each job $j \in J$ must be processed exactly once on each machine.
- Each job defines an order in which it should be processed on the machines. For a job $j \in J$ let $j(k)$, $\pi_j(k) 1 \leq k \leq m$, denote the kth machine for j.
- The processing of job $j \in J$ on machine $\pi_j(k)$ is called an operation, and denoted $o_{j,k}$. An operation $o_{j,k}$ has an duration of $p_{j,k}$ and cannot be preempted, i.e., the processing of an operation cannot be interrupted and then resumed at a later point in time.
- Each operation of a job must occur one after the other.
- Only one operation may be in progress at a time on each machine.
- The objective is to minimize the project makespan, i.e., the total time needed to complete all operations.
- The JSSP is $\mathcal{NP}$-Hard for $|M| \geq 2$ and $|J| \geq 3$ (see Garey et al. [54]).

The JSSP is from a modeling point of view very simple, yet general enough to model many real-life situations, such as a factory floor, where there is no choice of which machine to be used for a given operation, and each machine can only perform a single operation at a time.

It is important to mention that there are other extensions of the JSSP that share some similarities with Resource constrained Project Scheduling Problems. Particularly, we outline the FJSSP, which deals with the assignment of several machines that can be assigned to the same operation (mono-skilled resources). More precisely, another extension of the JSSP suitable with some of the features of the MSPSP is the Multiprocessor Job Shop Scheduling [75], which considers that each operation may require the utilization of several resources simultaneously.

2.4 The Cumulative Scheduling Problem

Another scheduling problem encountered in the literature is the Cumulative Scheduling Problem (CuSP) which is $\mathcal{NP}$-Hard ([6]). This problem introduces the concept of cumulative resources, which is an important resource feature considered in the Resource Constrained Project Scheduling Problems. The CuSP can be described as follows:

- A set A of activities must be performed on a single resource which has a constant capacity R.
- Each activity $A_i \in A$ has a processing time p_i (non-preemptive), and requires b_i units of the resource while it is in progress.
- Each activity $A_i \in A$, is associated with a release time t_i^r and due date t_i^d, and the activity must be performed in the interval $[t_i^r, t_i^d]$. Let T be a set of time steps.
- Usually, the aim is to find a feasible schedule, which minimizes the makespan.

The notion of cumulative resources plays an important role in Resource Constrained Project Scheduling Problems. It allows us to identify different features of the resources used in a given Project Scheduling environment.

2.5 Resource Constrained Project Scheduling Problem

As we discussed before, the MSPSP is considered as an extension of the RCPSP. Hence, we give a little overview of the main features of such a problem [22]. For keeping the similarity with the MSPSP, next description only considers renewable resources. Nevertheless, notice that there are other types of resource that can be considered such as: non-renewable resources, doubly constrained resources and partially renewable resources.

A project consists of a set of activities $A = \{A_0, \ldots, A_N\}$ and a set R of resources, where activities A_0 and A_N are dummy activities representing the start and completion of the project, respectively. Each activity A_i has a processing time of p_i, and during its non-preemptive processing it requires $b_{i,k}$ units of resource $R_k \in R$. Each resource $R_k \in R$, has capacity B_k in each time period. Activities must follow a precedence relation, which implies that certain activities have to be finished before others can start [64]. Thus, each activity A_i has a corresponding set of successors E_i^+.

Due to the $\mathcal{NP}$-Hardness of these project scheduling problem exact methods fail on large instances [2], but there are many constructive heuristics and local search techniques available to find good quality solutions. For an overview of these solution approaches the reader is referred to [83].

An importance thing to notice, is that despites that the scheduling of the RCPSP is entirely defined by the starting times of the activities, for the MSPSP it is mandatory to determine the assignment of workers according to their skills.

2.6 Multi-Mode Resource Constrained Project Scheduling Problem

Another interesting generalization of the RCPSP is the Multi-Mode Resource Constrained Project Scheduling Problem, which is also an $\mathcal{NP}$-Hard problem. Thereafter, we define it as follows [22]:

- A project consists of a set of activities $A = \{A_0, \ldots, A_N\}$ and a set R of resources, where activities A_0 and A_N are dummy activities representing the start and completion of the project, respectively.
- Each activity A_i can be performed in a number of different modes $M_i = 1, ..., |Mj|$, each representing an alternative way of performing the activity.
- Each resource $R_k \in R$, has capacity B_k in each time period.
- When an activity A_i is scheduled in mode $m \in M_i$, it has a processing time of $p_{i,m}$ (non-preemptive) and requires $b_{i,k,m}$ units of resource $R_k \in R$ in each time period.
- Activities must follow a precedence relation, which implies that certain activities have to be finished before others can start [64]. Thus, each activity A_i has a corresponding set of successors E_i^+.

For keeping similarity with the features of our studied problem, in the previous description we considered only renewable resources. There are several approaches that have been used for solving this problem, thus the reader is referred to [64].

The MSPSP can be seen as a Multi-Mode RCPSP [2]. In the context our problem, the Multi-Mode approach would require representing all the feasible combinations of workers to execute an activity [93].

For illustrate the previous statement, let us remember the described example for the MSPSP (see section 2.1).If we want to define the modes of execution of A_2, we would have to consider seven modes to perform such an activity, given by the next combination of workers: $(W_0, W_1); (W_0, W_2); (W_0, W_3); (W_1, W_2); (W_1, W_3); (W_2, W_3)$. Each combination could represent more than one mode according to how skills are distributed among the respective couple of workers. For example in the combination (W_1, W_3) both workers masters the two skills required by the mentioned activity (S_0 and S_2). Thus, in

this case there are two modes of skills assignment between both workers: (i) W_1 uses S_0 and W_3 uses S_2 or (ii) W_1 uses S_2 and W_3 uses S_0. In such situation, both modes implies that during the execution of A_2 the two workers will not be available.

If we consider bigger projects the number of modes could be really huge, making impossible to implement the existing methods related to the resolution of the Multi-mode RCPSP [2].

2.7 Multi-purpose machine model

Now, for including the notion of multi-skilled resources in the RCPSP, we introduce The Multi-purpose machine model [23], which also has a $\mathcal{NP}$-Hard complexity.

The general features of this problem are described as follows:

- There are m different machines, distinguish from M_1 to M_m.
- There are N jobs that must be scheduled.
- Each job j must be executed by a machine selected from a subset of machines that are able to carry it out during a constant processing time p_j.
- Each machine is able to perform a subset of jobs.

This problem allows us to put in context the notion of resources that are able to perform different operations (multi-skilled) as it occurs with the MSPSP.

The Multi-purpose machine model differs from the MSPSP in the fact that jobs have an unitary requirement, thus the related solution methods cannot be directly applied for solving the MSPSP, considering that in our problem we have to deal with the synchronization of certain resources for executing a single activity.

2.8 Multi-skilled personnel assignment background

After introducing different problems with similar features to the ones of the MSPSP, we established the importance of multi-skilled resources for determining the difference with other RCPSP problems. Thus, we aim on identifying how the notion of skills have been considered in personnel scheduling and assignment problems that deal particularly with multi-skilled resources.

Different papers related to scheduling and personnel assignment considers the notion of skills. On one hand, we have several problems that deal with multi-skilled resources, but only one resource can be assigned to a task. For example, the Home Health Care problem (HHC) considers the utilization of nurses with several qualifications or skills that has to be assigned into a set of jobs. Usually in this type of problem a single nurse is assigned to a job, but such nurse must have the set of skills required for the job. Related to this matter, Begur et al. [13] proposed a spatial decision support system for scheduling and routing

home-health-care nurses, while Bertels and Fahle [18] proposed a combination of linear programming, constraint programming, and meta-heuristics. Another interesting problem with similar features is the Skilled Workforce Project Scheduling Problem (SWPSP), for which Valls et al. [112] proposed a hybrid genetic algorithm that combines local searches with genetic population management techniques to solve it.

On the other hand, some problems consider the assignment of several workers with different skills to the same task. For instance, Eveborn et al. [47] worked on an extension to the HHC problem and proposed a set partitioning model, and as a solution method they developed a repeated matching algorithm. More recently, Cordeau et al. [31] developed a construction heuristic and an adaptive large neighborhood search heuristic for the Technician and Task Scheduling Problem (TTSP) in a large telecommunications company. Another problem with similar features, which was solved using column generation [44] is the Manpower Allocation Problem with time-windows, job-teaming constraints and a limited number of teams (m-MAPTWTC). This problem deals with the assignment of a set of teams into a set of tasks, restricted by time-windows. Additional assistance between teams might be required to perform a task, thus all cooperating teams must initiate execution simultaneously. The goal of this approach is to maximize the total number of assigned tasks. Li and Rodrigues [85] proposed construction heuristics used with simulated annealing to solve also the MAPTWTC.

Furthermore, the Synchronized Vehicle Dispatching problem (SVDP) as presented by Rousseau et al. [104] is a dynamic vehicle routing problem similar to the MSPSP. In SVDP, the visits of the vehicles may require additional assistance from other vehicles or special teams, thus vehicles and the special teams have to be synchronized. They proposed a constraint programming based greedy procedure with post-optimization using local search. Another vehicle routing extension is the Vehicle Routing Problem with Split Deliveries with time-windows (VRPTWSD), which allows a customer to be visited by several vehicles, each fulfilling some of the demand. For example Ho and Haugland [68] proposed a Tabu Search approach to solve such problem.

2.9 Multi-skilled resources in Project Scheduling background

Despite the fact that the notion of skills plays an important role in the field of personnel assignment [77], it is not often considered in the project scheduling field. Thus, regarding the Multi-Skill Project Scheduling Problem, it can be outlined the work done by Bellenguez-Morineau and Néron [15, 16, 17], who developed and implemented different procedures to determine lower and upper bounds for the makespan. For instance, Cordeau et al. [31] developed a construction heuristic and an adaptive large neighborhood search heuristic for the Technician and Task Scheduling Problem (TTSP) in a large telecommunications company. For solving also this last mentioned problem, more recently, Firat and Hurkens [50] developed a solution methodology that uses a flexible matching model as a core engine based on a underlying mixed integer programming model.

Additionally, we would like to mention that there are interesting methodologies in the literature of project scheduling with multi-skilled human resources. For example, Heimerl and Kolisch [65] proposed a mixed integer linear program to solve a multi-project problem where the schedule of each project is fixed. They also considered, multi-skilled human workforce with heterogeneous and static efficiencies. Li and Womer [85] developed a hybrid algorithm based on mixed integer modeling and constraint programming for solving a project scheduling problem with multi-skilled personnel, taking into consideration an individual workload capacity for each worker. In this approach a worker may be able to perform multiple skills, but only one at a time. Gutjahr et al. [63] proposed a greedy heuristic and a hybrid method using priority-based rules, ant colony optimization and genetic algorithm to solve the so-called “Project Selection, Scheduling and Staffing with Learning Problem". More recently, Correia et al. [33] presented a mixed-integer linear programming formulation and several sets of additional inequalities for a variant of the resource-constrained project scheduling problem in which resources are flexible, i.e., each resource has several skills.

2.10 Selected instances

The available MSPSP instances were generated by Bellenguez-Morineau [15] and are based on precedence graphs already studied in the project scheduling domain. More precisely, a part of the available instances are based on graphs taken from the PSPlib [82, 84]. These graphs have a density indicator named “Network Complexity", which rates between 1,5 and 2,1. Such an indicator covers the average number of successors per activity, which is the parameter used for generating different graphs in the PSPlib. Thus, a part of the studied MSPSP instances were build considering representative instances from the previous mentioned library, taking into account different values of the ‘Network Complexity". Nevertheless, it is important to notice that such an indicator doesn’t allow to entirely determine the structure of the graph.

Furthermore, other instances were generated considering graphs from Baptiste et al. [5], which have a lower density indicator. Additionally, RCPSP instances proposed by Patterson et al. [100] and Néron [93] were also considered, given the fact that they were generated based on real project scheduling problems.

The available MSPSP instances, are divided in three groups:

– Group 1: Considers the precedence graphs proposed by Baptiste et al. [5], Patterson et al. [100] and Néron [94]. There are 185 instances from this group, with a number of activities that ranges between 8 and 50. The number of skills was randomly generated between 3 and 8. The number of workers varies between 5 and 22.
– Group 2: Considers 174 instances based on precedence graphs created for the classical RCPSP and taken from the PSPlib [82] and from Kolisch and Sprecher [84]. Regarding this group of instances, generated graphs consider 30, 60 and 90 activities. The number of skills was randomly generated between 9 and 18. The number of workers oscillates between 5 and 30.

– Group 3: Considers the precedence graphs created for the Multi-mode RCPSP and taken from the PSPlib [82] and from Kolisch and Sprecher [84]. There are 198 instances that correspond to this group, which considers 12, 14, 16, 18, 20, 22, 32, 62 and 92 activities. The number of skills is fixed between 3 and 12. The number of workers varies between 4 and 15.

It is important to mention that in the work done by Bellenguez [15], it is also outlined that there is not an specific criteria that measures the level of difficulty of each of the three groups of studied instances.

2.11 Integer linear programming models

In this section we introduce five different integer linear programming (ILP) models, which help us to represent the MSPSP from different perspectives. Therefore, we discuss their efficiency and limits, based on several computational experiments carried out on instances taken from the literature. It is important to mention that the time horizon (T) was set equal to an upper bound computed with the Tabu Search procedure developed by Bellenguez-Morineau and Néron [17].

2.11.1 Time indexed model (TIM)

First, we present a time indexed model proposed by Bellenguez-Morineau and Néron [16]. This ILP follows the time indexed notion considered by Pritsker et al. [102] for solving the RCPSP. Originally, this model considers a binary decision variable that takes the value of one if a given activity starts at given time point, or takes the value of zero otherwise. In addition, resource conflicts are avoided by formulating a linear constraint for each time-point. The reformulation proposed by Bellenguez-Morineau and Néron [16] extends this time indexed notion in the MSPSP context, in which it is important to identify which worker is assigned to a given activity and which skill he uses to perform it. Therefore, the decision variables governing the target model are defined by:

Variables

$x_{i,m}^{t}$ 1 if worker W_m starts activity A_i at time t , 0 otherwise;
$y_{i,m}^{k}$ 1 if worker W_m uses skill S_k to performs activity A_i, 0 otherwise;

Model Formulation

Now, to facilitate the comprehension of the model, we fix the starting time of an activity A_i as:

$$t_i = \frac{\sum_{m \in W} \sum_{t \in [0,T]} (x_{i,m}^t \cdot t)}{\sum_{k \in S} b_i^k} \quad \forall i \in A \tag{2.1}$$

Hence, the associated mathematical formulation can be stated as:

$$Z[TIM] : Min\ C_{max} = t_N \tag{2.2}$$

$S.t.$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E_i^+ \tag{2.3}$$

$$es_i \leq t_i \leq ls_i \quad \forall i\ A \tag{2.4}$$

$$\sum_{t \in [0,T]} x_{i,m}^t \leq 1 \quad \forall i \in A, \forall m \in W \tag{2.5}$$

$$\sum_{i \in A} \sum_{d \in [t-p_i+1,t]} x_{i,m}^d \leq 1 \quad \forall t \in [0,T]\, \forall m \in W \tag{2.6}$$

$$\sum_{t \in [0,T]} (x_{i,m}^t \cdot t) \leq \frac{\sum_{h \in W} \sum_{t \in [0,T]} (x_{i,h}^t \cdot t)}{\sum_{k \in S} b_i^k} \quad \forall i \in A, \forall m \in W \tag{2.7}$$

$$y_{i,m}^k \leq M S_m^k \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.8}$$

$$\sum_{m \in W} y_{i,m}^k = b_i^k \quad \forall i \in A, \forall k \in S \tag{2.9}$$

$$\sum_{t \in [0,T]} x_{i,m}^t = \sum_{k \in S} y_{i,m}^k \quad \forall i \in A, \forall m \in W \tag{2.10}$$

$$x_{i,m}^t \in \{0,1\} \quad \forall i \in A, \forall m \in W, \forall t \in [0,T] \tag{2.11}$$

$$y_{i,m}^k \in \{0,1\} \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.12}$$

The objective (2.2) is to minimize the completion time (makespan) of the project, which is defined by the starting time of the dummy activity A_N. Constraint set (2.3) represents the precedence relation between the activities, which implies that the finishing time of an activity must be less or equal than the starting time of its successors. Constraint set (2.4) ensures that the starting time of each activity must be within a predefined time-window. Constraint set (2.5) ensures that a worker can start an activity at most once during the whole planning horizon. Constraint set (2.6) guarantees that a worker cannot perform more than one activity at a time. Constraint sets (2.7) ensures that workers assigned to a specific activity must work simultaneously during its whole processing time. Constraint set (2.8) states that a worker can only use a skill that he can carry out. Constraint set (2.9) guarantees for each activity the fulfillment of the skill requirements. Constraint set (2.10) ensures that a worker must use exactly one skill for each assigned activity. Finally, constraint sets (2.11) and (2.12) define the decision variables as binary.

Regarding constraint (2.4), it is important to notice that es_i (resp. ls_i) denotes a lower bound (resp. upper) for the starting date associated with activity A_i. Such a time-window is for instance simply induced by the precedence graph using recursively Bellman's conditions, and a given upper bound (UB) for the makespan. Hence, the time-windows of each activity $A_i\ \forall i \in A$ are initially defined as follows:

The earliest starting times (es_i) are calculated in the following way:

$$es_0 = 0$$
$$es_i = max_{\forall j \in E_i^-}\{es_j + p_j\} \quad \forall i \in A$$

The latest starting times (ls_i) are stated as next:

$$ls_N = UB$$
$$ls_i = min_{\forall j \in E_i^+}\{ls_j - p_i\} \quad \forall i \in A$$

Let us remind, that E_i^- and E_i^+ represents the set of predecessors and successors of activity A_i.

Furthermore, considering the number of activities (N), workers (M), skills (K) and the planning horizon (T), the spatial complexity of the TIM, in terms of the number of constraints and decision variables is defined as follows.

The spatial complexity for each decision variable involved is stated by:

$$x_{i,m}^t \qquad N \cdot M \cdot T \tag{2.13}$$
$$y_{i,m}^k \qquad N \cdot M \cdot K \tag{2.14}$$

Subsequently, setting N_i^+ as equal to the number of direct successors of an activity A_i, the spatial complexity in terms of the number of constraints is given by:

$$N \cdot ((3 \cdot M) + K + (2 \cdot M \cdot K) + (M \cdot T)) + (M \cdot T) + \sum_{i \in A} N_i^+ \tag{2.15}$$

2.11.2 Time indexed model with starting times (TIMWS)

This model is also based on a time indexed perspective. The main difference between this new model and the previous one (TIM), relies in the inclusion of a new decision variable (z_i^t). Hence, this new ILP, keeps a similar structure to the one of the original model proposed by Pritsker et al. [102]. Additionally, here, we have to include different constraints for linking the three decision variables. Furthermore, the new decision variable is defined as follows:

Variables

z_i^t 1 if A_i starts at t, 0 otherwise.

Model Formulation

Given the utilization of z_i^t we can represent the starting time of an activity A_i as:

$$t_i = \sum_{t \in [0,T]} (z_i^t \cdot t) \quad \forall i \in A \tag{2.16}$$

Furthermore, the resulting mathematical formulation is given by:

$$Z[TIMWS] : Min\, C_{max} = t_N \tag{2.17}$$

$S.t.$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E_i^+ \tag{2.18}$$

$$es_i \leq t_i \leq ls_i \quad \forall i\, A \tag{2.19}$$

$$\sum_{t \in [0,T]} x_{i,m}^t \leq 1 \quad \forall i \in A, \forall m \in W \tag{2.20}$$

$$\sum_{i \in A} \sum_{d \in [t-p_i+1,t]} x_{i,m}^d \leq 1 \quad \forall t \in [0,T], \forall m \in W \tag{2.21}$$

$$x_{i,m}^t \leq z_i^t \quad \forall i \in A, \forall m \in W, \forall t \in [0,T] \tag{2.22}$$

$$x_{i,m}^t + 1 \geq z_i^t + \sum_{k \in S} y_{i,m}^k \quad \forall i \in A, \forall m \in W, \forall t \in [0,T] \tag{2.23}$$

$$y_{i,m}^k \leq M S_m^k \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.24}$$

$$\sum_{m \in W} y_{i,m}^k = b_i^k \quad \forall i \in A, \forall k \in S \tag{2.25}$$

$$\sum_{t \in [0,T]} x_{i,m}^t = \sum_{k \in S} y_{i,m}^k \quad \forall i \in A, \forall m \in W \tag{2.26}$$

$$z_i^t \in \{0,1\} \quad \forall i \in A, \forall t \in [0,t] \tag{2.27}$$

$$x_{i,m}^t \in \{0,1\} \quad \forall i \in A, \forall m \in W, \forall t \in [0,T] \tag{2.28}$$

$$y_{i,m}^k \in \{0,1\} \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.29}$$

As we stated before the objective (7.2) is to minimize the makespan of the project. Constraint set (7.3) represents the precedence relation between the activities. Constraint set (7.4) ensures that the starting time of each activity must be within a predefined time-window. Constraint set (7.5) ensures that a worker can start an activity at most once during the whole planning horizon. Constraint set (7.6) ensures that any operator can carry out at most one activity at a given time. Constraint sets (7.7) and (7.8) ensure the synchronization of the starting times of all the workers assigned to an activity. Constraint set (7.9) states that a worker can only use a mastered skill. Constraint set (7.10) guarantees the fulfillment of the skill requirements for each activity. Constraint set (7.11) ensures that a worker must use exactly one skill for each assigned activity. Finally, constraint sets (7.12), (7.13) and (7.14) define the decision variables as binary.

As it can be seen, this model (TIMWS) differs from the previous one (TIM) mainly due to the utilization of z_i^t for representing the starting time of the activities (see equations (2.1) and (7.1)).

Moreover, considering the number of activities (N), workers (M), skills (K) and the planning horizon (T), the spatial complexity of this ILP, in terms of the number of constraints and decision variables is defined as follows.

The spatial complexity for each decision variable involved is stated by:

$$z_i^t \qquad N \cdot T \tag{2.30}$$
$$x_{i,m}^t \qquad N \cdot M \cdot T \tag{2.31}$$
$$y_{i,m}^k \qquad N \cdot M \cdot K \tag{2.32}$$

Subsequently, setting N_i^+ as equal to the number of direct successors of an activity A_i, the spatial complexity in terms of the number of constraints is given by:

$$N \cdot ((2 \cdot M) + (3 \cdot M \cdot T) + (2 \cdot M \cdot K) + T) + (M \cdot T) + \sum_{i \in A} N_i^+ \tag{2.33}$$

2.11.3 Modified time indexed model with starting times (MTIMWS)

This model has a similar structure than the TIMWS, since it uses the same decision variables and almost the same formulation for all the constraints. The only difference between this two models, lays in the modeling of the precedence relations constraints, which are represented in a disaggregated manner. Hence, in this new ILP, one linear constraint is formulated for stating the precedence relations between activities at each time period. Notice, also that this disaggregated approach was originally considered by Christofides et al.[29] for solving the RCPSP. Therefore, we might obtain better results at least in terms of the linear relaxation lower bounds, considering that theoretically the resulting mathematical formulation should be stronger [2]. Furthermore, we introduce directly the associated mathematical formulation as follows:

$$Z[MTIMWS] : Min\, C_{max} = t_N \tag{2.34}$$

$S.t.$

$$\sum_{d \in [p_i, t]} z_j^d - \sum_{d \in [0, t-p_i]} z_i^d \leq 0 \quad \forall i \in A, \forall j \in E_i^+, \forall t \in [p_i, T - p_j] \tag{2.35}$$
$$z_{j,t} = 0 \quad \forall i \in A, \forall j \in E_i^+, \forall t \in [0, p_i] \tag{2.36}$$
$$z_{i,t} = 0 \quad \forall i \in A, \forall j \in E_i^+, \forall t \in [T - p_i - p_j, T] \tag{2.37}$$
$$es_i \leq t_i \leq ls_i \quad \forall i\, A \tag{2.38}$$
$$\sum_{t \in [0,T]} x_{i,m}^t \leq 1 \quad \forall i \in A, \forall m \in W \tag{2.39}$$
$$\sum_{i \in A} \sum_{d \in [t-p_i+1, t]} x_{i,m}^d \leq 1 \quad \forall t \in [0, T], \forall m \in W \tag{2.40}$$
$$x_{i,m}^t \leq z_i^t \quad \forall i \in A, \forall m \in W, \forall t \in [0, T] \tag{2.41}$$

$$x_{i,m}^t + 1 \geq z_i^t + \sum_{k \in S} y_{i,m}^k \quad \forall i \in A, \forall m \in W, \forall t \in [0,T] \tag{2.42}$$

$$y_{i,m}^k \leq M S_m^k \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.43}$$

$$\sum_{m \in W} y_{i,m}^k = b_i^k \quad \forall i \in A, \forall k \in S \tag{2.44}$$

$$\sum_{t \in [0,T]} x_{i,m}^t = \sum_{k \in S} y_{i,m}^k \quad \forall i \in A, \forall m \in W \tag{2.45}$$

$$z_i^t \in \{0,1\} \quad \forall i \in A, \forall t \in [0,T] \tag{2.46}$$

$$x_{i,m}^t \in \{0,1\} \quad \forall i \in A, \forall m \in W, \forall t \in [0,T] \tag{2.47}$$

$$y_{i,m}^k \in \{0,1\} \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.48}$$

The objective (2.34) is again to minimize the makespan of the project. Constraint sets (2.35), (2.36) and (2.37) represent the precedence relation between the activities. Constraint set (2.38) ensures that the starting time of each activity must be within a predefined time-window. Constraint set (2.39) ensures that a worker can start an activity at most once during the whole planning horizon. Constraint set (2.40) guarantees that any operator can carry out at most one activity at a given time. Constraint sets (2.41) and (2.42) ensure the synchronization of the starting times of all the workers assigned to an activity. Constraint set (2.43) states that a worker can only use a mastered skill. Constraint set (2.44) guarantees the fulfillment of the skill requirements for each activity. Constraint set (2.45) ensures that a worker must use exactly one skill for each assigned activity. Finally, constraint sets (2.46), (2.47) and (2.48) define the decision variables as binary.

Furthermore, considering the number of activities (N), workers (M), skills (K) and the planning horizon (T), the spatial complexity of this ILP, in terms of the number of constraints and decision variables is defined as follows.

The spatial complexity for each decision variable involved is stated by:

$$z_i^t \qquad N \cdot T \tag{2.49}$$

$$x_{i,m}^t \qquad N \cdot M \cdot T \tag{2.50}$$

$$y_{i,m}^k \qquad N \cdot M \cdot K \tag{2.51}$$

Subsequently, setting N_i^+ as equal to the number of direct successors of an activity A_i, the spatial complexity in terms of the number of constraints is given by:

$$N \cdot ((2 \cdot M) + (3 \cdot M \cdot T) + (2 \cdot M \cdot K) + T) + (M \cdot T) + \sum_{i \in A} (N_i^+ \cdot (T + p_i)) \tag{2.52}$$

2.11.4 Order indexed model(OIM)

This model is based on a formulation proposed by Kesen et al. [81] for a Flexible Job Shop problem. It uses a different perspective from the one used in the time indexed models. In

the OIM, instead of defining in which time point a worker starts a given activity, we focus on fixing the order in which each worker will carry out each one of the activities that he might perform. Hence, for each worker W_m we define a set L_m which represents the set of activities that he could process. Thereafter, the corresponding decision variables are given by:

Variables

$x_{i,m}^{l}$	1 if W_m performs A_i on the order l, 0 otherwise;
$y_{i,m}^{k}$	1 if worker W_m uses skill S_k to performs activity A_i, 0 otherwise;
t_i	Starting time of activity A_i;
o_m^l	Starting time of an activity performed by a worker W_m on an order l.

Model Formulation

Furthermore, the related mathematical formulation is stated as follows:

$$Z[OIM] : Min\ C_{max} = t_N \tag{2.53}$$

$S.t.$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E_i^+ \tag{2.54}$$

$$es_i \leq t_i \leq ls_i \quad \forall i\ A \tag{2.55}$$

$$o_m^l + (p_i \cdot x_{i,m}^l) \leq o_m^{l+1} \quad \forall i \in A, \forall m \in W, \forall l \in L_m \tag{2.56}$$

$$\sum_{l \in L_m} x_{i,m}^l \leq 1 \quad \forall i \in A, \forall m \in W \tag{2.57}$$

$$\sum_{i \in A} x_{i,m}^l \leq 1 \quad \forall m \in W, \forall l\ \in L_m \tag{2.58}$$

$$t_i + (T \cdot (1 - x_{i,m}^l)) \geq o_m^l \quad \forall i \in A, \forall m \in W, \forall l\ \in L_m \tag{2.59}$$

$$o_m^l + (T \cdot (1 - x_{i,m}^l)) \geq t_i \quad \forall i \in A, \forall m \in W, \forall l\ \in L_m \tag{2.60}$$

$$y_{i,m}^k \leq MS_m^k \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.61}$$

$$\sum_{m \in W} y_{i,m}^k = b_i^k \quad \forall i \in A, \forall k \in S \tag{2.62}$$

$$\sum_{l \in L_m - 1} x_{i,m}^l = \sum_{k \in S} y_{i,m}^k \quad \forall i \in A, \forall m \in W \tag{2.63}$$

$$t_i \geq 0 \quad \forall i \in A \tag{2.64}$$

$$o_m^l \geq 0 \quad \forall m \in W, \forall l \in L_m \tag{2.65}$$

$$x_{i,m}^l \in \{0, 1\} \quad \forall i \in A, \forall m \in W, \forall l \in L_m \tag{2.66}$$

$$y_{i,m}^k \in \{0, 1\} \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.67}$$

The objective (2.53) is to minimize the makespan of the project. Constraint set (2.54) represents the precedence relation between the activities. Constraint set (2.55) ensures that the starting time of each activity must be within a predefined time-window. Constraint set (2.56) defines a precedence relation between the activities assigned to a single worker. Constraint set (2.57) ensures that a worker can start a given activity at most once.

Constraint set (2.58) ensures that a worker cannot perform more than one activity at a time. Constraint sets (2.59) and (2.60) ensure that the set of workers assigned to a specific activity must work simultaneously. Constraint set (2.61) states that a worker can only use a mastered skill. Constraint set (2.62) guarantees the fulfillment of the skill requirements for each activity. Constraint set (2.63) ensures that a worker must use exactly one skill for each assigned activity. Finally, constraint sets (2.64) and (2.65) define t_i and o_m^l as positive, while (2.66) and (2.67) fix the remaining decision variables as binary.

Furthermore, considering all the parameters involved, the spatial complexity of the OIM in terms of the number of constraints and decision variables is defined as follows.

The spatial complexity for each decision variable is stated by:

$$t_i \qquad N \tag{2.68}$$
$$o_m^l \qquad M \cdot L_m \tag{2.69}$$
$$x_{i,m}^l \qquad N \cdot M \cdot L_m \tag{2.70}$$
$$y_{i,m}^k \qquad N \cdot M \cdot K \tag{2.71}$$

Subsequently, the spatial complexity in terms of the number of constraints is given by:

$$\sum_{m \in W} L_m \cdot ((4 \cdot N) + 2) + N \cdot ((2 \cdot M) + (2 \cdot M \cdot K) + K + 1) + \sum_{i \in A} N_i^+ \tag{2.72}$$

2.11.5 Flow based model(FIM)

This model is based on the classical VRP mathematical formulation [110]. It uses a different perspective to the ones explored in the previous models. In the FIM, main decision variables aim to defining if an activity A_i is scheduled before an activity A_j. Thus, we establish a set O_i^m of activities that can be performed by a worker W_m after doing an activity A_i. Additionally, we introduce another set I_j^m of activities that can be processed by a worker W_m before the performance of an activity A_j. Thereby, we describe the associated decision variables as next:

Variables

$x_{i,j}^m$	1 if W_m performs A_j after processing activity A_i, 0 otherwise;
$y_{i,m}^k$	1 if worker W_m uses skill S_k to performs activity A_i, 0 otherwise;
t_i	Starting time of activity A_i;
o_i^m	Starting time of an activity A_i performed by a worker W_m.

Model Formulation

Thereafter, the related mathematical formulation is presented as follows:

$$Z[FIM] : Min\ C_{max} = t_N \tag{2.73}$$

$S.t.$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E_i^+ \tag{2.74}$$

$$es_i \leq t_i \leq ls_i \quad \forall i\ A \tag{2.75}$$

$$o_i^m + p_i - o_j^m \leq T \cdot (1 - x_{i,j}^m) \quad \forall i \in A, \forall m \in W, \forall l \in O_i^m \tag{2.76}$$

$$\sum_{j \in O_i^m} x_{0,m}^j \leq 1 \quad \forall m \in W \tag{2.77}$$

$$\sum_{i \in I_j^m} x_{i,m}^j = \sum_{i \in O_j^m} x_{j,m}^i \quad \forall i \in A, \forall m \in W \tag{2.78}$$

$$t_i + (T \cdot (1 - \sum_{j \in O_i^m} x_{i,m}^j)) \geq o_i^m \quad \forall i \in A, \forall m \in W \tag{2.79}$$

$$o_i^m + (T \cdot (1 - \sum_{j \in O_i^m} x_{i,m}^j)) \geq t_i \quad \forall i \in A, \forall m \in W \tag{2.80}$$

$$y_{i,m}^k \leq MS_m^k \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.81}$$

$$\sum_{m \in W} y_{i,m}^k = b_i^k \quad \forall i \in A, \forall k \in S \tag{2.82}$$

$$\sum_{j \in O_i^m} x_{i,m}^j = \sum_{k \in S} y_{i,m}^k \quad \forall i \in A, \forall m \in W \tag{2.83}$$

$$t_i \geq 0 \quad \forall i \in A \tag{2.84}$$

$$o_i^m \geq 0 \quad \forall i \in A, \forall m \in W \tag{2.85}$$

$$x_{i,m}^j \in \{0, 1\} \quad \forall i, j \in A, \forall m \in W \tag{2.86}$$

$$y_{i,m}^k \in \{0, 1\} \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{2.87}$$

The objective (2.73) is to minimize the makespan of the project. Constraint set (2.74) represents the precedence relation between the activities. Constraint set (2.75) ensures that the starting time of each activity must be within a predefined time-window. Constraint set (2.76) defines a precedence relation between the activities assigned to a single worker. Constraint sets (2.77) and (2.78) ensures that a worker cannot perform more than one activity at a time. Constraint sets (2.79) and (2.80) ensure that the set of workers assigned to a specific activity must work simultaneously. Constraint set (2.81) states that a worker can only use a mastered skill. Constraint set (2.82) guarantees the fulfillment of the skill requirements for each activity. Constraint set (2.83) ensures that a worker must use exactly one skill for each assigned activity. Finally, constraint sets (2.84) and (2.85) define t_i and o_m^l as positive, while (2.86) and (2.87) both fixes the corresponding binary decision variables.

Moreover, considering all the parameters involved, the spatial complexity of this ILP in terms of the number of constraints and decision variables is defined as follows.

The spatial complexity for each decision variable is stated by:

$$t_i \quad N \tag{2.88}$$
$$o_i^m \quad N \cdot M \tag{2.89}$$
$$x_{i,m}^j \quad N \cdot N \cdot M \tag{2.90}$$
$$y_{i,m}^k \quad N \cdot M \cdot K \tag{2.91}$$

Subsequently, the spatial complexity in terms of the number of constraints is given by:

$$N \cdot ((5 \cdot M) + (2 \cdot M \cdot K) + K + 1 + (N \cdot M)) + M + \sum_{i \in A} N_i^+ \tag{2.92}$$

2.11.6 Computational results

In order to have a first insight into the hardness of our benchmark instances, computational experiments were performed using the solver Gurobi OptimizerVersion 4.5 and considering a time limit of thirty minutes. We selected a subset of the available instances for the MSPSP according to their size in terms of number of activities, skills and number of workers. In general terms, the computational results shown in this section corresponds to a subset of 70 instances, which consider between: 20 and 35 activities, 2 and 6 skills, and 2 and 10 workers. We also tested instances with bigger sizes, nevertheless, we were not able to obtain optimal solutions beyond the sizes previously mentioned.

Initially, we have that table 2.4 shows that the time indexed models (TIM, TIMWS and MTIMWS) have the highest number of variables. Despite the similarities between such models, the TIMWS and MTIMWS consider additional binary variables related to the starting time of activities. Related to the number of constraints, the TIMWS, MTIMWS and OIM present the higher values. In addition, in the time indexed models, the magnitudes of the processing times and of the time horizon (T) influences the number of variables and constraints. On the other hand, in the OIM and FIM, the number of binary variables and constraints is mainly influenced by the number of activities.

	TIM	**TIMWS**	**MTIWS**	**OIM**	**FIM**
Av. # of variables per instance	10351	15138	15138	2311	2186
Av. # of binary variables per instance	10351	15138	15138	2173	2034
Av. # of constraints per instance	1296	15186	19171	6560	2475

Table 2.4: Average number of variables and constraints for each ILP

Furthermore, table 2.5 shows that the TIM, TIMWS, MTIMWS, OIM and FIM, found feasible solutions (FS) in 9, 21, 17, 4 and 12 instances, respectively. Additionally, the TIMWS outperforms the other models in terms of number of optimal solutions reached. Since we consider a time limit of thirty minutes, for 30 instances it was not possible to find at least a feasible solution with any of the five models.

	TIM	TIMWS	MTIMWS	OIM	FIM
# of optimal solutions	13	19	16	4	7
# of feasible but non optimal solutions	9	21	17	4	12

Table 2.5: Makespan performance of each ILP

From the 7 instances in which the FIM founded optimal solutions, only 1 was proven as optimal by the solver. The solutions of the remaining 6 instances were confirmed as optimal, because all of them were either equal to the best known lower bounds (BLB) [16] or to the respective optimal solution found by any of the other models. Thus with the FIM, in 5 instances the algorithm stops until the thirty minutes limit is reached, since their solutions were considered as feasible but not proven as optimal by the solver. Such explanation does not apply for the TIM and OIM, since all of their best solutions were proven as optimal before the time limit. Regarding to TIMWS and MTIMWS only for two and one instances respectively, the related solution was not proven as optimal by the solver, but later on we prove that it was equal to the corresponding BLB.

Thereby, table 2.6 compares the linear relaxations (LR) obtained with each model against the best known lower bounds (BLB) obtained by Bellenguez-Morineau and Néron [16] and the critical path(CP). Deviations were calculated by: ($DevBLB = LR - BLB)/BLB$) and ($DevCP = LR - CP)/CP$).

	TIM	TIMWS	MTIWS	OIM	FIM
Average DevBLB	-31,34%	-33,45%	-29,45%	-42,65%	-42,65%
Average DevCP	222%	222%	222%	0%	0%

Table 2.6: Linear relaxation performance of each ILP

Results show that the OIM and FIM models do not present good linear relaxations, which is justified by the fact that for this two models the linear relaxation values are always equal to the critical path values(CP). On the other hand, the time indexed models (TIM,TIMWS and MTIMWS), present better linear relaxations that are greater than the critical path values and are closer to the best known lower bounds.

Finally, table 2.7 presents performance measures related to computational times and expanded nodes.

	TIM	TIMWS	MTIWS	OIM	FIM
LR average CPU time per instance(sec)	0,95	9,36	8,23	1,87	0,46
Average CPU time for instances	1479,27	1424,07	1427,2	1702,34	1624,8
Average # of explored nodes(millinodes)	386,18	243,04	83,397	1366,86	10126,56

Table 2.7: Global performance of each ILP

First row shows that the time indexed models spend more time to calculate the linear relaxations values than the other two models. From the second row can be concluded

that the OIM and FIM are the ones that take more CPU time until proving optimality or reaching the time limit of thirty minutes. Also, it is important to clarify that the high magnitude in the computational times are justified by the fact that in average none of the introduced models was capable of finding optimal solutions for more than the 30% of the tested instances. The last row shows that in average the time indexed models (TIMW, TIMWS and MTIMWS) explored less nodes until the algorithm stops. The other two models (OIM and FIM) expanded a greater number of nodes, due, among other reasons, to the poor linear relaxations values obtained with such models.

2.11.7 Conclusion

In this section we presented five ILP models, obtaining better results in terms of the number of optimal solutions, with the TIMWS. Overall, the time indexed models outperformed OIM and FIM in terms of the linear relaxation values, number of explored nodes and computational times. Regarding the time indexed models, we can outline that the the disegragation of the precedence relations at each time period considered by the MTIMWS, indeed, enhanced a better linear relaxation behavior. Nevertheless, this last model was outperformed by the TIMWS in terms of the number of optimal solutions found within thirty minutes. Subsequently, another important issue related to the time indexed models, is that their respective number of variables and constraints will increase depending on the estimation of time horizon (T) and on the magnitude of the processing times. It is also important to mention that the implementation of the proposed ILP models gave us also an idea of the complexity of the MSPSP, motivating the search of alternatives methods that could allows us to reach the optimal solution for a larger number of instances.

3

Column Generation Lower Bounds

In the previous chapter we proposed several ILP formulations for the MSPSP. Now, before exploring new methods for solving to optimality instances of bigger sizes to the ones solved in the previous chapter, we aim on considering new approaches that could lead us to stronger linear relaxations. Hence, in this chapter we study and propose a Column Generation (CG) approach, which is a procedure that consists in solving iteratively a linear program (RMP) until reaching a certain stopping criteria. More precisely, in CG we decompose the problem into several sub-problems that contain less constraints, which can be solved more efficiently and independently from each other [10]. Subsequently, we propose and compare different CG approaches. Finally, we perform several computational experiments, in which we compare the linear relaxation that results from applying CG with the linear relaxation obtained by the ILP models introduced in the previous chapter.

3.1 Column Generation

3.1.1 Column Generation background

So far, Column Generation (CG) had not been used to solve specifically the Multi-Skill Project Scheduling Problem (MSPSP). Although, it has been used in combination with other optimization techniques for solving Project Scheduling Problems. Particularly, Brucker and Knust [22] implemented a destructive approach for finding tight lower bounds for the RCPSP by using both constraint propagation techniques and CG. Afterwards, authors extended their work for solving the Multi-Mode RCPSP with minimal and maximal time-lags [24]. Additionally, Van den Akker et al. [113] presented a destructive lower bound based on a Column Generation approach, for certain extensions of the RCPSP. In this approach, authors used a simulated annealing approach to find a schedule for each resource, also enforced by a time-indexed integer programming formulation.

On the other hand, CG has been widely use on the Vehicle Routing Problem (VRP) and several related extensions [72, 103, 49, 21, 80, 107, 44]. Some of these problems consider similar features to those of the MSPSP. For example, Dohn et al. [44] deal with an assignment problem where a set of teams must be assigned to a set of tasks, restricted by time-windows. As it occurs in the MSPSP, assigned resources must start and finish a given activity simultaneously. Authors developed a Branch-and-Price procedure and enforce the fulfillment of such a constraint with a branching scheme that limits the starting time of a given activity. Moreover, for a particular extension of the VRP, Ioachim et al. [72] modeled the synchronization constraint directly in the master problem with the consequence that a large number of columns with a small variation in the starting times (departure times) of the tasks (flights) are generated. To handle such a drawback, they introduced a tolerance in the side constraints to allow asynchronous departure times.

Additionally, Column Generation has also been used to solve Staff Scheduling Problems [76, 90, 14, 9, 89]. These type of problems, as it occurs with the MSPSP, involves the assignment of staff members to perform a set of activities, but they normally intend to minimize a total assignment cost, considering also a predefined time horizon. For example, Jaumard [76] and Bard [9] developed B&P approaches to solve the nurse scheduling problem. This method was also use by Mason [89] to solve the Tour Scheduling Problem and by Mehrotra [90] to solve the Shift Scheduling Problem. Beliën et al. [14] also developed a B&P procedure to solve a particular problem that involves scheduling staff members (trainees).

Finally, CG has also been used to solve Shop Scheduling Problems [28, 114, 115, 55]. These problems are related to Single Machine, Flexible and Job Shop Scheduling Problems, sharing also some common features with the MSPSP. For example, Gelinas et al. [55] deal with precedence relations constraints for solving the Job Shop Scheduling Problem. They handle this constraint in the master problem and modeled the sub-problem as a single machine problem with time constraints. On another side, Van den Akker et al. [114] used CG based on a time-indexed formulation for solving Single Machine Scheduling Problems. They used Dantzig-Wolfe [37] decomposition techniques to deal with the difficulties related to the size of a time-indexed formulation, given its capacity to obtain strong lower bounds. This approach supports the one considered in this thesis, since our work is also based on a time-indexed perspective.

3.1.2 Introduction to Column Generation and problem decomposition

Column Generation (CG) is a method to solve linear programs that involve a large variables (i.e columns). Formulations of problems with a large number of variables arise in many real-life situations, e.g crew scheduling or vehicle routing, among others. Introduced independently by Dantzig and Wolfe [37] and Gilmore and Gomory [58], CG consists in solving alternatively and iteratively a (Restricted) Master Problem (RMP) and a sub-problem resulting from the decomposition of the original problem.

In algorithm 1, we describe the Column Generation procedure to solve the linear programming relaxation of the following master problem (MP):

$$Z[MP] : Min \sum_{j \in N} (c_j \cdot x_j) \tag{3.1}$$

$$S.t.$$

$$\sum_{j \in N} (a_{i,j} \cdot x_j) = d_i \ \ \forall i \in W \tag{3.2}$$

$$x_j \in 0, 1 \ \ \forall j \in N \tag{3.3}$$

Given the assumption that we deal with a huge number of variables N, we also can obtain the ILP of the MP by relaxing (3.3), implying that $x_j \geq 0, \ \forall j \ \in N$. Then, as it occurs in a major part of practical situations we assume that it is impossible to explicitly keep all columns in main memory and, thereafter, to solve the master linear program from scratch. Instead, we solve a sequence of restricted master linear programs (RMP) where each problem contains only a subset of all columns. We start the algorithm (see algorithm 1) with an initial column set $N' \subseteq N$ that contains a feasible solution. This initial solution can either be generated by a heuristic approach or by adding appropriate artificial variables to the RMP. Then, after solving the restricted master problem we can use the dual information to price out(select) candidate variables (new columns) with a reduced cost susceptible to improve the objective function associated with the master problem. Furthermore, for obtaining the reduced cost r_j of a given column j we have to solve the next sub-problem (SP) (i.e pricing problem) which uses the optimal dual solution (π) of the RMP:

$$Z[SP] : \bar{r*} = Min \ \{c_j - \sum_{i \in W} a_{i,j} \cdot \pi_i : j \in N\} \tag{3.4}$$

If the solution of the SP returns a column j with a negative reduced cost ($\bar{r*} < 0$), the current set of columns N' is updated with the inclusion of j to the RMP. The process iterates until there is not a column left with a negative reduced cost. Then, the current solution of the RMP solves the linear relaxation of the MP without having to enumerate all the columns. Furthermore, several interesting findings and features related to Column Generation are described in [87, 43].

Notice that decomposing the original problem into a master problem and a sub-problem is possible due to the exploitation of some specific structure of the problem formulation whose pricing sub-problem leads to an “easier" optimization problem such as shortest path or knapsack problems.

Depending on the structure of a problem it is possible to have a formulation that leads to a decomposition into several sub-problems that contain less complicated constraints and hence can be solved more efficiently and independently of each other. This implies that deciding how to decompose a particular problem will play an important role in obtaining efficient solutions [14].

Algorithm 1 Column Generation algorithm

1: Define initial set of columns N'.
2: $negRed = true$ ▷ It is set to false if there is not at least one possible new column with a negative reduced cost
3: **while** $negRed = true$ **do**
4: Solve RMP ▷ Use the solver for solving the RMP
5: Update dual solution(π)
6: $newColumnsList = \emptyset$
7: Price out new columns
8: Update $newColumnsList$ ▷ Inserts new column with negative reduced cost
9: Update the set of columns N' ▷ Update the set of columns
10: **if** $newColumnsList = \emptyset$ **then**
11: $negRed = false$;
12: **end if**
13: **end while**

For the MSPSP, decomposition could be done in the resources or in the activities (tasks). Decomposition on the resources is the most usual applied approach. In this case, a sub-problem consists in finding a feasible schedule for a single resource (vehicle, worker, etc)[14, 103, 49, 113, 44, 55]. Considering the features of the MSPSP, decomposition on the resources could lead to a difficult master problem, where it is necessary to deal with constraints such as the precedence relation between activities, synchronization of the starting times of the workers assigned to an activity and fulfillment of the requirements of each activity.

On the other hand, when decomposing on the activities the synchronization and the requirements fulfillment constraints are treated in the sub-problem. With this approach, the master problem will have mainly to deal with the precedence relation and workers disjunction constraints. In this case, the aim of a sub-problem is to find a feasible schedule for a single activity or task. Belien et al. [14] compared both decomposition approaches for solving a particular case of the Staff Scheduling Problem. Thereafter, they concluded that the decomposition on the activities outperformed the decomposition on the resources (staff members).

3.2 Proposed Column Generation Approach

Despites that work done so far on project scheduling with Column Generation involves decomposing on the resources, based on the previous assumptions and on the features of the MSPSP, we introduce in the next sections an activity-based decomposition approach, in which we consider two master problem formulations (MP_1 and MP_2), that lead to the same sub-problem.

3.2.1 First Master Problem (MP_1)

The basic idea underlying our CG approach relies on a time-indexed reformulation of the problem. In this new mathematical formalization, a column ω describes the execution attributes of an activity A_i. Such an activity pattern ω is represented by a triplet $[A_i(\omega), t(\omega), W(\omega)]$ where $A_i(\omega)$ denotes the activity related to ω, $t(\omega)$ its starting time, and $W(\omega)$ the subset of operators assigned to $A_i(\omega)$ for its processing. We assume that the workers assigned to ω satisfy the skill requirements of the related activity. More precisely, let us define the following parameters:

Parameters

α_i^ω 1 if activity A_i is processed in activity pattern ω, 0 otherwise;
β_i^ω t if activity A_i starts at time t in activity pattern ω, 0 otherwise;
$\gamma_{m,t}^\omega$ 1 if worker W_m is assigned on activity pattern $A_i(\omega)$ in ω at time t, 0 otherwise.

Additionally, we denote Ω as the set of all feasible activity patterns. The decision variables governing the target model are defined by:

Variables

x_ω 1 if activity pattern ω is selected, 0 otherwise.
t_i Starting time of an activity A_i;

The associated mathematical formulation can then be stated as follows:

Model formulation

$$Z[MP_1] : Min\ t_N \tag{3.5}$$

$$S.t.$$

$$\sum_{\omega\in[0,\Omega]} (x_\omega \cdot \alpha_i^\omega) = 1 \quad \forall i \in A \tag{3.6}$$

$$\sum_{\omega\in[0,\Omega]} (x_\omega \cdot \beta_i^\omega) = t_i \quad \forall i \in A \tag{3.7}$$

$$\sum_{\omega\in[0,\Omega]} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{3.8}$$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E_i^+ \tag{3.9}$$

$$es_i \leq t_i \leq ls_i \quad \forall i\ A \tag{3.10}$$

$$x_\omega \in \{0,1\} \quad \forall \omega\ \in [0,\Omega] \tag{3.11}$$

The objective is again to minimize the makespan (7.15). Constraint set (7.16) ensures that only a unique activity pattern can be assigned to any task A_i. Constraint set (7.17) recovers the associated starting times, while constraint set (7.18) ensures that any operator can carry out at most one activity at a given time. Constraint (7.19) states the precedence relations connecting the activities in G, and constraint set (7.20) ensures that the starting time of each activity must be within a predefined time-window. For this purpose, we remind to the reader that es_i (resp. ls_i) denotes a lower bound (resp. upper) for the starting date associated with activity A_i.

Furthermore, the master problem (MP) can simply be obtained by relaxing the bivalence constraints relating to decision variables x_ω. Notice that although this problem is a pure continuous linear program, it is likely to involve a huge number of columns (activity patterns).

Moreover, for understanding better the notion of activity patterns let us remind the small project of four activities illustrated in section 2.1. For simplifying the features of such an example, we consider only two workers (W_0 and W_1) and one single skill (S_0). Let us assume, that each activity requires one worker that masters skill S_0. We remind the precedence graph of the project in Figure 3.1.

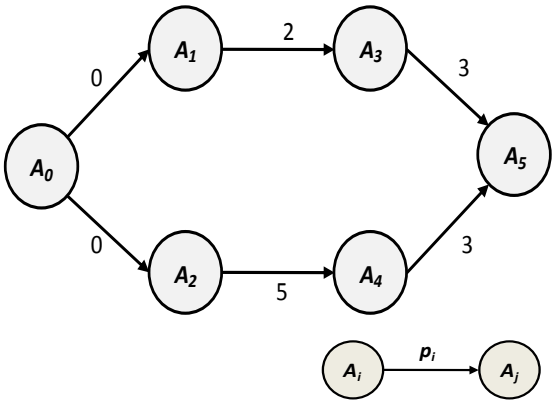

Figure 3.1: Precedence relations Graph G

Notice that A_0 and A_5 are additional dummy activities which represent the beginning and termination of the project respectively. Arcs weight represents the processing times of each activity.

Given the features of the project, we can state a feasible time-window for A_1 where $es_1 = 0$ and $ls_1 = 3$. Thus: $t_1 = \{0, 1, 2, 3\}$. Thereafter, since both W_0 and W_1 can be assigned to A_1, we can enumerate in total eight activity patterns related to A_1 (two for each possible starting time).

Now, we can introduce a single activity pattern ($\omega = 0$), related to starting A_1, at $t = 0$ using worker W_0 as follows:

$\alpha^0_1 = 1$ This parameter relates the activity pattern $\omega = 0$ with A_1;

$\beta^0_1 = 0$ This parameter relates the activity pattern $\omega = 0$ with starting time $t = 0$;

$\gamma^0_{0,0} = 1$ This parameter links the activity pattern $\omega = 0$ with worker W_0 at time-point $t = 0$, in which such a worker is unavailable when starting the execution of A_1;

$\gamma^0_{0,1} = 1$ This parameter relates the activity pattern $\omega = 0$ with worker W_0 at time-point $t = 1$, which is the last period where such a worker is unavailable when executing A_1 given that $p_1 = 2$.

Therefore, this small example illustrates how to build an activity pattern considering the notation previously introduced.

Restricted Master problem (RMP) definition

Considering the first master problem formulation, for any partial pool of activity patterns $\bar{\Omega} \subseteq \Omega$ we can then define the restricted master problem ($RMP_1(\bar{\Omega})$) in the following way:

RMP formulation

$$Z[RMP_1[\bar{\Omega})] : Min\ t_N + (L \cdot \sum_{i \in A} s_i) + (L \cdot \sum_{i \in A} u_i) \tag{3.12}$$

$$S.t.$$

$$\sum_{\omega \in [0,\bar{\Omega}]} (x_\omega \cdot \alpha_i^\omega) + s_i = 1 \quad \forall i \ \in A \tag{3.13}$$

$$\sum_{\omega \in [0,\bar{\Omega}]} (x_\omega \cdot \beta_i^\omega) + u_i = t_i \quad \forall i \in A \tag{3.14}$$

$$\sum_{\omega \in [0,\bar{\Omega}]} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{3.15}$$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E_i^+ \tag{3.16}$$

$$es_i \leq t_i \leq ls_i \quad \forall i \in A \tag{3.17}$$

$$x_\omega \geq 0 \quad \forall \omega \in [0,\bar{\Omega}] \tag{3.18}$$

In this formulation, s_i and u_i are positive slack variables that are added to the MP in order to ensure feasibility of any partial selection of activity patterns. Since L is defined as an arbitrary big positive constant, a feasible solution is guaranteed if the slack variables are equal to zero.

Assuming that an optimal solution of the $RMP_1(\bar{\Omega})$ has been computed with a standard LP solver, the corresponding simplex multipliers (dual variables) associated to constraints (7.23),(7.24),(7.25) are defined as next:

π_i	Dual variables related to the constraint set (7.23);
λ_i	Dual variables related to the constraint set (7.24);
μ_m^t	Dual variables related to the constraint set (7.25).

Thereafter, the reduced cost associated with an activity pattern $\bar{\omega}$ related to the processing of activity A_i at time t can be stated as follows:

$$r_{i,t} = 0 - \pi_i - (\lambda_i \cdot t) - \sum_{m \in W} \sum_{\theta \in [0,T]} (\mu_m^\theta \cdot \gamma_{m,\theta}^\omega) = r_{i,t}^1 + r_{i,t}^2 \tag{3.19}$$

Where:

$$r^1_{i,t} = -\pi_i - (\lambda_i \cdot t) \tag{3.20}$$

$$r^2_{i,t} = -\sum_{m \in W} \sum_{\theta \in [0,T]} (\mu^{\theta}_m \cdot \gamma^{\omega}_{m,\theta}) \tag{3.21}$$

3.2.2 Second Master Problem (MP_2)

This new master problem formulation has a similar structure to the one previously explained. In this new mathematical formalization a column (i.e activity pattern) is also represented by the triplet $[A_i(\omega), t(\omega), W(\omega)]$, where ω describes the execution attributes of an activity A_i. Now, in this new formulation we consider parameters α^{ω}_i and $\gamma^{\omega}_{m,t}$, which were already introduced in the first master problem formulation (MP_1). This new mathematical model relies on an alternative way for integrating the precedence relations constraints. Hence, let us consider a particular precedence relation between a couple of activities A_i and A_j ($A_j \succ A_i$), that must be processed in the time slot $\Theta_{i,j} = [es_i, ls_j + p_j]$. Mapping this time slot, we can extend the notion of an activity pattern ω by defining a new parameter $\delta(\theta)^{\omega}_{i,j}$ in the following way:

Additional parameters

If ω corresponds to the execution of A_i at a starting time t:

$\delta(\theta)^{\omega}_{i,j}$ $\quad 1\ \forall\theta \in [es_i, t + p_i - 1]$;
$\delta(\theta)^{\omega}_{i,j}$ $\quad 0\ \forall\theta \in [t + p_i, ls_j + p_j]$.

If ω corresponds to the execution of A_j at a starting time t:

$\delta(\theta)^{\omega}_{i,j}$ $\quad 1\ \forall\theta \in [es_i, t - 1]$;
$\delta(\theta)^{\omega}_{i,j}$ $\quad 0\ \forall\theta \in [t, ls_j + p_j]$.

If ω does not correspond to the execution of neither A_i nor A_j:

$\delta(\theta)^{\omega}_{i,j}$ $\quad 0\ \forall\theta \in [es_i, ls_j + p_j]$.

Clearly, two work patterns ω and ω' should be consistent for the precedence relation constraint of A_i and A_j, if:

$$(\delta(\theta)^{\omega}_{i,j} \cdot x_{\omega}) + (\delta(\theta)^{\omega'}{}_{i,j} \cdot x_{\omega'}) \leq 1 \quad \forall\theta \in \Theta_{i,j} \tag{3.22}$$

Let us remind that the earliest and latest starting times of an activity A_i (es_i and ls_i) are initially induced by the precedence graph using recursively Bellman's conditions, and a given upper bound (UB) for the makespan.

Additionally, we also consider a coefficient c_{ω}, related to each activity pattern ω. Hence, given that $t(\omega)$ represents the starting time linked to activity pattern ω, we define c_{ω} as follows:

c_{ω} $\quad$ $t(\omega)$ if activity pattern ω is related to the execution of the dummy activity A_N, 0 otherwise.;

Let us also recall that Ω represents the set of all feasible activity patterns. The only decision variable governing the target model is x_ω which was already introduced for MP_1. Therefore, the associated mathematical formulation can then be stated as follows:

Model formulation

$$Z[MP_2] : Min \sum_{\omega \in [0,\Omega]} (c_\omega \cdot x_\omega) \tag{3.23}$$

$S.t.$

$$\sum_{\omega \in [0,\Omega]} (x_\omega \cdot \alpha_i^\omega) = 1 \quad \forall i \in A \tag{3.24}$$

$$\sum_{\omega \in [0,\Omega]} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{3.25}$$

$$\sum_{\omega \in [0,\Omega]} (x_\omega \cdot \delta(t)_{i,j}^\omega) \leq 1 \quad \forall i \in A, \forall j \in E_i^+, \forall t \in \Theta i,j \tag{3.26}$$

$$x_\omega \in \{0,1\} \quad \forall \omega \ \in [0,\Omega] \tag{3.27}$$

Notice that this formulation has a structure (set packing problem) with a pure 0-1 coefficients constraint matrix. More precisely, we have that constraint set (3.24) ensures that only a unique activity pattern can be assigned to any task A_i. Constraint set (3.25) ensures that any operator can carry out at most one activity at a given time. Constraint (3.26) states the precedence relations connecting the activities in G at give time-point t. The master problem (MP_2) can simply be obtained by relaxing the bivalence constraints relating to decision variables x_ω. Notice that although this problem is a pure continuous linear program, it is likely to involve a huge number of columns (activity patterns).

Restricted Master problem (RMP) definition

Considering this second master problem formulation (MP_2), for any partial pool of activity patterns $\bar{\Omega} \subseteq \Omega$ we can then define the restricted master problem ($RMP_2(\bar{\Omega})$) in the following way:

RMP formulation

$$Z[RMP_2[\bar{\Omega}]] : Min \sum_{\omega \in [0,\Omega]} (c_\omega \cdot x_\omega) + (L \cdot \sum_{i \in A} s_i) + (L \cdot \sum_{i \in A} u_i) \tag{3.28}$$

$S.t.$

$$\sum_{\omega \in [0,\bar{\Omega}]} (x_\omega \cdot \alpha_i^\omega) + s_i = 1 \quad \forall i \ \in A \tag{3.29}$$

$$\sum_{\omega \in [0,\bar{\Omega}]} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{3.30}$$

$$\sum_{\omega \in [0,\Omega]} (x_\omega \cdot \delta(t)_{i,j}^\omega) \leq 1 \quad \forall i \in A, \forall j \in E_i^+, \forall t \in \Theta_{i,j} \tag{3.31}$$

$$x_\omega \geq 0 \quad \forall \omega \in [0,\bar{\Omega}] \tag{3.32}$$

In this formulation, we also include a positive slack variable s_i to ensure feasibility of any partial selection of activity patterns. Since L is defined as an arbitrary big positive constant, a feasible solution is guaranteed if the slack variables are equal to zero.

Assuming that an optimal solution of the $RMP_2(\bar{\Omega})$ has been computed with a standard LP solver, the corresponding simplex multipliers (dual variables) associated to constraints (3.29),(3.30), and (3.31) are defined as next:

π_i Dual variables related to the constraint set (3.29);
μ_m^t Dual variables related to the constraint set (3.30);
$\eta_{i,j}^t$ Dual variables related to the constraint set (3.31).

Thereafter, the reduced cost associated with an activity pattern $\bar{\omega}$ related to the processing of activity A_i at time t can be stated as follows:

$$r_{i,t} = 0 - \pi_i - \sum_{i \in A} \sum_{j \in E_i^+} \sum_{\theta \in \Theta_{i,j}} (\delta(\theta)_{i,j}^{\omega} \cdot \eta_{i,j}^{\theta}) - \sum_{m \in W} \sum_{\theta \in [0,T]} (\mu_m^{\theta} \cdot \gamma_{m,\theta}^{\omega}) = r_{i,t}^1 + r_{i,t}^2 \quad (3.33)$$

Where:

$$r_{i,t}^1 = -\pi_i - \sum_{i \in A} \sum_{j \in E_i^+} \sum_{\theta \in \Theta_{i,j}} (\delta(\theta)_{i,j}^{\omega} \cdot \eta_{i,j}^{t}) \quad (3.34)$$

$$r_{i,t}^2 = - \sum_{m \in W} \sum_{\theta \in [0,T]} (\mu_m^{\theta} \cdot \gamma_{m,\theta}^{\omega}) \quad (3.35)$$

Given that we consider an activity-based decomposition approach, clearly the sub-problem definition will be the same, despites which of the two described restricted master problems (RMP_1 or RMP_2) is used.

3.2.3 Column Generation Sub-problem (SP)

Since activity A_i starts at time t, each worker devoted to its processing should work during time instants $t, t+1, .., t+p_i-1$, inducing a total cost:

$$\sigma_m(t) = - \sum_{\theta=t}^{t+p_i-1} \mu_m^{\theta} \quad (3.36)$$

To formally state the Column Generation sub-problem (SP), let us define the following decision variables:

Variables

y_m 1 if worker W_m is assigned to perform activity A_i, 0 otherwise;
z_m^k 1 if worker W_m uses skill S_k to perform activity A_i, 0 otherwise.

Clearly, finding an activity pattern for A_i, starting at time t, with a minimal reduced cost leads to the following sub-problem:

Sub-problem formulation

$$Z[SP] : Min\ r_{i,t}^2 = \sum_{m \in W} (\sigma_m(t) \cdot y_m) \tag{3.37}$$

$S.t.$

$$\sum_{m \in W} z_m^k = b_{i,k} \quad \forall k \in S \tag{3.38}$$

$$y_m = \sum_{k \in S} z_m^k \quad \forall m \in W \tag{3.39}$$

$$y_m \in \{0, 1\} \quad \forall m \in W \tag{3.40}$$

$$z_m^k \in \{0, 1\} \quad \forall m \in W, \forall k \in S \tag{3.41}$$

In this formulation, the objective is to minimize the total assignment cost to perform activity A_i at a time t. Constraint set (7.34) guarantees its requirements fulfillment. Constraint set (7.35) ensures that an assigned worker uses only one skill. Finally, constraint sets (7.36) and (7.37) define the decision variables as binary. Moreover, we can state that solving the SP aims to exhibit a feasible selection of workers/skills for processing activity A_i at time t.

Furthermore, after obtaining the value of $r_{i,t}^2$ we can calculate the reduced cost ($r_{i,t} = r_{i,t}^1 + r_{i,t}^2$) of a given activity pattern. Hence, if $r_{i,t} < 0$, then the corresponding column is candidate to enter the base since its negative reduced cost will decrease the objective function of the current restricted master problem RMP($\bar{\Omega}$). Consequently, this activity pattern can be added to the current pool of columns by setting:

$$\bar{\Omega} \leftarrow \bar{\Omega} \cup \bar{\omega} \tag{3.42}$$

$$\alpha_i^{\bar{\omega}} = 1 \tag{3.43}$$

$$\beta_i^{\bar{\omega}} = t \tag{3.44}$$

$$\gamma_{m,t}^{\bar{\omega}} = y_m \quad \forall m \in W, \forall \theta \in [t, t + p_i - 1] \tag{3.45}$$

Of course, an enumeration on each activity in each potential starting date ($es_i \leq t_i \leq ls_i$) is necessary for exhibiting an activity pattern with global minimal reduced cost. We refer to the next section for more details related to the global resolution method.

3.2.4 Solution Method

Solving the CG sub-problem

As it is shown in figure 3.2, the assignment problem with minimal cost, which corresponds to the sub-problem(SP), can be represented as a min-cost max-flow problem [17]. In graph F_c, the skills requirements $b_{i,k}$ of activity A_i are represented as the maximum capacity of the arcs between the source and each skill S_k. The arcs between each skill S_k and each worker W_m have a weight equal to one, to ensure that a worker cannot be assigned to more than one unit of a skill S_k. Finally, the arcs between each worker W_m and the sink have an assignment cost $\sigma_m(t)$ and a maximal capacity equal to one. The capacity value

ensures that a worker cannot be assigned to more than one skill. The objective is then to obtain an assignment that guarantees the fulfillment of the requirements of A_i (maximal flow), by minimizing $r^2_{i,t}$ (minimal cost).

To solve the sub-problem we use the min-cost max-flow algorithm proposed by Busacker and Gowen [25] as it was done in [17]. The complexity of such an algorithm is defined as $\mathcal{O}(Q^3)$, where Q represents the number of nodes. In the worst case scenario the number of nodes can be equal to the sum of the number of workers and the number of skills, leading to a complexity of $\mathcal{O}((M+K)^3)$.

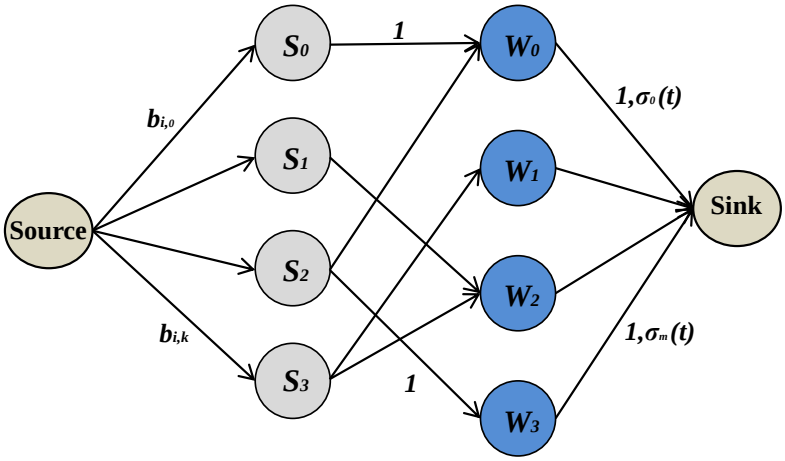

Figure 3.2: Graph F_c skills assignment for activity A_i.

Furthermore, the graph F_c only includes the set of skills that are required by activity A_i and the set of workers that master at least one of those skills. Finally, after building the graph with the maximum flow at a minimum cost, we identify each worker $W_m \in W$ with a positive flow for setting the subset of workers assigned to A_i at time t. It is important to mention that we store the solution obtained for each activity at a given starting time, in order to ensure that we do not generate the same activity pattern more than once.

Column Selection

Given that the decomposition is done on the activities and on their possible starting times, selecting the most promising column(s) implies having to solve a sub-problem for each activity at each possible starting time t within its corresponding time-window.

To limit the number of sub-problems solved at each iteration of the CG procedure, it is possible to filter the activities and starting time values that might lead to columns with a negative reduced cost($r_{i,t} < 0$). According to duality properties, the total assignment cost of a given sub-problem $r^2_{i,t}$ (7.31) must be always greater than or equal to zero. Thus, the columns that might have a negative reduced cost will be the ones for which $r^1_{i,t}$ (7.30) is less than zero. Considering that such an expression can be computed with the simplex

multipliers obtained after solving to optimality the RMP($\bar{\Omega}$), a sub-problem may have to be solved only for the columns in which $r^1_{i,t}$ is less than zero.

Additionally, we also can easily estimate a lower bound $lr^2_{i,t}$ for $r^2_{i,t}$. Given that we only consider the sub-problems for which $r^1_{i,t}$ is less than zero, after computing $lr^2_{i,t}$, we are able to obtain a lower bound $lr_{i,t}$ ($lr_{i,t} = r^1_{i,t} + lr^2_{i,t}$) for $r_{i,t}$. Thereafter, we can solve the respective subset of sub-problems, for which $lr_{i,t}$ is less than zero, and then calculate the real total reduced cost ($r_{i,t}$) to define which column(s) will be added to the RMP($\bar{\Omega}$).

To obtain the value of $lr^2_{i,t}$ for a specific activity A_i and starting time t: first, among the workers that master at least one of the required skills to perform A_i, we select the ones with the lowest assignment cost $\sigma_m(t)$. Afterwards, we can get the value of $lr^2_{i,t}$ by summing the assignment costs of the selected workers. This assignment approach ensures obtaining a lower bound for the value of $r^2_{i,t}$, since we choose the cheapest set of required workers without checking if there is a feasible skill assignment.

Before solving a sub-problem, we check if the group of workers used to obtain the value of $lr^2_{i,t}$, have been previously assigned to an activity pattern related to A_i with a starting time different to t. If that is the case, we can imply that such an assignment has been proved as feasible already, thus we don't have to solve the sub-problem, concluding that $r^2_{i,t}$ is equal to $lr^2_{i,t}$, thus $r_{i,t}$ is equal to $lr_{i,t}$.

After defining the subset of sub-problems that could lead to an improvement in the RMP($\bar{\Omega}$) objective value, we have to decide if we add either one or several columns per iteration [87]. We tested both approaches by applying CG in the root node, obtaining better results with the second one in terms of the average computational time. It is important to notice that adding several columns per iteration might lead to spend more time on solving the RMP($\bar{\Omega}$). Nevertheless, the CG procedure might require less iterations until there are not negative columns left.

Particularly, when generating several columns per iteration we aim on adding at least one column per activity. Additionally, we might generate several columns for a given activity A_i at a time-point t whenever there are different subsets of possible assigned workers that lead to the same assignment cost $r^2_{i,t}$.

Initializing the pool of activity patterns

For the first CG iteration, we initialize the subset of columns $\bar{\Omega}$ for solving the RMP($\bar{\Omega}$) according to a schedule obtained by the Tabu Search (TS) developed by Bellenguez-Morineau and Néron [17]. In this approach a solution is evaluated by a serial schedule scheme. A solution S, built on a priority list L, is considered to be a neighbor of a solution S', from a list L', if L' is computed from L with only one swap of two activities. Obtained results reveal an average deviation from the best lower bound [16] around 6% on the instances we tested in our work. Nevertheless, it is important to state that the reformulations proposed in 3.2.1 and 3.2.2 allows solving the RMP without having an initial set

of columns $\bar{\Omega}$. Thereafter, preliminary results show that initializing the pool of columns by means of the TS allows us to prove optimality faster and enhance the possibility of keeping a structure of activity patterns that could lead to an integer feasible schedule.

3.3 Lagrangian Relaxation

In the previous Section, we considered a linear programming perspective based on a Column Generation approach. A different notion can be taken into account if we consider its relation with Lagrangian relaxation.

3.3.1 Lagrangian Relaxation background

Lagrangian relaxation was first used by Geoffrion [56] for obtaining lower bounds in integer programming. Thereafter, there has been an extensive work done in this topic over the last years [12, 11, 118, 51].

Now, for a further explanation, let's consider the following integer problem:

$$
\begin{aligned}
& Z[P] : Min\ c^T x && (3.46)\\
& S.t. \\
& A \cdot x \geq b && (3.47)\\
& M \cdot x \leq d && (3.48)\\
& x \in \mathbb{Z}_+ && (3.49)
\end{aligned}
$$

where $x \in \mathbb{R}^n, b \in \mathbb{R}^m$ and $d \in \mathbb{R}^k$. Suppose that constraints (3.47) are hard constraints in such a way, where without them, the optimization problem becomes easier to solve. Afterwards, these hard constraints, can be dualized by adding them to the objective function with a penalty term w. Then, we obtain the next Lagrangian function:

$$
\begin{aligned}
& Z[L(w)] = Min\ c^T x - w(b - Ax) && (3.50)\\
& S.t. \\
& M \cdot x \leq d && (3.51)\\
& x \in \mathbb{Z}_+ && (3.52)
\end{aligned}
$$

Where w corresponds to the vector that represents the Lagrangian multipliers associated to the dualized hard constraints. When we dualize the inequality constraints (3.47) the corresponding Lagrangian multipliers are restricted in sign, obtaining that $w \geq 0$.

$Z[L(w)]$ defines a lower bound on the original problem P for any fixed vector w given that each feasible solution for the original problem is also feasible for the Lagrangian function. Hence, we can obtain the best possible lower bound by solving the Lagrangian dual problem (LDP):

$$Z[LDP] = Max_w\ L(w) \tag{3.53}$$

$$\tag{3.54}$$

It can be shown [56] that the optimal solution of the Lagrangian dual problem always provides a lower bound on the original problem that is at least as good as the objective value of the linear relaxation LP:

$$Z[LP] \leq Z[LDP] \leq Z[P] \tag{3.55}$$

$$\tag{3.56}$$

Furthermore, the Lagrangian dual problem maximizes a piecewise linear concave, but non-differentiable function $L(w)$. This implies that the LDP is not everywhere differentiable. Thereafter, in the context of combinatorial optimization one efficient way to solve LDP is to use a subgradient procedure introduced by Held and Karp [66] which iteratively updates the lagrangian multipliers.

Nevertheless other methods like volume [7], bundle [48] and analytic center cutting plane methods [59] among others (see [19]) can be used for solving the Lagrangian dual. We focus particularly in description of the subgradient method since it is the most diffused one to solve the LDP. Besides the fact that it was the first one used in the context of combinatorial optimization, it has, at least, two main advantages: it is easy to code and has minimal memory requirements. However, since it does not keep in memory the obtained solutions, it does not guarantee anything (not even feasibility) about the solution of the original problem.

3.3.2 Subgradient Algorithm

The subgradient algorithm is an iterative search procedure developed for optimizing non-differentiable functions. It is well-known that a differentiable function f can be optimized by means of an iterative gradient method by starting with an initial solution u^0, the next sequence:

$$u^{t+1} = u^t + \rho^t \cdot \bigtriangledown f(u^t) \tag{3.57}$$

After some iterations, this sequence converges to an optimal solution with $\bigtriangledown f(u^t)$ as the gradient of f at u^t and ρ^t as a suitable step length. Nevertheless, we cannot use a gradient method since we are dealing with non-differentiable functions, for which there are some points that doesn't have a gradient. Instead, we replace gradients with subgradients by using a subgradient method which is a generalization of the gradient method for the non-differentiable case.

Now, in the context of the previous example, a subgradient at a point w_0 of a concave function L: $\mathbb{R}^m \rightarrow \mathbb{R}^1$ is a vector $s \in \mathbb{R}^m$ such that $L(q) \leq L(w_0) + s \cdot (q - w_0)$ for all $q \in \mathbb{R}^m$. Thereafter, the vector $b - (A \cdot x)$ is easily shown to be a subgradient s for the Lagrangian function with x as optimal solution to this problem [117].

Finally for updating vector of Lagrangian multipliers (w) we have:

$$w^{t+1} = max\{0, w^t + (\rho^t \cdot s)\} \tag{3.58}$$

where the subgradient s and the step size ρ^t are defined also as follows:

$$s = b - (A \cdot x^t) \tag{3.59}$$

$$\rho^t = \theta \cdot \frac{(UB - L(w_t))}{(s)^2} \tag{3.60}$$

This last expression (3.60) is a known step length rule which was empirically justified by Held et al. [67]. Moreover, this rule is less expensive in terms of CPU times in comparison to other step length rules with proven convergence [101].

We also consider setting the step size parameter $\theta = 2$, as was proposed by Held and Karp [66]. Additionally it is important to set a limit number of iterations which defines also the accuracy of the solution.

3.4 Lagrangian Relaxation and Column Generation

3.4.1 Introduction

Typically, Column Generation is used to solve the LP-relaxation of the master problem, but it can also be combined with Lagrangian relaxation as we will discuss in this section. Additionally, we describe the resulting models and procedures after combining these two methods for solving the MSPSP. Finally, we show and discuss the obtained results.

According to Wolsey [117], it is possible to solve the Lagrangian dual either by means of the subgradient method or by solving the linear relaxation of the extensive formulation (RMP) by using a CG approach. Thereafter, the optimal lower bound of the restricted linear master problem (RMP) and the best Lagrangian dual will have the same value. Both solution methods for the Lagrangian dual have advantages and disadvantages, hence some authors have proposed procedures that try to combine the advantages of both approaches [71, 8].

As we will show later on, each Lagrangian multiplier vector is linked with the dual variables related to the relaxed constraint. Consequently, this implies that the dual values obtained by solving the RMP can be estimated by the Lagrangian multipliers used in the related Lagrangian dual problem. Thus, instead of solving the RMP with the simplex method by using a solver, we can use the subgradient procedure for solving the Lagrangian dual approximately and obtaining the Lagrangian multipliers which at the end of the subgradient phase can be used for estimating the values of the dual variables related to the constraints of the RMP. Finally, the Lagrangian multipliers can be used to price out new columns.

Overall, there are different reasons for using this last mentioned approach. The subgradient method is fast, easy to implement, and does not require a commercial solver. When solving the RMP with a simplex method, we obtain a basic dual solution that corresponds to a vertex of the optimal face of the dual polyhedron. Given, that a new column of the RMP may cut that vertex, a dual solution interior (in the center) of the dual face allows stronger dual cuts (i.e. better primal columns). Bixby et al. [20] and Barnhart et al. [10] obtained from their research that this may improve the convergence of a Column Generation algorithm and reduce degeneracy. The subgradient method naturally provides non-basic solutions with many non-zero elements. Jans and Degraeve [74] and Huisman et al. [71] provide computational results that indicates that Lagrangian multipliers are beneficial. Finally, is also shown that during the subgradient phase possible feasible solutions are generated.

More specifically, considering the arguments previously mentioned, we aim for combining CG with Lagrangian relaxation, which in principle could lead to a faster way for solving the RMP, rather than using only the simplex method. Hence, given that we proposed two different master problem formulations, we introduce two lagrangian models, one based on MP_1 (3.2.1) and another based on MP_2 (3.2.2).

3.4.2 MP_1 based model for combining Lagrangian Relaxation and Column Generation

Before introducing the first Lagrangian model, we present a reformulation of MP_1. The new resulting linear model, allows us to obtain a Lagrangian function in terms of the Lagrangian multipliers that afterwards, we will use for estimating the dual variables π_i, λ_i and μ_m^t linked to the corresponding constraints of the RMP proposed in section 3.2.1).

$$Z[RMP_1'(\bar{\Omega}]) : Min\ t_N \tag{3.61}$$

$$S.t.$$

$$\sum_{\omega\in[0,\bar{\Omega}]} (x_\omega \cdot \alpha_i^\omega) = 1 \quad \forall i \in A \tag{3.62}$$

$$\sum_{\omega\in[0,\bar{\Omega}]} (x_\omega \cdot \beta_i^\omega) = t_i \quad \forall i \in A \tag{3.63}$$

$$\sum_{\omega\in[0,\bar{\Omega}]} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{3.64}$$

$$\sum_{t\in[0,T]} \sum_{m\in W} \sum_{\omega\in[0,\Omega]} (x_\omega \cdot \gamma_{m,t}^\omega) = \sum_{i\in A} \sum_{k\in S} (p_i \cdot b_i^k) \tag{3.65}$$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E_i^+ \tag{3.66}$$

$$es_i \leq t_i \leq ls_i \quad \forall i \in A \tag{3.67}$$

$$x_\omega \geq 0 \quad \forall \omega \ \in [0,\bar{\Omega}] \tag{3.68}$$

As it can be seen, this model (RMP_1') is based on the master problem introduced in 3.2.1 (MP_1), but it includes an additional surrogate constraint (3.65) which enhance the

relaxation showed in the next section. Such a constraint establishes that the accumulated time per resource unit assigned (left term) must be equal to the total amount of time per resource unit required during the whole project duration (right term).

Lagrangian Relaxation model

Now, let us associate with constraints (3.62),(3.63) and (3.64) the respective Lagrangian multipliers ($\pi_i, i \in A$), ($\lambda_i, i \in A$) and ($\mu_m^t, m \in W, t \in T$). The corresponding Lagrangian function can be written as follows:

$$\Gamma(x,t,\pi,\lambda,\mu) = t_N + \sum_{i\in A}\pi_i \cdot (1 - \sum_{\omega\in[0,\Omega]}(x_\omega \cdot \alpha_i^\omega)) + \sum_{i\in A}\lambda_i \cdot (t_i - \sum_{\omega\in[0,\Omega]}(x_\omega \cdot \beta_i^\omega)) + \sum_{m\in W}\sum_{t\in[0,T]}\mu_m^t \cdot (1 - \sum_{\omega\in[0,\Omega]}(x_\omega \cdot \gamma_{m,t}^\omega)) \quad (3.69)$$

$$\Gamma(x,t,\pi,\lambda,\mu) = t_N + \sum_{i\in A}(\lambda_i \cdot t_i) + \sum_{\omega\in[0,\Omega]}((-\alpha_i^\omega \cdot \pi_i) - (\beta_i^\omega \cdot \lambda_i) - \sum_{m\in W}\sum_{t\in[0,T]}(\mu_m^t \cdot \gamma_{m,t}^\omega)) \cdot x_\omega + \sum_{i\in A}\pi_i + \sum_{m\in W}\sum_{t\in[0,T]}\mu_m^t \quad (3.70)$$

For a given distribution (π, λ, μ) of Lagrangian multipliers, the associated dual function $L(\pi, \lambda, \mu)$ can be computed solving the following independent Lagrangian sub-problems:

$$Z[LSP_1(\lambda)] : Min\ t_N + \sum_{i\in A}(\lambda_i \cdot t_i) \quad (3.71)$$

$$S.t.$$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j\ \in E_i^+ \quad (3.72)$$

$$es_i \leq t_i \leq ls_i \quad \forall i \in A \quad (3.73)$$

and

$$Z[LSP_2(\pi,\lambda,\mu,x)] : Min \sum_{\omega\in[0,\Omega]}(c'_\omega \cdot x_\omega) \quad (3.74)$$

$$S.t.$$

$$\sum_{\omega\in[0,\Omega]}(x_\omega \cdot r_\omega) = \sum_{i\in A}\sum_{k\in S}(p_i \cdot b_i^k) \quad (3.75)$$

where

$$c'_\omega = -\pi_{i(\omega)} - (t(\omega) \cdot \lambda_i) - \sum_{m\in W}\sum_{t\in[0,T]}(\mu_m^t \cdot \gamma_{m,t}^\omega)) \quad (3.76)$$

$$r_\omega = \sum_{k\in S}(p_{i(\omega)} \cdot b_i^k) \quad (3.77)$$

Hence, we have obviously:

$$Z[L(\pi,\lambda,\mu)] = Z[LSP_1(\pi,\lambda,\mu,t)] + Z[LSP_2(\pi,\lambda,\mu,x)] + \sum_{i\in A}\pi_i + \sum_{m\in W}\sum_{t\in[0,T]}\mu_m^t \tag{3.78}$$

Initially we solved the first Lagrangian sub-problem ($LSP_1(\lambda)$) with a solver, but we also explored an alternative solution approach that will be explained later on.

Clearly, $LSP_2(\pi,\lambda,\mu)$, which is a classical 0-1 Knapsack problem, can be solved with a pseudo-polynomial time complexity of $\mathcal{O}(qB)(q = |\bar{\Omega}|)$, which can be quite time consuming when the pool of patterns $\bar{\Omega}$ increases. Nevertheless, we can focus on the linear relaxation of MP_1' according to the work pattern generation process ($x_\omega \geq 0 \ \ \forall\omega \ \in [0,\bar{\Omega}]$). First, let us sort the work pattern in $\bar{\Omega}$ in such a way that:

$$c'_{\omega_1}/r_{\omega_1} \leq c'_{\omega_2}/r_{\omega_2} \leq \ldots \leq c'_{\omega_q}/r_{\omega_q} \tag{3.79}$$

Now, let s be the maximal index in $]q]$ such that: $\sum_{j=0}^{s} r_{\omega_j} \leq \sum_{i\in A}\sum_{k\in S}(p_i \cdot b_i^k)$. An optimal solution to LSP_2 is given by:

$$\bar{x_{\omega_j}} = 1 \ \ \forall j \ \in [0,s] \tag{3.80}$$

$$\bar{x_{\omega_{s+1}}} = \frac{(\sum_{i\in A}\sum_{k\in S}(p_i \cdot b_i^k)) - r_{\omega_j}}{r_{\omega_{s+1}}} \tag{3.81}$$

$$\bar{x_{\omega_j}} = 0 \ \ \forall j \ \in [s+2,q] \tag{3.82}$$

$Z[L(\pi,\lambda,\mu)]$ defines a lower bound on the RMP_1', given that each feasible solution for the original problem is also feasible for the Lagrangian function. Hence, we can obtain the best possible lower bound by solving the Lagrangian dual problem ($LDRMP_1'$):

$$Z[LDP] = Max_{\pi,\lambda,\mu}\ L(\pi,\lambda,\mu) \tag{3.83}$$

Thereafter, as we explained in 3.3.2 we use the subgradient procedure for estimating the values of the Lagrangian multipliers (π,λ,μ). Hence, after solving $LSP_1(\lambda)$ and $LSP_2(\pi,\lambda,\mu)$ we obtain the starting times vector $\bar{t}$ and the column assignment vector $\bar{x}$ from the solution of each sub-problem respectively. Consequently, we can define a subgradient for $L(\pi,\lambda,\mu)$ as follows:

$$\phi_i^1 = 1 - \sum_{\omega\in[0,\Omega]}(x_\omega \cdot \alpha_i^\omega) \ \ \forall i \in A \tag{3.84}$$

$$\phi_i^2 = \bar{t}_i - \sum_{\omega\in[0,\Omega]}(x_\omega \cdot \beta_i^\omega) \ \ \forall i \in A \tag{3.85}$$

$$\varphi_m^t = (1 - \sum_{\omega \in [0,\Omega]} (x_\omega \cdot \gamma_{m,t}^\omega)) \ \ \forall m \in W \forall t \in [0,T] \tag{3.86}$$

Now, given an upper bound UB for the makespan and a size parameter θ, we can update the current Lagrangian multipliers by:

$$\pi_i = \pi_i + (\rho \cdot \phi_i^1) \tag{3.87}$$
$$\lambda_i = \lambda_i + (\rho \cdot \phi_i^2) \tag{3.88}$$
$$\mu_m^t = min\{0, \mu_m^t + (\rho \cdot \varphi_m^t)\} \tag{3.89}$$

The step size ρ is defined also as follows:

$$\rho = \frac{(UB - L(\pi, \lambda, \mu))}{Norm} \tag{3.90}$$

where $Norm$ is given by:

$$Norm = \sum_{i \in A} (\phi_i^1 + \phi_i^2)^2 + \sum_{m \in W} \sum_{t \in [0,T]} {\varphi_m^t}^2 \tag{3.91}$$

As was justified in section 3.3.2 we start the subgradient procedure by setting $\theta = 2$. At the end of each iteration we update the parameter $theta$ with a systematic geometric revision: $\theta = \kappa \cdot \theta$. Normally the second parameter κ ranges between 0,87 and 0,9995, depending on the targeting problem.

As said before, we update the Lagrangian multipliers during a limited number of iterations ϵ, in order to have an estimation of the dual multipliers for pricing out new columns. Note that Lagrangian multiplier μ is defined as non- positive (see equation (3.89)), in order to keep the same structure of the non-positive dual variables linked to the disjunction constraint (see equation (3.64))of the RMP_1'. The other two Lagrangian multipliers (π and λ) are defined as free of sign to keep the similarity with the related dual variables linked to constraints (3.62) and (3.63) respectively.

Despites the sign definition of the Lagrangian multipliers, it can be seen in the structure of the Lagrangian function (3.69) that the relaxed constraints are correctly penalized.

Finally, when there are no more columns with negative reduced cost by using the Lagrangian multipliers, we continue with the CG procedure, and solving the RMP with the simplex method by using the solver for obtaining the values of the dual variables for pricing out new columns. The whole procedure is summarized by the algorithm 2.

Alternative procedures for solving $LSP_1(\lambda)$ ($ALSP_1$)

Algorithm 2 General algorithm for combining CG with Lagrangian Relaxation considering the MP_1 formulation

Input: $\bar{\Omega}$ Current pool of activity patterns;

A Set of activities;

W Set of workers;

T Upper bound for the planning horizon (makespan).

Output: π, λ, μ Lagrangian multipliers

$negRed = true$ ▷ It is set to false if there is not at least one possible new column with a negative reduced cost
while $negRed = true$ **do**
    $Iter = 0$ ▷ Initialize the iterator for the subgradient procedure
    $\theta = 2$ ▷ Initialize the step size parameter
    Set π, λ, μ equal to zero ▷ Initialize the lagrangian multipliers
    while $iter \leq \epsilon$ **do** ▷ Starts the subgradient procedure loop
        Solve $LSP_1(\lambda) \rightarrow$ optimal solution $(\bar{t}_i, \forall i \in A)$
        Solve $LSP_2(\pi, \lambda, \mu) \rightarrow$ optimal solution $(\bar{x}_\omega, \forall \omega \in [0, \bar{\Omega}])$
        Compute $Z(L(\pi, \lambda, \mu))$
        Compute $Z(L(\pi, \lambda, \mu))$
        Compute $\phi_i^1, \ \forall i \in A$ ▷ Compute Subgradient
        Compute $\phi_i^2, \ \forall i \in A$
        Compute $\varphi_m^t \ \forall m \in W, \forall t \in [0, T]$
        Compute $\pi_i, \ \forall i \in A$ ▷ Update Lagrangian Multipliers
        Compute $\lambda_i, \ \forall i \in A$
        Compute $\mu_m^t \ \forall m \in W, \forall t \in [0, T]$
    end while
    $newColumnsList = \emptyset$ ▷ List that stores new columns with negative reduced cost
    Use multipliers π, λ and μ for pricing out new activity patterns
    Update $newColumnsList$ ▷ Inserts new activity patterns with negative reduced
    Update $\bar{\Omega}$ ▷ Update the pool of activity patterns
    if $newColumnsList = \emptyset$ **then**
        $negRed = false$;
    end if
end while
if $negRed = false$ **then** ▷ Continue with the CG procedure
    Apply CG algorithm (see algorithm 1) solving RMP_1 with the simplex method
end if

As we mentioned before, we explored another alternative way for solving $LSP_1(\lambda)$, besides using the solver. It consists, first on modifying RMP_1', by changing constraint (3.63) as next:

$$\sum_{\omega \in [0,\Omega]} (x_\omega \cdot \beta_i^\omega) \leq t_i \quad \forall i \in A \tag{3.92}$$

Thus, we called the resulting model RMP_1''. Given that the sign of the modified constraint is $\leq$ the linked dual variable (λ) must be non-positive. Nevertheless, we can use exactly the same Lagrangian function as before. The only thing that has to be changed is the sign of the Lagrangian multiplier λ, which now, is defined as non-positive in order to keep the similarity with the associated dual variable. Hence the definition of λ is given by:

$$\lambda_i = min\{0, \lambda_i + (\rho \cdot \phi_i^2)\} \tag{3.93}$$

Now, with this modification, the minimization of $t_N + \sum_{i \in A}(\lambda_i \cdot t_i)$, can be easily solved if $(1 + \sum_{i \in A} \lambda_i) \geq 0)$. This last statement implies, that the sum between the coefficient of t_N, which is equal to one, and the total sum of all the coefficients λ_i of each $t_i \ \forall i \in A$, is larger than zero. Hence, whenever this last condition takes place and considering that $t_N \geq t_i \ \forall i \in A$, we can ensure that minimizing the value of t_N will have a greater impact in the global minimization of $LSP_1(\lambda)$, than minimizing $\sum_{i \in A}(\lambda_i \cdot t_i)$, by means of high t_i values considering that $\lambda_i \leq 0$. Thus, setting t_N equal to the current lower bound, will lead us to the related optimal solution. Subsequently, regarding the values of the remaining starting times, and given that their related coefficients are non-positive, we can minimize the Lagrangian sub-problem by fixing the starting times of each activity (t_i) as equal to the latest starting times that allows us to obtain a t_N equal to the best known lower bound for the current explored node. Nevertheless, when $(1 + \sum_{i \in A} \lambda_i) < 0)$, we cannot fix an specific criteria that allows us to ensure the optimal solution of $LSP_1(\lambda)$. Therefore, when this last case arises, we use the solver for estimating the starting times values.

Finally, it is important to notice, that when $(1 + \sum_{i \in A} \lambda_i), \geq 0)$, we use recursively the Bellman's conditions, for estimating the latest starting times of each activity A_i. These conditions are reminded as follows:

$$\begin{aligned} t_N &= lb \\ t_i &= min_{\forall j \in E_i^+}\{t_j - p_i\} \quad \forall i \in A \end{aligned}$$

Let us recall, that E_i^+ represents the set of successors of activity A_i. The previous definition enhances the obtention of the latest starting times values when setting t_N equal to the best lower bound lb of the current node.

3.4.3 MP_2 based model for combining Lagrangian Relaxation and Column Generation

The second Lagrangian model proposed, is based on the master problem formulation proposed in section 3.2.2. Therefore, we recall the related restricted master problem (RMP_2) as next:

$$Z[RMP_2[\bar{\Omega})] : Min \sum_{\omega\in[0,\Omega]} (c_\omega \cdot x_\omega) \tag{3.94}$$

$$S.t.$$

$$\sum_{\omega\in[0,\bar{\Omega}]} (x_\omega \cdot \alpha_i^\omega) = 1 \quad \forall i \in A \tag{3.95}$$

$$\sum_{\omega\in[0,\bar{\Omega}]} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{3.96}$$

$$\sum_{\omega\in[0,\Omega]} (x_\omega \cdot \delta(t)_{i,j}^\omega) \leq 1 \quad \forall i \in A, \forall j \in E_i^+, \forall t \in \Theta_{i,j} \tag{3.97}$$

$$x_\omega \geq 0 \quad \forall \omega \in [0,\bar{\Omega}] \tag{3.98}$$

Lagrangian Relaxation model

Now, let us associate with constraints (3.95),(3.97) and (3.96) the respective Lagrangian multipliers ($\pi_i, i \in A$), ($\eta_{i,j}^t, i \in A, j \in E_i^+, t \in \Theta_{i,j}$) and ($\mu_m^t, m \in W, t \in T$). The corresponding Lagrangian function can be written as follows:

$$\Gamma(x,\pi,\eta,\mu) = \sum_{\omega\in[0,\Omega]} (c_\omega \cdot x_\omega) + \sum_{i\in A} \pi_i \cdot (1 - \sum_{\omega\in[0,\Omega]} (x_\omega \cdot \alpha_i^\omega)) + \sum_{i\in A} \sum_{j\in E_i^+} \sum_{t\in\Theta_{i,j}} \eta_{i,j}^t \cdot (1 - \sum_{\omega\in[0,\Omega]} (x_\omega \cdot \delta(t)_{i,j}^\omega)) + \sum_{m\in W} \sum_{t\in[0,T]} \mu_m^t \cdot (1 - \sum_{\omega\in[0,\Omega]} (x_\omega \cdot \gamma_{m,t}^\omega)) \tag{3.99}$$

$$\Gamma(x,\pi,\eta,\mu) = \sum_{\omega\in[0,\Omega]} (c_\omega - (\alpha_i^\omega \cdot \pi_i) - \sum_{i\in A} \sum_{j\in E_i^+} \sum_{t\in\Theta_{i,j}} (\delta(t)_{i,j}^\omega \cdot \eta_{i,j}^t) - \sum_{m\in W} \sum_{t\in[0,T]} (\mu_m^t \cdot \gamma_{m,t}^\omega)) \cdot x_\omega + \sum_{i\in A} \pi_i + \sum_{i\in A} \sum_{j\in E_i^+} \sum_{t\in\Theta_{i,j}} \eta_{i,j}^t + \sum_{m\in W} \sum_{t\in[0,T]} \mu_m^t \tag{3.100}$$

For a given distribution (π,η,μ) of Lagrangian multipliers, the associated dual function $L(\pi,\eta,\mu)$ can be computed solving the following Lagrangian sub-problem:

$$Z[LSP(\pi,,\mu,x)] : Min \sum_{\omega\in[0,\Omega]} (c'_\omega \cdot x_\omega) \tag{3.101}$$

$$S.t.$$

$$x_\omega \geq 0 \quad \forall \omega \in [0, \Omega] \tag{3.102}$$

where

$$c'_\omega = c_\omega - \pi_{i(\omega)} - \sum_{i \in A} \sum_{j \in E_i^+} \sum_{t \in \Theta_{i,j}} (\delta(t)^\omega_{i,j} \cdot \eta^t_{i,j}) - \sum_{m \in W} \sum_{t \in [0,T]} (\mu^t_m \cdot \gamma^\omega_{m,t})) \tag{3.103}$$

Hence, we have obviously:

$$\begin{aligned} Z[L(\pi, \eta, \mu)] &= Z[LSP(\pi, \eta, \mu)] + \sum_{i \in A} \pi_i \\ &+ \sum_{i \in A} \sum_{j \in E_i^+} \sum_{t \in \Theta_{i,j}} \eta^t_{i,j} + \sum_{m \in W} \sum_{t \in [0,T]} \mu^t_m \end{aligned} \tag{3.104}$$

Thereafter, the Lagrangian sub-problem ($LSP(\pi, \eta, \mu)$) can be solved to optimality, by setting:

$$\bar{x}_\omega \quad 1 \text{ if } c'_\omega \leq 0, \, 0 \text{ otherwise.}$$

$Z[L(\pi, \eta, \mu)]$ defines a lower bound on the RMP'_2, given that each feasible solution for the original problem is also feasible for the Lagrangian function. Hence, we can obtain the best possible lower bound by solving the Lagrangian dual problem ($LDRMP'_2$):

$$Z[LDP] = Max_{\pi, \eta, \mu} \, L(\pi, \eta, \mu) \tag{3.105}$$

Thereafter, as was done for the first proposed Lagrangian model (see section 3.4.2) we use the subgradient procedure for estimating the values of the Lagrangian multipliers (π, η, μ). Hence, after solving $LSP(\pi, \eta, \mu)$ we obtain the column assignment vector $\bar{x}$. Consequently, we can define a subgradient for $L(\pi, \eta, \mu)$ as follows:

$$\phi_i = 1 - \sum_{\omega \in [0,\Omega]} (x_\omega \cdot \alpha^\omega_i) \;\; \forall i \in A \tag{3.106}$$

$$\vartheta^t_{i,j} = 1 - \sum_{\omega \in [0,\Omega]} (x_\omega \cdot \delta(t)^\omega_{i,j}) \quad \forall i \in A, \forall j \in E_i^+, \forall t \in \Theta_{i,j} \tag{3.107}$$

$$\varphi^t_m = 1 - \sum_{\omega \in [0,\Omega]} (x_\omega \cdot \gamma^\omega_{m,t}) \;\; \forall m \in W \forall t \in [0, T] \tag{3.108}$$

Now, given an upper bound UB for the makespan and a size parameter θ, we can update the current Lagrangian multipliers by:

$$\pi_i = \pi_i + (\rho \cdot \phi_i) \quad \forall i \in A \tag{3.109}$$

$$\eta^t_{i,j} = min\{0, \eta^t_{i,j} + (\rho \cdot \vartheta^t_{i,j})\} \quad \forall i \in A, \forall j \in E_i^+, \forall t \in \Theta_{i,j} \tag{3.110}$$

$$\mu^t_m = min\{0, \mu^t_m + (\rho \cdot \varphi^t_m)\} \quad \forall m \in W \forall t \in [0, T] \tag{3.111}$$

Thereafter, we remind that the step size ρ is defined as next:

$$\rho = \frac{(UB - L(\pi, \eta, \mu))}{Norm} \tag{3.112}$$

where $Norm$ is given by:

$$Norm = \sum_{i \in A} (\phi_i)^2 + \sum_{i \in A} \sum_{j \in E_i^+} \sum_{t \in \Theta_{i,j}} (\vartheta_{i,j}^t)^2 + \sum_{m \in W} \sum_{t \in [0,T]} (\varphi_m^t)^2 \tag{3.113}$$

As was justified in section 3.3.2 we start the subgradient procedure by setting $\theta = 2$. At the end of each iteration we update the parameter θ with a systematic geometric revision: $\theta = \kappa \cdot \theta$. Normally the second parameter κ ranges between 0,87 and 0,9995, depending on the targeting problem.

Overall, the general idea is to update the Lagrangian multipliers during a limited number of iterations ϵ. Thereafter, we use the last updated multipliers as an estimation of the dual multipliers for pricing out new columns. Note that Lagrangian multipliers η and μ are defined as non-positive (see equations (3.110) and (3.111) respectively), in order to keep the same structure of the non-positive dual variables linked to the disjunction and precedence relations constraints (see equations (3.97) and (3.96)) of the RMP_2'. The remaining Lagrangian multiplier (π) is defined as free of sign to keep the similarity with the related dual variable linked to constraints (3.95).

Despites the sign definition of the Lagrangian multipliers, it can be seen in the structure of the Lagrangian function (3.99) that the relaxed constraints are correctly penalized.

Thereafter, when there are no more columns with negative reduced cost by using the Lagrangian multipliers, we perform ψ iterations solving the RMP with the simplex method by using the solver for obtaining the values of the dual variables for pricing out new columns. Hence, if after ψ iterations there are still columns with negative reduced costs, we go back to the Lagrangian procedure, otherwise we stop. The whole procedure is summarized by the algorithm 3.

3.5 Computational Results

Computational experiments were performed using the solver Gurobi OptimizerVersion 4.5. As was already illustrated in section 2.10, we selected a subset of the available instances for the MSPSP [93] according to their size in terms of number of activities, skills and number of workers. In general terms, the computational results shown in this section corresponds to instances which consider between: 20 and 62 activities, 2 and 15 skills, and 2 and 19 workers. We show results for 271 instances, which are divided in three groups:

Algorithm 3 General algorithm for combining CG with Lagrangian Relaxation considering the MP_2 formulation

Input: $\bar{\Omega}$ Current pool of activity patterns;

A Set of activities;

W Set of workers;

T Upper bound for the planning horizon (makespan).

Output: π, η, μ Lagrangian multipliers

$negRed = true$ ▷ It is set to false if there is not at least one possible new column with a negative reduced cost
while $negRed = true$ **do**
  $Iter = 0$ ▷ Initialize the iterator for the subgradient procedure
  $\theta = 2$ ▷ Initialize the step size parameter
  Set multipliers π, η, μ equal to zero ▷ Initialize the lagrangian multipliers
  while $iter \leq \epsilon$ **do** ▷ Starts the subgradient procedure loop
    Solve $LSP(\pi, \eta, \mu) \rightarrow$ optimal solution $(\bar{x}_\omega, \forall \omega \in [0, \Omega])$
    Compute $Z(L(\pi, \eta, \mu))$
    Compute $\phi_i, \ \forall i \in A$ ▷ Compute subgradient
    Compute $\vartheta^t_{i,j}, \ \forall i \in A, \forall j \in E^+_i, \forall t \in \Theta_{i,j}$
    Compute $\varphi^t_m \ \forall m \in W, \forall t \in [0, T]$
    Compute $\pi_i, \ \forall i \in A$ ▷ Update Lagrangian Multipliers
    Compute $\eta^t_{i,j}, \ \forall i \in A, \forall j \in E^+_i, \forall t \in \Theta_{i,j}$
    Compute $\mu^t_m \ \forall m \in W, \forall t \in [0, T]$
  end while
  $newColumnsList = \emptyset$ ▷ List that stores new columns with negative reduced cost
  Use multipliers π, η and μ for pricing out new activity patterns
  Update $newColumnsList$ ▷ Inserts new activity patterns with negative reduced
  Update $\bar{\Omega}$ ▷ Update the pool of activity patterns
  if $newColumnsList = \emptyset$ **then**
    $negRed = false$;
  end if
end while
$simplexIter = 0$ ▷ Initialize the iterator for the simplex iterations
while $negRed = false$ and $simplexIter \leq \psi$ **do**
  Apply CG algorithm, solving RMP_2 with the simplex method (see algorithm 1)
end while
if $negRed = false$ **then**
  Go back to step 2;
end if

– Group 1: We studied 110 instances from this group, considering: between 20 and 51 activities, between 2 and 8 skills, and between 5 and 14 workers.
– Group 2: In this chapter we include the results for 71 instances. Regarding this group of instances, we include results for instances which consider between: 32 and 62 activities, 9 and 15 skills, and 5 and 19 workers.
– Group 3: In this chapter we studied 90 instances which considers between: 22, and 32 activities, 3 and 12 skills, and 4 and 15 workers.

Thereafter, in table 3.1 we compare the linear relaxations obtained with each of the time indexed models introduced in the previous chapter (TIM, TIMWS, MTIMWS), against the resulting lower bound after applying Column Generation. Initially, we evaluated the CG approach based on the RMP_0 (see section 3.2.1) and using the simplex method for solving the linear program (LP). For notation purposes, we refer to this last approach as CG_1. Moreover, at first, we introduce the average deviation between the lower bound obtained with each evaluated model against the best known lower bounds (BLB) obtained by Bellenguez-Morineau and Néron [16]. Notice that deviations were calculated by: $(LB - BLB)/BLB$), where LB represents the Column Generation lower bound. Subsequently, we also compare the average computational times required by each tested model for obtaining their respective lower bound.

		Group of instances		
		Group 1	**Group 2**	**Group 3**
Average deviation against BLB	CG_1	-26,2%	-5,42%	-17,20%
	TIM	-34,98%	-7,08%	-25,09%
	TIMWS	-36,87%	-7,12%	-25,21%
	MTIMWS	-37,57%	-6,98%	-24,41%
Average CPU time (sec)	CG_1	13,90	6,77	9,54
	TIM	1,07	1,10	0,42
	TIMWS	13,78	10,19	3,25
	MTIWS	13,22	9,54	2,5

Table 3.1: Linear relaxations comparison between CG_1 and the time indexed models

Results shown in table 3.1 allows us to state that for each of the tested group of instances, the CG_1 is able to reach a stronger a linear relaxation than the ones obtained by each of the time indexed models. Furthermore, we can also notice that for all the tested models, the linear relaxation seems to be more close to the BLB for the group 2 of tested instances, reaching an average deviation close to -5%, when applying CG_1, and around -7% with the time indexed models. In the other hand, for the remaining groups of instances the increase of the linear relaxation when using CG_1 it was a little bit more significant, close to the 10%. In addition, we can distinguish that in the TIM, we spend much less time solving the LP. Nevertheless, CG_1, presents an acceptable performance in terms of computational times, considering the improvement obtained in terms of the quality of the obtained linear relaxations.

Now, at next we compare the performance of the different CG approaches introduced in the previous sections. In one hand, besides CG_1, we have also $CGLR_1$, which is still based on the resolution of RMP_1, but the LP is solved with the combined Lagrangian relaxation and Column Generation approach proposed in section 3.4.2. In the other hand, we have CG_2 and $CGLR_2$, which corresponds to utilization of RMP_2, which is based on the master problem reformulation introduced in section 3.2.2. Hence, in CG_2 we use only the simplex method for solving the LP, while in $CGLR_2$ we combine the use of Lagrangian relaxation and the simplex method for solving the LP, as it was proposed in section 3.4.3 for solving the restricted master problem. Therefore, in table 3.2 we compare the results of the four CG approaches in terms of the average deviation against BLB, computational times and average number of generated columns. It is important to mention that, for enforcing the value of the lower bound of obtained with Column Generation, we calculated a preliminary lower bound based on the principle of the stable set. This bound was proposed by [91] and adapted to the MSPSP by Bellenguez-Morineau and Néron [16]. In the next chapter we will explain with more details this last mentioned bound.

		Group of instances		
		Group 1	**Group 2**	**Group 3**
Average deviation against BLB	CG_1	-10,80%	-4,96%	-6,17%
	$CGLR_1$	-10,80%	-4,96%	-6,17%
	CG_2	-4,31%	-3,20%	-3,30%
	$CGLR_2$	-4,31%	-3,20%	-3,30%
Average CPU time (sec)	CG_1	11,37	9,88	5,10
	$CGLR_1$	7,07	7,97	3,42
	CG_2	216,58	119,82	95,98
	$CGLR_2$	193,15	99,54	87,37
Average number of generated columns	CG_1	738,05	1181,65	1817,77
	$CGLR_1$	1181,19	1645,07	2571,8
	CG_2	3498,45	3841,40	5814,11
	$CGLR_2$	9336,71	10370,57	11458,41

Table 3.2: Comparison between CG approaches proposed

Thereby, results introduced in table 3.2 show that the obtained lower bounds are much more closer to BLB, due in part to the inclusion of the stable set lower bound. The impact of using this last bound can be noticed when comparing the results related to CG_1 shown in table 3.1 and the ones shown in 3.2. Hence, we can find that the average deviation against the BLB increased from -26,20% to -10,80% for the group 1 of tested instances; from -5,42% to -4,96% for the group 2 of tested instances and from -17,20% to -6,17% for the group 3 of tested instances. Thereafter, we can also see that CG_2 and $CGLR_2$ leads to better lower bounds than CG_1 and $CGLR_1$, implying that the resolution of RMP_2 indeed enhances a stronger linear relaxation than RMP_1. Nevertheless, the CG approaches based on the resolution of RMP_2 required a considerably larger amount of computational time until obtaining a lower bound. Although, we can indeed notice that the utilization of

Lagrangian relaxation allowed us to accelerate the resolution of the respective restricted master problems. Moreover, we can distinguish that the CG approaches based on the resolution of RMP_2 generates more columns (i.e activity patterns) per instance than CG_1 and $CGLR_1$. Additionally, we can also see that the number of generated columns also increases when using the approach that combines Lagrangian relaxation and the simplex method for solving the LP.

3.6 Conclusion

In this chapter we studied and applied Column Generation as an alternative for obtaining strong linear relaxations for the set of instances evaluated. Therefore, we can conclude that CG allowed us to reach better linear relaxations than the ones obtained by the time indexed models introduced in the previous section. Thereafter, we compared different Column Generation approaches. The main differences between the proposed CG approaches relies in the MP formulation and the methods used for solving the related LP. Hence, we were able to conclude that RMP_2 allowed us obtain better linear relaxations than ones obtained when using RMP_1. Nevertheless, we could argue that the improvement in the quality of the resulting lower bound after applying the RMP_2 based CG approaches is not that significative, given the considerable increase of the computational time invested in the resolution of each tested instance. Additionally, we were also able to distinguish that the utilization of the simplex method along with the proposed Lagrangian relaxation models allowed us to decrease the computational time consumed in the resolution of each tested instance. Nevertheless, it is important to mention that there are some new perspectives that could be considered regarding to the utilization of CG for solving the MSPSP. On one hand the generation of certain additional inequalities (cuts) could lead to a stronger linear relaxation when solving the restricted master problem. In addition, regarding the particular performance of the RMP_2 based CG approaches it could be interesting to take into account certain measures for accelerating the convergence, which could lead to a decrease of the related computational times. Finally, we have to mention, that other decomposition approaches could be explored, such as decomposing on the resources (workers).

4

Column Generation utilization: Branch and price and Recovering Beam Search

In this chapter, we present two tree searching methods. In the first one we perform an exhaustive search (exact), while in the second one we explore only a certain number of nodes (heuristic). Both proposed methods are based on the utilization of the Column Generation (CG) approach explained in the previous chapter for estimating the lower bound of a given node. At first we introduce the exact approach, which consists in the implementation of a Branch and Bound procedure, which is commonly known as Branch and Price (B&P), given the utilization of CG for calculating the lower bound in each evaluated node. In addition, we also propose several methods for estimating the upper bound for each evaluated node. Subsequently, we also explore different branching squemes and strategies for pruning the search tree. Thereby, we introduce the obtained results. Furthermore, we introduce a Recovering Beam Search approach, which is a known heuristic search tree procedure. In this last method, we exploit the structure of the previously mentioned Branch and Price, but we aim on expanding a certain number of nodes according to a given criteria. Finally, we show the computational experiments and respective results.

4.1 Branch and Price (B&P)

4.1.1 Introduction

As we already mentioned, B&P combines the utilization of CG with a Branch and Bound procedure. Therefore, it is important to recall to the reader that the application of Column Generation implies the iterative resolution of a linear program (RMP). Now, it is important to notice that the solution to the RMP fulfills all constraints of a master problem except for the integrality constraints. In the case where the linear relaxation of the master problem does not lead to an integral optimal solution, a branching strategy must be applied to lead the solution into integrality. Different branching strategies have been developed, that are

appropriated in Branch-and-Price algorithms [109]. Usually, branching is done in the original variables or in the variables of the sub-problem. In most cases, a branching 1-0 in the variables of the master problem is not advised since it may destroy the structure of the sub-problem or it could provide an unbalanced Branch-and-Price tree, between other reasons as presented in [10]. According to the structure of the MSPSP and the proposed activity oriented decomposition approach we explored two branching strategies. The first one consists of reducing the time-windows of activities and the second one is based on a chronological approach [2].

4.1.2 Branching strategies

Reducing time-windows of activities

This branching strategy was also implemented in a Branch-and-Bound procedure proposed by Bellenguez-Morineau and Néron [15] for solving the MSPSP. It is based on reducing the time-window $(es_i; ls_i)$ of a given activity by means of a dichotomic search approach, originally inspired from Carlier and Latapie [26]. Hence, according to a certain criteria an activity A_i is selected. Thereafter, the time-window of the starting time of A_i is divided in half obtaining two new nodes, corresponding to two new disjoint time-windows for the selected activity. This branching strategy allows us to define two disjointed subset of schedules for the two generated nodes. Afterwards, we propagate on the precedence graph, and then, we update the time-windows and subsequently the initial pool of columns of the RMP for the later execution of the CG procedure. Next figure shows an example of how the two new nodes are generated, after applying CG on a given node, and selecting an activity A_i to branch on.

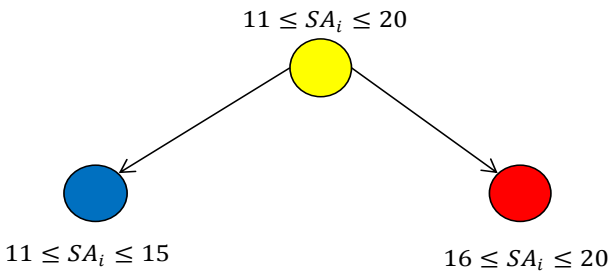

Figure 4.1: Example of the dichotomic time-windows branching strategy.

Since, the criterion to select the activity A_i has an important impact in the performance of the Branch-and-Price algorithm, several criteria were considered:

1. Activity with a lower number of columns. It selects the activity with the lowest number of related columns given the pool of activity patterns of the current RMP of a given node.
2. Activity with a higher number of columns. It selects the activity with the highest number of related columns given the pool of activity patterns of the current RMP of a given node.

3. Activity with a larger time-window size. Selects the activity with the highest value of $ls_i - es_i$, which represents the size of the time-window of an activity A_i.
4. Activity with a smaller time-window size. It selects the activity with the smallest value of $ls_i - es_i$.
5. Activity with a higher number of resource conflicts. It selects the activity with the highest number of resource conflicts with other activities. These conflicts appear when there is a couple of activities that, according to their current time-windows can be simultaneously executed. Nevertheless, their parallel execution is not feasible due to the capacity of the available resources.
6. Activity that leads to a larger reduction in the time-windows. It selects the activity that, after doing a preliminary propagation, leads to a larger reduction in the size of the time-windows of all the activities.

With the purpose of defining a single selection criteria, we performed some preliminary tests on a predefined subset of instances (see table 4.1). Obtained results show that criteria 5 and 6 are the ones that leads the Branch-and-Price to a better performance in terms of the number of instances with optimal solutions within an imposed time limit of thirty minutes. Thereafter, we fix a single strategy that considers criteria 5 and 6 as primary and secondary rules respectively, improving the results obtained with each criterion separately.

Chronological branching strategy

In this branching strategy, a new node consists of adding at least one activity A_j to a partial schedule. Given that there are several types of chronological branching schemes [2], we initially explored the option of adding one activity to a partial schedule. In this branching scheme, initially we define a set of eligible activities EL composed by all the available activities whose predecessors have already been scheduled. Therefore, one node is created for each activity A_j in EL, adding such an activity as soon as possible, at a time point t that ensures the fulfillment of the precedence and resource constraints. Notice, that this last procedure considers all the available activities for building new partial schedules. Furthermore, with the purpose of generating a new partial schedule we have to ensure a feasible assignment of workers for the activity $A_j \in EL$ that could be included in the current partial schedule (PS). Hence, we represent the resource assignment of the activity A_j as a min-cost max-flow problem, as we did for the resolution of the SP (see section 3.2.4). In this case, we try to fulfill the skills requirements of A_j as it is shown in figure 4.2 considering the subset of available workers at time t ($W_a(t)$). Such a group of workers, is defined according to the resource assignment stated at a previous level for all the activities included in PS, which were already scheduled.

Additionally, in this new graph F_a we do not take into account an assignment cost for each worker. If the maximum flow along the graph F_a is less than the total number of workers required by activity A_j we can state that it is not possible to find a feasible assignment at time t.

Moreover, for ensuring that the new candidate activity A_j cannot be executed at time t, we try to reassign all the workers involved in the execution of all the activities in PS.

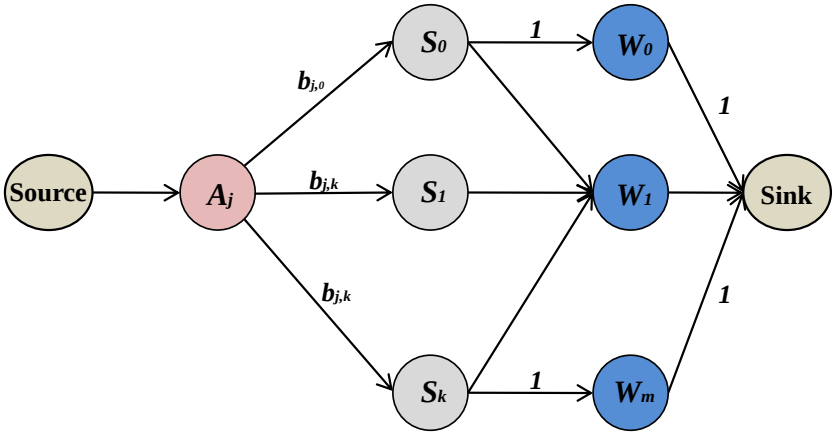

Figure 4.2: Graph F_a skills assignment for the candidate activity A_j without considering an assignment cost for each worker.

Subsequently, we keep the previously defined starting times of such activities and assume that A_j starts at time t, thereafter, we solve an assignment model (AM($\Omega^{'}$)) which is described as follows:

$$\sum_{\omega \in [0,\Omega']} (x_\omega \cdot \alpha_i^\omega) = 1 \quad \forall i \in (PS \bigcup A_j) \tag{4.1}$$

$$\sum_{\omega \in [0,\Omega']} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{4.2}$$

$$x_\omega \in \{0,1\} \quad \forall \omega \ \in [0,\Omega'] \tag{4.3}$$

This last model is based on the integer linear program proposed for the master problem in section 3.2.1. Since the starting times of each activity are fixed, we only have to consider the next set of constraints: (i) constraint set (4.1), which states that only a unique activity pattern can be assigned to any task A_i; and (ii) constraint set (4.2) which ensures that any operator can carry out at most one activity at a given time.

Before running the previous model we generate all the possible combinations of workers that could be assigned to each activity. Additionally, we denote Ω' as the set of all feasible activity patterns generated for each activity that belongs to a new partial schedule. All the possible combinations of workers that could be assigned to each activity are obtained by means of the already mentioned min-cost max-flow algorithm. This implies that we have to solve the min-cost max-flow problem represented in figure 4.2 but considering only the arcs related to a single activity A_i and assuming that all the workers in W are available. Thus, we generate a new combination of workers for each activity A_i until there are no more combinations of workers that could lead to a feasible resource assignment.

This proposed approach for obtaining a new partial schedule avoids branching on the possible set of workers that could be assigned to a given activity A_i at a time t, which

could lead to a big increase in the number of generated nodes. Computational results show that the procedure for finding a feasible assignment of workers for a new partial schedule (node) takes in average less than 5% of the total computational time. Additionally, the time invested in generating all the combinations of workers for each activity takes only the 3,27% of the total computational time.

4.1.3 Lower and upper bounds

For enforcing the convergence of a Branch-and-Bound type procedure it is important to define good strategies that enhance strong lower and upper bounds. Hence, the approaches that we use for estimating such bounds, are described as follows:

Lower Bounds

When calculating the lower bound of a given node we take into account the next three strategies:

Time-windows propagation bound: Considering that generating a new node is related to an update of the time-window of a given activity, we propagate on the precedence graph, which lead us to a preliminary lower bound for the makespan of the current node.

Stable set bound: After updating the time-windows of a current node we estimate a new lower bound, which is based on the compatibility graph (G_c) [91] and adapted to the MSPSP by Bellenguez-Morineau and Néron [16]. To construct G_c, it is necessary to do a pre-processing procedure to know which pairs of activities may be in progress at the same time: they must not have precedence relation, their time-windows must overlap and their execution in parallel must not lead to a resource conflict. Hence, the compatibility graph is built by creating a node per activity, with a weight equal to its processing time. Two activities are linked if they may be in progress at the same time. In this graph, a maximum weighted stable set is computed. This $\mathcal{NP}$-Hard problem is solved efficiently with an ILP formulation, which is stated as follows:

$$Max \sum_{i \in G_c} (u_i \cdot p_i) \tag{4.4}$$

$$S.t.$$

$$u_i + u_j \leq 1 \quad \forall (A_i, A_j) \in G_c \tag{4.5}$$

$$u_i \in \{0, 1\} \quad \forall i \in G_c \tag{4.6}$$

The only decision variable is u_i which takes the value of 1 if A_i it is included in the stable set or it takes the value of 0 otherwise. The stable set corresponds to a subset of activities that cannot be in progress at the same time. The resulting total weight ($\sum_{i \in G_c}(u_i \cdot p_i)$) after computing the stable set gives a destructive lower bound.

CG bound: After applying the two previous procedures, we enforce the best lower bound calculated so far for the current node by using the CG approach explained in the previous sections. Hence, we can obtain a stronger bound that takes into account the disjunction and precedence constraints included in the RMP. The stable set bound estimation takes in average less than 0,016 seconds per node, and it covers less than the 4% of the total computational time. For further details, obtained results will be discussed later on in section 4.1.5.

Upper Bounds

On the other hand, for computing the UB of a given node, we explore also three approaches:

ILP based upper bound (ILPUB): This approach consists in solving the integer linear program proposed for the MP in section 3.2.1, where the integrality constraint (7.21) (x_c is defined as integer) is activated. The solution of this ILP allows us to obtain a feasible upper bound for the makespan. The execution of the IP solver is done until a feasible solution is founded within an imposed time limit of 10 seconds. Such an approach helps to accelerate the process of finding integer solutions and it might lead to the optimal solution for the current pool of columns. Other time limits were tested (5, 30, 45 and 60 seconds) without increasing the number of achieved optimal solutions or improving the computational times.

Reduced cost priority list upper bound (RCPL): This upper bound is obtained by means of a heuristic in which the activities are scheduled iteratively. Thus, we build a priority list EL that includes a subset of activities whose predecessors have already been scheduled. Then, the starting times of the activities in EL are defined as equal to latest termination time of their respective predecessors. Thereafter, we give priority to scheduling the activity with a lower starting time. Now, let us suppose that an eligible activity A_i, starts at time t and the respective subset of available workers is defined by $W_a(t)$. Given that this heuristic is applied after solving a RMP, we can calculate the assignment cost of each available worker ($\sigma_m(t)$) for finding a feasible assignment of workers with the minimum total assignment cost $r^2_{i,t}$, as we did for solving the SP in section 3.2.4. Whenever there is more than one activity with the same starting time t, we give priority to the activity with a lower total reduced cost $r_{i,t}$ (see equation (7.29)). If the resource assignment of A_i at time t is not feasible, t is increased until there is at least one additional worker to the ones in $W_a(t)$ that is released.

For enforcing the description of the proposed heuristic, let us consider a project based on the precedence graph represented in figure 3.1. We assume, that the set of workers W is composed by only two workers W_0 and W_1 that masters one single skill S_0. Additionally, we state that activities A_1 and A_2 require two workers, and that A_3 and A_4 require one worker. Based on the related precedence graph we can state that initially $EL = \{A_1, A_2\}$ and that both A_1 and A_2 can start at time $t = 0$. Therefore, supposing that $r_{1,0} < r_{2,0}$, A_1 is scheduled at first, which implies the assignment of W_0 and W_1. Thus, we have to update the set of eligible activities, obtaining that $EL = \{A_2, A_3\}$, then, since $A_2 \succ A_0$

and $A_3 \succ A_1$, we, have, that A_2 could start at $t = 0$ ($p_0 = 0$) and A_3 at $t = 2$ ($p_1 = 2$). Subsequently, we try to schedule A_2 at next, but, since there are no workers available at $t = 0$, we have to delay the execution of A_2 until time $t = 2$, in which both W_0 and W_1 are released after executing A_1. Hence, the two current eligible activities could start at time $t = 2$. Thereafter, we compare the total reduced cost of each eligible activity ($r_{2,2}$ and $r_{3,2}$) for selecting the next activity that will be added to the current partial schedule. Afterwards, we continue with the same procedure until scheduling all the activities of the project and obtaining a feasible upper bound.

When applying CG in a given node, we use this heuristic after solving each RMP. Therefore, we can get different upper bounds for the same node according to the dual information (simplex multipliers) obtained every time a new RMP is solved.

Reinforced reduced cost priority list upper bound (ERCPL): This upper bound reinforces the RCPL heuristic. We follow the same procedure for selecting an activity A_i from a list EL. The only difference is that every time the execution of a given activity A_i at time t is not feasible due to the current assignment of workers, instead of delaying its starting time, we search for a re-assignment of workers that allows the execution of A_i at time t. Hence, we use the assignment model(AM(Ω')) described in the previous section. For solving such a model, we fix the starting time of A_i at time t and the starting times of the activities already scheduled. This procedure is more time consuming than the previous one but it might lead to a better UB at a lower depth of the search tree. Hence, this heuristic is only applied as long as the starting time t of an activity A_i is less or equal than its latest starting time ($t \leq ls_i$), which implies that the current best UB might be improved.

Finally, we integrate these three approaches for obtaining the upper bound of a given node. Therefore, when applying CG we use the RCPL heuristic after solving the RMP until there are no more columns left to be added. Hence, the best upper bound found with this last heuristic is stored. Then, whenever the number of activity patterns (i.e columns) generated is lower than a fixed limit value of columns, we use ILPUB. Finally, if we don't obtain a solution before the imposed time limit of 10 seconds we apply the ERCPL heuristic. Consequently, we set the best UB found for the examined node. For fixing the limit number of columns for applying ILPUB, we performed some preliminary tests in a certain subset of instances, obtaining that in average a feasible solution was found within 10 seconds whenever the number of activity patterns were lower than 1000. This limit is iteratively decreased every time a feasible solution is not found before the imposed time limit for the execution of ILPUB.

Furthermore, after applying one of the two described branching schemes, the corresponding child nodes are generated as long as the lower bound reached by the column generation procedure is less than the best feasible upper bound. Thus, the branching procedure is repeated until the best lower bound is equal to the best obtained upper bound. Such a lower bound is updated while new nodes are generated.

4.1.4 Reducing the search tree size

Different strategies were developed to prune nodes that could lead to infeasible solutions.

Stable set bound: This strategy consists in comparing the stable set bound explained previously with the best known feasible upper bound. Hence, we can prune the current node whenever such a lower bound is greater than the best UB.

Infeasible parallel execution: After generating a new node, and updating the time-windows of all the activities, we check if the execution of a couple of activities will have a mandatory overlap. Hence, if the parallel execution of such activities leads to a resource conflict, the execution of the project with the current time-windows is infeasible, thus we can prune the generated node.

Precedence relation adjustment: After generating a new node, we perform another pre-processing procedure that identifies the pairs of activities that in principle could be done in parallel, but due to their time-windows and resources capacity, one must be performed after the other one (implying a new precedence relation between such activities). Then, we can enforce the RMP by updating the directed graph (G) of the project with the new generated precedence relations. Notice these new precedence relations enhances a new propagation on the time-windows of the activities.

Infeasibility penalization with the inclusion of slack variables in the RMP: A given node is also pruned if the lower bound obtained with the CG procedure is larger or equal than the current upper bound. A high value of such a lower bound takes place normally, when the current solution for the RMP is infeasible. This infeasibility is detected by means of the slack variables included in the RMP (see section 3.2.1), which are penalized with a high cost in the objective function.

Moreover, the steps in which we applied these strategies are described as follows:

- Update the time-windows of each activity according to the current precedence relations.
- Check which couple of activities have to be done in parallel (mandatory overlap).
- Apply the “Infeasible parallel execution" strategy to state if the current node is pruned.
- Verify which couple of activities might be done in parallel according to the current time-windows.
- Check if we could apply the “Precedence relation adjustment" rule.
- Update the compatibility graph for calculating the stable set bound and compare it with the best current UB.
- Apply the “Infeasibility penalization with the inclusion of slack variables in the RMP" rule after applying CG in the remaining nodes.

Notice, that the lower bound obtained by applying CG considers the time-windows, precedence relations and workers disjunction constraints which implies that we can either improve the stable set bound or identify an infeasibility that was not detected with the previous strategies.

4.1.5 Computational Results

Computational experiments were performed using the solver Gurobi OptimizerVersion 4.5. As was done for the previous solution methods we selected a subset of the available instances for the MSPSP [93] according to their size in terms of number of activities, skills and number of workers. We consider, the same set of instances evaluated in the previous chapter. As a reminder, results shown in this chapter corresponds to instances which consider between: 20 and 62 activities, 2 and 15 skills, and 2 and 19 workers. We show results for 271 instances, thereafter, the considered sizes for each group of tested instances (see section 2.10) are reminded as follows:

- Group 1: We studied 110 instances from this group, considering: between 20 and 51 activities, between 2 and 8 skills, and between 5 and 14 workers.
- Group 2: In this chapter we include the results for 71 instances. Regarding this group of instances, we include results for instances which consider between: 32 and 62 activities, 9 and 15 skills, and 5 and 19 workers.
- Group 3: In this chapter we studied 90 instances which considers between: 22, and 32 activities, 3 and 12 skills, and 4 and 15 workers.

We also tested instances with bigger sizes, but we were not able to obtain optimal solutions beyond the sizes previously mentioned.

Now, before introducing all the obtained results we outline that we applied the approach that combines Lagrangian relaxation and the simplex method for solving the related restricted master problem. Results shown in section 3.5 reflected that using Lagrangian relaxation allowed us to accelerate the resolution of the restricted master problem.

Concerning the time-windows branching strategy (B&PTW), table 4.1 compares the results obtained for a subset of 70 instances, between each one of the tested criteria for selecting an activity to branch on. Such a comparison is done in terms of the number of instances with optimal solutions within a time limit of thirty minutes. Thereafter, the following tables show more detailed results using criteria 5 as primary rule and 6 to break ties, since such an strategy dominates the other ones in terms of the number of optimal solutions founded. This strategy ensured an optimal solution for all the instances for which optimality was also achieved when applying the remaining selection rules.

Moreover, table 4.2 show results that compare the B&PTW, the B&P with the chronological branching scheme (B&PCB) in terms of average computational times, number of optimal solutions founded within the imposed time limit of thirty minutes and the average deviation between the best UB and the final LB achieved by means of the applied B&P approaches. Deviations were calculated by $(UB - LB)/LB$.

Activity selection rules for the B&PTW	Number optimal solutions
(1) Activity with a lower # of columns	34
(2) Activity with a higher # of columns	25
(3) Activity with a larger time-window size	32
(4) Activity with a smaller time-window size	38
(5) Activity with a higher # of resource conflicts	47
(6) Activity that leads to a larger reduction in the time-windows	41
(7) Criteria (5) as primary rule and (6) as secondary rule	50

Table 4.1: Branching strategies performance

		Group of instances		
		Group 1	**Group 2**	**Group 3**
Average CPU time (sec)	B&PTW	1081,07	647,17	1016,34
	B&PCB	1151,46	714,75	1195,86
Number optimal solutions	B&PTW	48	47	46
	B&PCB	42	44	40
Average deviation between final UB and LB	B&PTW	16,86%	5,38%	8,97%
	B&PCB	17,03%	5,2%	8,7%

Table 4.2: Performance comparison between the proposed B&P approaches

Overall, these results show that the B&PTW is able to reach optimal values for 141 (48, 47 and 46 for each group of tested instances) out of the 271 tested instances. On the other hand, the B&PCB solved 126 instances to optimality within thirty minutes. Additionally, the B&PTW outperforms the B&PCB also in terms of the average computational times per instance, nevertheless both approaches present a similar behavior in terms of the deviation between the final upper and lower bounds.

Furthermore, it is important to state that the magnitude of the CPU times values shown in table 4.2 are affected by the thirty minutes spent in the instances in which was not possible to find the optimal solution. Thus table 4.3 shows the computational times spent for solving the 141 and 126 instances in which the optimal solution was reached by each B&P approach respectively.

Additionally, this last table also shows that from the 271 tested instances the proposed B&PTW and B&PCB were able to find optimal solutions for 72 and 57 instances for which the optimal value was previously unknown given the lower and upper bounds found

		Group of instances		
		Group 1	**Group 2**	**Group 3**
Average CPU time (sec)	B&PTW	148,95	57,09	283,02
	B&PCB	166,55	57,55	218,21
Number of new optimal solutions	B&PTW	16	11	45
	B&PCB	10	8	39

Table 4.3: Performance comparison between the proposed B&P approaches for instances solve to optimality

in previous work by Bellenguez-Morineau and Néron [16, 17].

Furthermore, given that the dichotomic time branching strategy is the one that obtained better results, next tables are related to the performance of the B&PTW. Table 4.4 compares the average percentage of the total computational time spent on both the master problem and sub-problem, on finding the upper bound and on obtaining the stable set lower bound. This table also includes the percentage of the total computational time invested in the definition of all the possible combinations of workers that can be assigned to each activity. Additionally, in this table we also consider the percentage of time consumed for finding a feasible assignment of workers whenever a new node (i.e partial schedule) is generated.

	Group of instances		
	Group 1	**Group 2**	**Group 3**
MP% of total CPU time	54,17%	33,22%	50,74%
SP% of total CPU time	8,46%	3,56%	6,51%
UB% of total CPU time	30,49%	40,42%	32,83%
Stable set bound % of total CPU time	1,74%	3,56%	2,49%

Table 4.4: B&PTW computational time performance measures

Overall, results show that the MP takes close to the 50% of the total computational time, while the SP takes arround the 9%, the estimation of the upper bound takes near to the 34%, while the stable set bound calculation covers close to the 3% of the total CPU time.

The remaining percentage is related to other procedures such as the propagation on the time-windows, infeasibility detection, etc. It is important to state that around the 21% of the computational time spent in the calculation of the upper bound is due to the time invested in the RCPL heuristic (see section 4.1.3) which is applied after solving each RMP generated in the CG procedure. The other techniques considered for finding an UB, ERCP and the ILPUB, cover in average, for all the tested instances, the 8,05% and the 3,96% of the CPU time respectively. These last two procedures are applied at most once in each node according to certain conditions, while the ERCP can be executed several times in a given node.

Finally, next table compares the performance of the strategies used to prune nodes. Obtained results show that more nodes were expanded in the instances that belongs to Group 3 within the imposed time limit of thirty minutes. Additionally, we can establish that the strategy that allows us to prune more nodes is the inclusion of slack variables in the RMP. In addition we can also notice that the infeasible parallel execution pruning strategy is able to prune more nodes than the stable set bound strategy.

	Group of instances		
Pruning Strategy	**Group 1**	**Group 2**	**Group 3**
Number of expanded nodes	2531,55	763,79	4891,35
Stable set bound	0,42%	0%	0,62%
Infeasible parallel execution	2,67%	0,17%	7,93%
Inclusion of slack variables in the RMP	58,67%	37,66%	41,44%

Table 4.5: Performance strategies for pruning nodes

4.1.6 Conclusion

In this section, we proposed a Branch-and-Price procedure based on two possible branching strategies. We developed a CG procedure based on an activity and time-oriented decomposition approach. This approach allows us to deal with hard constraints in the sub-problem, such as synchronization of the assigned workers. One possible drawback of considering an activity and time-oriented decomposition approach is that it could lead to have to solve several number of sub-problems per iteration, and then, to spend a large amount of time solving them, nevertheless obtained results show that the time spent on the SP is not significant in comparison to the time consumed in other procedures of the B&PTW and B&PCB. Regarding, the B&PTW, the activities selection criteria, we were able to find a unique strategy that dominates the other ones tested. Overall, we concluded that the the time-widows branching scheme outperforms the chronological branching strategy in terms of both number of optimal solutions and computational times. It is important to mention that new optimal solutions were also found in several instances for which the optimal value was previously unknown. Nevertheless, there are some perspectives that could lead to an improvement of the current performance of the proposed B&P procedures. For instance regarding the chronological branching strategy, there are other alternatives that could be explored in which more than one activity could be added to the partial schedule of a given node. Thereafter, fixing the starting times of several activities (instead for only one) when generating a new node could lead to a strong lower bound when applying CG. Additionally, there are different dominance rules developed in the RCPSP field which could also be applied, with the purpose of reducing the search tree. Finally, a different decomposition approach when applying CG could required the utilization of different branching strategies that might lead to interesting results.

4.2 Recovering Beam Search

In this section, we introduce a recovering beam search (RBS) approach that exploits the structure of the branch and price procedures discussed in the previous section 4.1. The

recovering beam search (RBS) is an enhancement of the beam search (BS) [96]. The implementation of the RBS implies expanding a given limited number of nodes at each level of the search tree. In addition, such a method aims at recovering from previous wrong decisions by checking if the current partial solutions might be dominated by other partial solutions at the same level of the tree that haven't been developped. Therafter, the resulting solution method is a heuristic that integrates the resolution of different mathematical models for solving instances of bigger sizes than the ones discussed in the previous sections. Computational experiments throwed comparable results to the ones obtained with the tabu search developed by Bellenguez-Morineau and Néron [17].

4.2.1 Beam search (BS)

BS is a heuristic search algorithm for solving combinatorial optimization problems. The classic beam search approach is based on a truncated branch and bound, where at each level of the search tree only the w most promising nodes are selected for further examination and the other nodes are permanently discarded. Beam search is similar to breadth-first search as it progresses level by level through a highly structured search tree. Typically, the value of the parameter w known also as the beam width is fixed before starting the search. Therefore, the agressive pruning enforced by the exclusive selection of a certain number of nodes might provoke the premature elimination of a good node that might lead to the optimal solution. Nevertheless, the selection of a wider beam width allows to control the risk of making wrong branching decisions but at the cost of increasing computational effort and memory [97].

An important issue to consider when applying the BS technique is the definition of a cost function that is used to evaluate which nodes at each level of the search tree are promising or not. Thereafter, there are two BS variants, each one based on a given type of cost function: priority BS and detailed BS [111].

In the priority BS a crude cost function is considered by establishing a priority for each successor node and selects based on those priorities. At the root of the search tree, up to w most promising successors (i.e. those with the highest priorities) are selected, while in each subsequent level only one successor with the highest priority is selected per examined node.

The detailed BS considers a more accurate cost function, which estimates the total-cost of the best path that can be found continuing from the current node. At each level up to w most promising nodes (i.e. those with the lowest total-cost values) are selected regardless of who their parent nodes are.

Furthermore, BS has so far been mainly used in searching trees with a high density of nodes. The first known application of this method was in the context of speech recognition ([86]). Thereafter, scheduling problems became a common target for using the beam search algorithm. For example, Fox [52] used beam search for solving complex scheduling problems. Later, Ow and Smith [97] incorporated a BS procedure in systems designed

for complex job shop environments. In another study, related to a Flexible Manufacturing Mystem (FMS), Chang et al. [27] used beam search as a part of a scheduling algorithm called bottleneck-based beam search. Results indicated that their algorithm outperformed different dispatching rules used for minimizing the makespan. Moreover, an overview of the beam search and its applications to optimization problems can be found in Morton and Pentico [92].

Beam search improvements

In the previous section we implied the existence of two types of cost functions. In one hand, we have that crude cost functions are not expensive in terms of computational performance, but could be inaccurate and may result in discarding good nodes. On the other hand, total cost evaluation functions, are more accurate but require a higher computational effort. Thereafter, there are two ways to improve the BS procedure that combines the use of both crude and accurate cost functions: the filtered beam search (FBS) and the recovering beam search (RBS) [111]. Notice that the aim of these two last methods is to provide a process that enhances a good node evaluation without spending a significative amount of computational time.

Filtered beam search (FBS)

The filtered beam search method uses as a first step a filtering procedure that selects a certain number f of the children of each beam node for a more accurate evaluation. Typically, the parameter f which is known as the filter width, is fixed before starting the search. Usually, a priority cost function is used for defining an initial cost evaluation value for selecting the best f children of a given node. Therefore, the selected nodes are considered for a more accurate evaluation, choosing finally, the best w nodes [95].

Furthermore, the FBS was initially proposed by Ow and Morton [95] as an application to the single machine early/tardy problem. In such a paper authors studied the effects of using different evaluation functions to guide the search and compare the performance of beam search with other heuristics. A filtered beam search application to a Flexible Manufacturing System (FMS) scheduling environment is reported by De and Lee [38] who showed that the solution quality of a FBS algorithm is better than depth-first type heuristics in terms of the average maximum lateness and average flow time measures. The authors also showed that beam search was better than breadth-first type heuristics in terms of number of nodes created during the search. Sabuncuoglu and Karabuk [106] proposed a filtered beam search algorithm for a complex FMS environment. Their results indicated that the beam search based scheduling algorithm exploits flexibilities inherent in FMS more effectively than other methods. Thereafter, Sabuncuoglu and Bayiz [105] applied a FBS for the job Shop Scheduling problem. Franck and Neuman [53] evaluated different truncations of a branch and bound algorithm for solving the RCPSP. Among such methods best results were obtained by means of a filtered beam search.

Algorithm 4 Recovering beam search algorithm

Input: w Beam width;

u Search tree depth: $u = |A| - 2$ ▷ The search tree depth is set as equal to the number of activities to schedule, given that A_0 and A_N are dummy activities

1: Initialization:
   - Initialize the current search tree level: $l = 0$
   - Define initial current partial solution: σ_1 = root node ▷ typically no variables has been fixed
   - Set vector of current partial solution: $S = \sigma_1$

2: **for** $(k = 1, k \leq min\{|S|, w\}k++)$ **do**
   - Branch σ_k for generating all the corresponding children.
   - Prune all the nodes that are eliminated by the filtering procedure.

3: **end for**
4: Set S as empty.
5: **for** each remaining child node **do**
   - Calculate a lower bound LB and an upper bound UB on the optimal solution value of that node.
   - Compute the cost evaluation function: $V = (1 - \alpha) \cdot LB + \alpha \cdot UB$.

6: **end for**
7: Add the remaining nodes into a set B.
8: Sort B in non-decreasing order of their evaluation function.
9: **while** $(|S| < w$ and $k \leq |B|)$ **do**
   - Recovering step: search for a partial solution $\bar{\sigma_k}$ that dominates σ_k.
   - If σ_k is found set $\sigma_k = \bar{\sigma_k}$.
   - If $\sigma_k \notin S$ add σ_k to S.
   - $k = k + 1$.

10: **end while**
11: $l = l + 1$. If $l < u$ go to step 2, else stop.

Recovering beam search

The recovering beam search, also uses a filtering phase, and then, the retained nodes are evaluated according to a total cost evaluation function. Subsequently, a recovering step is executed, with the aim of searching an improved partial solution that dominates the current one having the same level of the search tree. This step allows to recover from previous wrong branching decision where a promissing node could have been pruned. Finally, the best w are retained for further branching. The main steps of the RBS procedure are summarized by the algorithm 4.

Recovering step

The main feature of the RBS is the recovering step which is applied at each level of the search tree. Let us consider the current partial solutions at a given level, which are considered one at a time. In the machine scheduling context, which is the field where the

RBS is mostly applied, each of these solutions is related to the partial schedule of a subset of jobs.

Thereafter, the recovering step checks if a current partial solution x can be dominated by another partial solution y. This last solution is typically obtained by interchanging operators of the current partial schedule (x). Such an interchange implies changing the sequence (therefore the starting times), in which the jobs included in the current schedule are executed. The global idea of this procedure is to identify a new partial solution y that dominates x, given that both schedules share the same tree level, which guarantees that the total number of explored nodes is polynomial considering that the search tree depth is polynomial.

Notice that initially, the eventual obtention of a good partial solution could be missed by means of the filtering procedure or by the principle of expanding only the most promising w nodes. Therefore, if we consider the RBS representation shown in figure 4.3, we can identify, that due to a beam width equal to two ($w = 2$), nodes d and e are not expanded. Nevertheless, when applying the recovering step for node h, we obtain a dominant solution, represented by node i. Subsequently, we can state that we were able to recover from a previous wrong decision, given that the forbidden branching on node d could also lead us to i.

Additionally, it is important to notice that the execution of the recovering phase implies the definition of a dominance rule which can be defined according to the features of the studied problem. Nevertheless, if it is not possible to define such a dominant condition, an alternative meassure is to consider a pseudo dominance rule which is determined in a heuristic fashion [40].

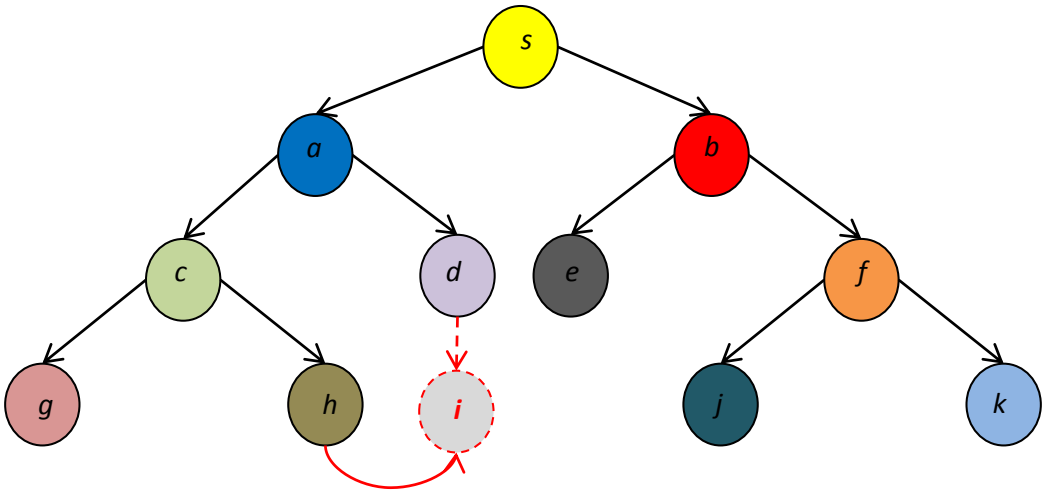

Figure 4.3: Representation of the Recovering Beam Search.

Moreover, Della Croce and T'kindt [40] introduced the recovering beam search (RBS) for solving the one-machine dynamic total completion time scheduling problem. The recovering beam search approach has since then been applied to several other problems. Thereafter, Della Croce et al. [39] applied a RBS approach for solving the two-machines total completion time flow shop scheduling problem and the uncapacitated p-median location problem. Ghirardi and Potts implemented a RBS [57] for the problem of scheduling jobs on unrelated parallel machines to minimize the makespan. Esteve [46] developed a recovering beam search procedure for a just-in-time scheduling problem with multiple criteria. Valente [111] compared several heuristic algorithms based on the beam search technique for solving the single machine weighted tardiness scheduling problem with sequence-dependent setups. Authors obtained that the RBS outperformed the other proposed approaches.

In this work, we explore the utilization of the RBS technique, since the recovering step feature could be exploited by means of the ILP models already explained in the previous sections. Consequently, the local improvement performed in the recovering step might lead to better results in comparison to the FBS [111]. When applying a recovering beam search, the total cost evaluation is typically based on a weighted sum between a lower bound and an upper bound ($V = (1 - \alpha) \cdot LB + \alpha \cdot UB$), where α is a parameter defined by experimental testing ($0 \leq \alpha \leq 1$). Notice that an accurate evaluation function will lead to a smaller deviation bteween the final solution value and the optimal one.

One way for calculating the lower bound can be by means of lagrangian relaxation [39, 57]. A different approach was applied by Della Croce and T'kindt, who considered the optimal solution of a the preemptive version of the considered problem [40] as a lower bound. Therefore, we can take adavantage of the column generation procedure proposed in section 3.2 for estimating a strong lower bound for a given node. We also exploit the dual information from the column generation master problem for estimating the upper bound of a node (see section 4.1.3), which represents a feasible solution.

4.2.2 Proposed recovering beam search

In order to apply the RBS to the MSPSP, it is necessary to specify their main components, namely the branching scheme, filtering procedure, upper and lower bounding procedures, and recovering step. Notice, that the proposed RBS considers also an additional repairing step executed in the branching phase. This additional feature is also explained in the next summary of the main components of the proposed approach.

Branching schemes

For this type of search tree algorithms, typically, in the scheduling field, branching approaches are based on the progressive building of a partial schedule. Thereafter, we consider a chronological branching approach, in which one activity is added to a partial schedule for generating a new node [2].

Adding one activity to a partial schedule: This branching scheme consists of adding one activity to a partial schedule for obtaining a new node. Thus, initially we define a set of elegible activities EL composed by all the available activities whose predecessors have already been scheduled. Therefore, one node is created for each activity A_i in EL, adding such an activity as soon as possible, at a time point that ensures the fulfillment of the precedence and resource constraints. Notice, that this last procedure considers all the available activities for building new partial schedules. Nevertheless, there are other procedures based on adding one activity to a partial schedule, for which the reader is referred to [2] for further details. This branching scheme was also considered in one of the B&P procedures introduced in section 4.1.2, allowing us to solve to optimality several small and medium size instances.

Regardless the branching scheme used, with the purpose of generating a new node we have to ensure a feasible assignment of workers for the subset of activities that could be added to a current partial schedule. Therefore, as was done for the B&P with the chronological branching strategy (B&PCB), we represent the resource assignment of such activities as a min-cost max-flow problem (see section 3.2.4). Hence, we try to fulfill the skills requirements of the activities involved considering the subset of available workers. Such a group of workers is defined according to the resource assignment stated at a previous level for the activities already scheduled.

For solving this last assignment problem, every time we apply Column Generation in a given node, we have to solve a linear program (RMP), from which we can obtain and store the dual information, that will allows us to approximate an assignment cost for each worker. Consequently, we aim at finding a feasible assignment of workers with the minimum total assignment cost as we did for solving the column generation sub-problem (see section 3.2.4).

Thereafter, if such an assignment is unfeasible, we perform a repairing step, in which we search for an alternative assignment of workers for the activities scheduled before time-point t. Hence, this reassignment could enhance a feasible assignment of workers for the subset of activities that might be added to the current partial schedule at given time-point t.

Repairing step and Filtering procedure

Repairing step

This step consists in solving the assignment model (AM($\Omega^{'}$)), applied also for the B&P with the delaying alternatives strategy (see section 4.1.2). In this ILP we fix the starting times of all the activities involved, and then, we aim on obtaining an assignment of workers that leads to a feasible new partial schedule, where the new subset of added activites can start at time t.

For applying AM($\Omega^{'}$) it is mandatory to predefine which combinations of workers can be considered for scheduling each activity involved in the execution of the recovering step. Therefore, we consider each subset of workers that has been created for each activity whenever a new column (i.e activity pattern) is generated by means of the CG procedure.

Filtering procedure

For filtering the nodes that will be evaluated, whenever there are two nodes x and y with partial schedules that contains the same subset of activities, we keep the one with a lower partial makespan. This implies, for example, that if the partial makespan related to node x is larger than partial makespan related to node y, x is pruned from the tree. Notice, that the partial makespan is equal to the maximal termination time among all the activities included in a partial schedule. In addition, we also apply the strategies for pruning nodes in the B&P approaches explained in section 4.1.4. Furthermore, the remaining child nodes are included in a set B.

Lower and upper bounds

With the purpose of calculating the evaluation cost function, which is equal to $V = (1 - \alpha) \cdot LB + \alpha \cdot UB$ we need to determine both a lower and an upper bound. Thus, we use the same methods that were applied for the proposed B&P procedures in section 4.1.3.

As a reminder, for defining a lower bound, we obtain a preliminary LB by propagating on the precedence graph. Subsequently, we estimate the stable set lower bound [16], and then, we enforce the best lower bound calculated so far for the current node by using the CG approach introduced in section 3.2.1.

In the other hand, for obtaining an upper bound we integrate the utilization of three methods: RCPL, ILPUB and ERCPL, which were already introduced in section 4.1.3. RCPL and ERPCL are heuristic methods based on the use of the dual information for determining which activity to schedule at a given time and which workers assign to it. The main difference betwen the two methods relies on the fact that ERPCL is reinforced by the repairing step in which an alternative reassignment of workers is explored for ensuring the schedule of a given activity at a certain time point. The ILPUB consists in solving the integer linear program proposed for the column generation master problem in section 3.2.1. For further details on this methods and how they are used to obtain a single UB for a certain node, the reader is referred to section 4.1.3.

Finally, the set of nodes B is organized in a non-decreasing order of their evaluation function.

Recovering step

Now, given that we retain at most w nodes at each level of the tree, the recovering step consists in considering both a reassignment of workers and a redefinition of the starting times of the activities that might lead to detect another partial solution that dominates the

current one. Moreover, instead of considering different permutations of a current partial schedule, we solve an ILP that allows us to search directly for a new partial schedule that could dominate the current one. This ILP, is based on the master problem formulation related to the column generation approach introduced in section 3.2.1. Thereafter, we initially introduce the mentioned model, and then, we describe two dominance conditions and one pseudo dominance condition, obtaining three different RBS procedures, which are tested and compared.

In this mathematical formalization, a column (i.e activity pattern) ω describes the processing attributes of an activity $A_i \in A'$. The subset A' considers all the activities included in a current partial schedule. We remind to the reader that an activity pattern ω is represented by three parameters: (i) α_i^ω which takes the value of 1 if activity A_i is processed in activity pattern ω or 0 otherwise; (ii) β_i^ω which takes the value of t if activity A_i starts at time t in activity pattern ω or 0 otherwise; (iii) $\gamma_{m,t}^\omega$ which takes the value of 1 if worker W_m is assigned on activity pattern ω at time t or 0 otherwise. We assume that the workers assigned to ω satisfy the skills requirement of the related activity.

Notice that Ω' covers the activity patterns only of the activities included in A', considering a fixed time-windows definition and each subset of workers that is created for each activity involved whenever a new column (i.e activity pattern) is generated by means of the CG procedure.

Thereafter the decision variables governing the target model are defined by: (i) x_ω which takes the value of 1 if activity pattern ω is selected or 0 otherwise; (ii) t_i which represents the starting time of an activity A_i;

Additionally, we define the termination time of A_i as C_i, where $C_i = t_i + p_i$. Now, the related ILP ($AM'''(\Omega')$) is stated as follows:

$$Z[AM'''[\bar{\Omega}']] : Min \sum_{i \in A'} v_i \cdot C_i \tag{4.7}$$

$$S.t.$$

$$\sum_{\omega=0}^{\Omega'} (x_\omega \cdot \alpha_i^\omega) = 1 \quad \forall i \in A' \tag{4.8}$$

$$\sum_{\omega=0}^{\Omega'} (x_\omega \cdot \beta_i^\omega) = t_i \quad \forall i \in A' \tag{4.9}$$

$$\sum_{\omega=0}^{\Omega'} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in M, \forall t \in [0, T] \tag{4.10}$$

$$C_i \leq t_j \quad \forall i \in A', \forall j \in E_i' \tag{4.11}$$

$$es_i \leq t_i \leq ls_i \quad \forall i \; A' \tag{4.12}$$

$$x_\omega \in \{0, 1\} \quad \forall \omega \in \{0, \Omega'\} \tag{4.13}$$

In this new model, we introduce a new parameter v_i which represents a weight related to the possible termination time of each activity A_i. Subsequently, the objective is to minimize the weigthed sum of the termination times of the activities in A' (4.7). Constraint set (4.8) ensures that only a unique activity pattern can be assigned to each activity A_i. Constraint set (4.9) recovers the associated starting times, while constraint set (4.10) ensures that any operator can carry out at most one activity at a given time. Constraint (4.11) states the precedence relations of the activities in A', where E'_i contains the direct successors of each activity A_i. Constraint set (4.12) ensures that the starting time of each activity must be within a predefined time-window. The values of es_i and ls_i are defined according to different dominance and pesudo-dominance conditions, that are explained later on.

ILP weights definition

The aim of solving the proposed ILP is to obtain a new partial schedule that dominates the current one and that could lead to an improvement of the current makespan upper bound. Thereafter, we part from the assumption that it could be useful to capture information that let us know which activities should start sooner considering the current partial schedule. Hence, the purpose of solving $AM'''(\Omega')$, is to obtain a new partial schedule that dominates the current one, in which we can reduce the termination times of the activities that have a higher influence (weight) in the final makespan of the project. Therefore, for estimating the weights v_i of each activity involved, we introduce a new parameter q_i. Now, if we remind the precedence graph G, hence, we can state that q_i represents the maximal path from the node of activity A_i to the node of A_N which represents the termination of the project. So, it corresponds to the tail of the job. Then, we can obtain a lower bound lb_i for the makespan of the project, by setting $lb_i = t_i^0 + q_i$, where t_i^0 represents the starting time defined for activity A_i in the current partial schedule.

Now, if we consider the lower bound of the current node (LB), we can define the value of v_i as follows:

$$v_i = 1/(1 + (LB - lb_i)) \quad \forall ß \in A' \tag{4.14}$$

Consequently the value of v_i will be higher for the activities for which lb_i is closer to LB. Let us remind that LB is the resulting bound after applying the CG procedure in the node related to the current partial schedule. In order to state the impact of given more priority to reducing the termination times of certain activities, we also evaluate the performace of the proposed ILP considering a weight equal to one ($v_i = 1$) for all the activities in A', which mean that the tails of the jobs are not taken into account.

Dominance and pseudo dominance conditions

Notice, that the definition of the dominance and pseudo dominance conditions are strictly related to constraint (4.12), which defines the time-windows domain of the starting times of the activities in A'. We consider two dominance conditions and one pseudo

dominance condition, which are separately tested and compared. The utilization of the next three rules leads to three different RBS procedures.

Strict dominance condition (RBS1): This condition states that the resulting termination time C_i after solving the previous ILP should be less or equal than the termination time C_i' of each activity A_i included in the current partial schedule PS_0 ($C_i \leq C_i'$). In the worst case scenario the resulting partial schedule could be the same as the current one, but we can ensure that it will never lead to a greater makespan lower bound than the one that corresponds to the current partial solution. Thus, the values of es_i and ls_i can be defined as follows ($es_i \leq t_i \leq ls_i$):

$$es_i = es_i^0 \quad \forall ß \in A' \tag{4.15}$$

$$ls_i = t_i^0 \quad \forall ß \in A' \tag{4.16}$$

Notice that es_i^0 represents the earliest starting time of each activity A_i defined in the root node, which is induced by the precedence graph using recursively Bellman's conditions. Additionally, we remind that t_i^0 represents the starting times related to the current partial schedule.

This dominance rule is considered as strict, taking into account that it does not allow to increase any of the starting times related to the current partial schedule. Hence, the obtention of a dominant schedule relies on finding at least one activity that could be executed sooner without having to increase the starting time of another activity.

Cut-set dominance condition (RBS2): This dominance condition takes into account both the termination times (C_i') associated to the current partial schedule and the resulting earliest time-point (m^0) in which a new eligible activity could be executed. Now, this new condition is based on the cut-set dominance rule proposed by Demeulemeester and Herroelen [42]. Such a rule is known as one of the most efficient dominance rules when using Branch-and-Bound methods for solving the RCPSP. This dominance rule is based on the concept of a cut-set, in which at every time-point m^0, a cut-set $CS(m^0)$ is defined, containing all unscheduled activities for which all of their predecessors belong to the partial schedule $PS(m^0)$. Additionally, a second cut-set $CS(m)$ previously stored ($PS(m)$ is the corresponding partial schedule) is also considered. This last cut-set contains the same activities as $CS(m^0)$. If $m \leq m^0$, and if each activity A_i in progress at time m does not finish in $PS(m)$ later than $max\{m^0, C_i'\}$, then the current partial schedule $PS(m^0)$ is dominated by $PS(m)$.

Thereafter, when applying the recovering step, instead of comparing the current partial schedule with a previous stored one, we enforce $AM'''(\Omega')$ to obtain a new partial schedule ($PS(m)$) that dominates the current one ($PS(m^0)$). For such a purpose, and considering the dominance cut-set rule principle, the starting times domain for the activities of the new partial schedule are defined as follows ($es_i \leq t_i \leq ls_i$):

$$es_i = es_i^0 \quad \forall ß \in A' \tag{4.17}$$

$$ls_i = max\{(m^0 - p_i), t_i^0\} \quad \forall ß \in A' \tag{4.18}$$

Notice also that the earliest time-point (m) in which a new eligible activity could be executed according to the new partial schedule should be less or equal than m^0.

It is important to mention that this dominance rule is more flexible than the previous one, given that an activity of the new partial schedule is not forced to start before t_i^0. Thus, it could be more likely to obtain a dominant schedule in which at least one activity could be executed sooner, even if it is mandatory to increase the starting time of another activity.

Pseudo dominance condition (RBS3): This alternative condition does not ensure obtaining a new dominant partial schedule, but it aims at defining time-windows that includes a wider domain of values for the starting times of each activity A_i in A'. Moreover, for each activity A_j in A' that has at least one direct successor in the list of eligible activities (EL), let us define a new parameter parameter m_j^0. Hence, m_j^0 is set equal to the minimal possible starting time among the eligible activities that are direct successors of A_j. Thus, we state that the latest starting time of A_j should be equal to $m_j^0 - p_j$ ($ls_j = m_j^0 - p_j$). Subsequently, for each activity A_l that does not have any direct succesors in EL, we define its latest starting time according to the cut-set rule ($ls_l = max\{m^0 - p_l, t_l^0)\}$, in order to increase the possibility of obtaining a dominant new partial schedule. Additionally, for each activity A_i in A' the corresponding earliest starting times are defined as equal to the the ones defined for the root node ($t_i \geq es_i^0$).

Notice that the earliest time-point (m^0) in which a new eligible activity could be executed, is always lower or equal than each m_j^0, which implies that the latest starting of a given activity A_j defined by means of the proposed pseudo dominance rule should be larger or equal than the one defined by the cut-set dominance condition.

Furthermore, it is important to consider that whenever there is an activity A_j for which $m_j^0 > m^0$, $m_j^0 > C_j^0$, and $t_j = m_j^0 - p_j$, it is possible that in the new resulting schedule it could take place a resource conflict that forces the unscheduled activities to start later than in the current partial schedule. Hence, when such a situation arises we can not state that new partial schedule dominates the current one.

Finally, for enforcing the impact of the described dominance and pseudo dominance conditions, we studied the inclusion of the next constraint, related to the total completion time of the resulting partial schedule, in the ILP previously described:

$$\sum_{i \in A'} C_i \leq \sum_{i \in A'} C_i^0 - 1 \tag{4.19}$$

This constraint ensures the reduction of the termination time of at least one activity in A'. Nevertheless, it is important to notice that a new partial schedule does not have to fulfill this last constraint to be dominant over the current partial schedule. Therefore, we compared the results of using the original ILP ($AM'''(\Omega')$) and the results of the resulting model after including constraint (4.19) ($AM'''(\Omega')$ & $\sum_{i \in A'} C_i \leq \sum_{i \in A'} C_i^0$).

Furthermore, when applying the recovering step, the mentioned ILP is solved considering also an imposed time limit of 10 seconds. If a feasible solution is obtained, the current partial solution is replaced by the resulting new one, which is included in the set of nodes S. This recovering step is performed until adding w nodes to S or exploring all the nodes in the set of kept nodes B. Initially, we state that the starting times of all the activities involved can be redefined. Additionally, we keep register of the maximum number of activities (Q_{max}) for which we can obtain a feasible solution after solving the ILP. Thereafter, for decreasing the amount of time invested in the second recovering step, the first time a solution is not found by the ILP within the imposed time limit, we fix the starting time of Q_{max} activities included in a certain partial schedule. Therefore, every time a solution is not found within the 10 seconds the value of Q_{max} is increased. The second recovering step is executed if the number of activities of the current partial schedule is larger than the value of Q_{max}.

4.2.3 Computational results

Computational experiments were performed using the solver Gurobi OptimizerVersion 4.0. As was done for the previous solution methods we selected a subset of the available instances for the MSPSP [93] according to their size in terms of number of activities, skills and number of workers. We consider, the same three groups of instances studied in the previous sections, nevertheless, here we show results for instances with bigger sizes. In general terms, the computational results showed in this section correspond to instances which consider between: 20 and 92 activities, 2 and 15 skills, and 2 and 19 workers. We show results for 384 instances, thereafter, the considered sizes for each group of tested instances (see section 2.10) are summarized as follows:

- Group 1: In this section we studied 113 instances from this group, considering: between 20 and 51 activities, between 2 and 8 skills, and between 5 and 14 workers.
- Group 2: In this section we include results for 94 regarding this group of instances. Evaluated instances consider between: 32 and 92 activities, 9 and 15 skills, and 5 and 19 workers.
- Group 3: In this section we studied 177 instances which considers between: 22, and 92 activities, 3 and 12 skills, and 4 and 15 workers.

Moreover, regarding to the parameters required for applying the proposed RBS, it is important to notice that there is a trade-off between solution quality and computational time, since increasing the value of the parameter w usually improves the objective function value, but at the cost of increased computation times. Therefore, we tried to determine the values that provided the best balance between solution quality and computational effort. The following values were considered for the two parameters considered when applying the RBS: (i)$\alpha = \{0, 1, 0, 2, \ldots, 0, 9\}$; (ii) $w = \{1, 2, \ldots, 8\}$.

The proposed algorithm was applied to selected problem sizes for all combinations of the relevant parameter values. Considering the objective function values and runtimes the parameters values that seemed to provide the best trade-off between solution quality and computation time were $w = 3$ and $\alpha = 0.1$.

Furthermore, with the purpose of analyzing the impact of the proposed recovering step we compared four possible beam search approaches. Therefore, initially we considered a beam search procedure (BS) which only includes the repairing step. Subsequently, we tested three RBS procedures, that correspond to the utilization of the dominance and pseudo dominance conditions previously explained. Hence, RBS1 corresponds to the RBS procedure in which the strict dominance rule is applied in the recovering step. In addition, RBS2 corresponds to the RBS procedure in which the cut-set dominance rule is applied in the recovering step. Finally, RBS3 corresponds to the RBS procedure in which the pseudo dominance rule is applied in the recovering step. It is important to state that a time limit of 5 minutes was imposed for applying BS, RBS1, RBS2 and RBS3. Notice that in the three RBS procedures we considered the resolution of the recovering step ILP model $AM'''(\Omega')$ including constraint (4.19) ($\sum_{i \in A'} C_i \leq \sum_{i \in A'} C_i^0$). Additionally, weights values are calculated according to equation (4.14)($v_i = 1/(1 + (LB - lb_i))$).

Table 4.6 introduces a comparison between the four beam search procedures in terms of the total average computational time, average time until reaching the best upper bound, average deviation against the best known lower bound, average deviation against the solution obtained by the tabu search developed by Bellenguez-Morineau and Néron [17] and total number of instances optimally solved.

		Group of instances		
		Group 1	**Group 2**	**Group 3**
Average CPU time (sec)	BS	136,55	167,53	182,91
	RBS1	161,70	165,21	183,25
	RBS2	161,91	167,59	184,27
	RBS3	160,74	167,26	183,22
Average CPU time for obtaining best UB(sec)	BS	45,68	75,73	66,62
	RBS1	46,58	76,34	73,63
	RBS2	56,72	75,21	70,49
	RBS3	56,79	68,76	71,11
Average deviation between best UB and best known LB	BS	3,99%	6,10%	5,47%
	RBS1	3,83%	5,84%	5,37%
	RBS2	3,71%	5,88%	5,25%
	RBS3	3,68%	6,02%	5,24%
Average deviation between best UB obtained and the tabu search UB	BS	1,44%	2,83%	1,96%
	RBS1	1,28%	2,62%	1,90%
	RBS2	1,20%	2,63%	1,77%
	RBS3	1,16%	2,77%	1,74%
Number of optimal solutions	BS	37	40	47
	RBS1	39	42	50
	RBS2	45	41	50
	RBS3	44	40	51

Table 4.6: Performance comparison between the proposed beam search approaches

Obtained results show that there is not a signficant difference in the computational times between the four procedures, although, in general terms, BS seems to be slightly faster. Additionally, we can notice that the total CPU times are greater for groups of instances 2 and 3, given that they cover instances with up to 60 and 90 activities, which leads to increasing the global computational time.

Regarding the makespan performance measures, the three proposed RBS procedures outperformed the BS in terms of the deviation against the best known lower bounds and the tabu search upper bounds. In addition, the BS was able to reach optimality in 124 instances, while the RBS1, RBS2 and RB3 achieved the optimal solution in 131, 136 and 135 instances. Therefore, we can outline the impact of the recovering step, which overall enhances a better makespan performance. It is important to state that optimality was proven by comparing the obtained upper bound with the best known lower bounds. Notice also, that RBS2 and RBS3 slightly outperformed RBS1 in terms of the mentioned makespan performance measures. This last result can be justified by the fact that RBS1 considers a strict dominance rule in the recovering step, which limits the possibilty of identifying new dominant partial schedules. In the other hand, the respective time-windows definition rules applied in the recovering step of RBS2 and RB3 are less strict and cover a larger domain of values. Hence, such RBS procedures are more likely to find new dominant partial schedules than RBS1.

Moreover, in table 4.7 we compare the results of the four beam search procedures in terms of the total number of generated nodes, average percentage of repaired nodes, average percentage of recovered nodes and the percentage of the total computational time invested in the repairing and recovering steps.

Concerning the repairing step we can state that overall, for all the tested procedures, more than the 23% of the generated nodes are repaired, taking only close to 1% of the total computational time. This, last finding, outlines the impact of applying the repairing step, considering that it allows us to avoid branching on the skills or in the possible set of workers that could be assigned to a given activity at a certain time. Therefore, this step is applied every time a reassignment of workers is required for ensuring the execution of a subset of activities, for which their starting times were previously defined.

Now, regarding the recovering step, we can state that, considering all the instances, for all the tested procedures, less than 3% of the generated nodes, it was possible to find a new dominant partial solution, taking near to 10% of the total computational time. Despite this low percentage of recovered nodes, in table 4.6 we already show the impact of the recovering step for obtaining better makespan values. It is important to notice that the execution of the recovering step relies in the resolution of an ILP more complex that the one solved in the repairing step. In one hand, in the repairing step we aim at finding a reassignment of workers considering that the starting times of the activities involved were already fixed. On the other hand, in the recovering step we deal with finding a reassignment of workers and a redefinition of the starting times of the activities involved. Hence, we can justify the difference between the percentage of time invested in both the repairing step and the recovering step.

		Group of instances		
		Group 1	**Group 2**	**Group 3**
Average number of generated nodes	BS	202,03	114,75	227,55
	RBS1	205,65	109,02	216,51
	RBS2	191,35	108,19	204,43
	RBS3	191	111,45	207,62
Average % of repaired nodes	BS	22,34%	76,62%	28,10%
	RBS1	23,39%	73,74%	25,80%
	RBS2	24,19%	75,25%	26,98%
	RBS3	24,01%	74,38%	27,41%
Average % of recovered nodes	BS	-	-	-
	RBS1	1,75%	1,10%	1,81%
	RBS2	2,43%	1,32%	3,07%
	RBS3	2,52%	1,29%	3,07%
Repairing step % of total CPU time obtained and the tabu search UB	BS	1,02%	1,86%	0,69%
	RBS1	0,75%	1,86%	0,46%
	RBS2	0,68%	1,78%	0,44%
	RBS3	0,63%	1,76%	0,42%
Recovering step % of total CPU time obtained and the tabu search UB	BS	-	-	-
	RBS1	15,39%	0,46%	7,63%
	RBS2	19,74%	0,95%	14,41%
	RBS3	21,20%	0,67%	14,93%

Table 4.7: Repairing and recovering steps performance comparison between the proposed beam search approaches

Furthermore, in table 4.8 we compare the performance of RBS2, which obtained good results in terms of the makespan performance measures shown in table 4.6, and two new RBS procedures: RBS2.1 and RBS2.2. In RBS2.1, we use the cut-set dominance condition in the recovering step without the inclusion of the total completion time constraint (4.19) ($\sum_{i \in A'} C_i \leq \sum_{i \in A'} C_i^0$). In addition the corresponding weights values are calculated according to equation (4.14)($v_i = 1/(1 + (LB - lb_i))$). On the other hand, in RBS2.2 the weights values are defined as equal to one ($v_i = 1$), the total completion time constraint is included in the recovering step ILP and the time-windows are also defined according to the cut-set dominance condition. Therefore, the comparison between RBS2 and RBS2.1 allows us to analize the impact of adding the total completion time constraint (4.19) ($\sum_{i \in A'} C_i \leq \sum_{i \in A'} C_i^0$) in the recovering step ILP. Subsequently, the comparison between RBS2 and RBS2.2 allows us to evaluate the influence in the final makespan value of given more priority, to reducing the termination times of certain activities.

Obtained results show that RBS2 outperforms RBS2.1 in terms of the number of optimal solutions achieved and the average deviation between the obtained upper bound against the best known lower bound and the tabu search upper bound. More specifically, we can notice that RBS2 reached the optimal solution in 136 instances, while RBS2.1 achieved the optimal solution in 116 instances. In addition, considering all the tested instances, average deviations against the best known lower bounds and the tabu search

		Group of instances		
		Group 1	**Group 2**	**Group 3**
Average deviation against best UB and best known LB	RBS2	3,71%	5,88%	5,25%
	RBS2.1	4,06%	5,80%	6,02%
	RBS2.2	3,74%	5,87%	5,37%
Average deviation against best UB obtained and the tabu search UB	RBS2	1,20%	2,63%	1,77%
	RBS2.1	1,53%	2,59%	2,46%
	RBS2.2	1,23%	2,65%	1,87%
Number of optimal solutions	RBS2	45	41	50
	RBS2.1	39	40	37
	RBS2.2	42	42	48
Average number of recovered nodes	RBS2	4,64	1,24	6,28
	RBS2.1	6,37	1,34	5,82
	RBS2.2	4,75	1,35	5,93

Table 4.8: Impact of the weight definition criteria and the total completion time constraint in the recovering beam search ILP

upper bounds are: 4,95% and 1,82% for the RBS2 and 5,40% and 2,22% for the RBS2.1. Thus we can infer, that given that there is not a significant difference between the percentage of recovered nodes by RBS2 and RBS2.1, the total completion time constraint somehow is able to lead to new dominant partial solutions that are more likely to lead to a better makespan upper bound.

Regarding the comparison between the performance of RBS2 and RBS2.2, obtained results show that both procedures present similar results in terms of the makespan performance measures. However, taking into account all the tested instances, RBS2.2 reached the optimal solution in 132 instances, obtaining also an average makespan deviation of 1,88% against the best known lower bound and a deviation of 5,02% against the tabu search upper bound. Thus, we can state that RBS2 slightly outperforms RBS2.2, which implies that given priority to reducing the termination times of certain activities by means of the proposed weight definition criteria ($v_i = 1/(1 + (LB - lb_i))$) can lead to better makespan upper bounds.

4.2.4 Conclusion

In this section, we introduced and compared several beam search procedures that exploit the structure of the branch and price procedures discussed in section 3.2. In addition, we were able to show the impact of the repairing and recovering steps for obtaining good makespan upper bounds. Computational experiments throwed comparable results to the ones obtained with the tabu search developed by Bellenguez-Morineau and Néron [17]. Nevertheless, there can be other approaches that can be explored in the recovering step. In addition, other pseudo dominance rules can be proposed, in order to generate new partial schedules that might lead to better makespan upper bounds. Additionally, based on the impact of the inclusion of the total completion time constraint, it could be interesting to exploit the fact that the obtention of a new good dominant or pseudo dominant solution

can be guided by means of new constraints that can be included in the recovering step ILP. When searching for a new dominant partial solution, other weights definition criteria can be also explored, in order to give more exact information about which activities we should try to finish earlier, based on an estimation of the impact in the final makespan value.

5

Cut Generation Procedure

In this section we explore another approach for solving the MSPSP. Overall, the solution methods previously explored are related to the resolution of mathematical models that integrates the definition of the starting times of the activities and the assignment of workers in a single solution phase. Therefore, we also explore the option of solving the MSPSP using a two-phase approach. The first one involves the definition of the starting times of the activities of the project. Thereafter, in the second phase we focus on finding a feasible assignment of workers that allows the execution of the activities according to the starting times defined in the first phase. The resolution of this two-phase procedure is done in an iterative fashion in which new inequalities (cuts) are generated and included into an ILP until ensuring an optimal schedule.

5.1 Cut Generation background

Any Integer Linear Program (ILP) can be solved without branching by finding a linear programming description of the set of all the convex combinations of feasible solutions. In order to do that, one can iteratively solve the so called separation problem, which consists in finding a violated cut [78]. A cut is a valid inequality that is not a part of the current formulation and that is not satisfied by all feasible points to the linear programming (LP) relaxation. Hence, a cut that is not satisfied by the given LP relaxation optimal solution is called a violated cut. Thereafter, adding a violated inequality could lead to a tighter linear relaxation. Subsequently, it is conceivable to solve an ILP by solving its linear relaxation and progressively augmenting it with more and more valid inequalities. This last procedure is known as the (pure) cutting planes method.

The cutting plane method dates back to the 1950s and the pioneering work of George Dantzig and Ralph Gomory. George Dantzig together with Ray Fulkerson and Selmer Johnson [35, 36] developed the cutting plane method to tackle the famous Traveling Salesman Problem (TSP) [1]. While their algorithmic scheme was based on special purpose

cutting planes, Gomory [60, 61] introduced a fully generic cutting plane method based on the famous Gomory mixed integer cuts. He also proved that his method converges after a finite number of steps in case of a pure integer program and rational data. See the work of Zanette et al. [119] for a recent discussion on the convergence of the pure cutting plane method based on Gomory cuts.

Notice that no finite cutting plane algorithm is known for general mixed Integer Programs. Branch-and-bound and cutting planes have been combined first by Hong [69] while the term branch-and-cut was first used by Padberg and Rinaldi [98], see [30]. Branch-and-cut is state-of-the-art and implemented in modern commercial solvers such as Cplex, Gurobi, or Xpress among others. In branch-and-cut globally or locally valid cutting planes are added to all sub-problem formulations in the search tree to tighten the relaxations and to improve the dual bounds. In case cuts are only used in the root node of the tree before branching we speak of cut-and-branch [34]. Overall, a major step was required to prove that cutting plane generation in conjunction with branching could work in general, i.e., without exploiting the structure of the underlying polyhedron, in particular in the cutting plane separation.

Given a particular problem, it is usually possible to identify at least certain classes of (strong) valid inequalities. After early successes by Grötschel and Padberg [62] and many others in understanding the TSP polytope and solving larger TSP instances by branch-and-cut, the research on cutting planes experienced a rapid growth considering all kinds of different problems by polyhedral methods and devising problem specific codes with tailored cutting planes, see [1].

Despite the success of branch-and-cut for combinatorial optimization problems, cutting planes have long been ignored in the context of solving general Mixed Integer Programs (MIP) instances with general purpose solvers. Gomory cuts have, initially, been believed to be a theoretical tool only, causing numerical instabilities and slow convergence in practice. Hence early MIP codes mainly focused on efficient LP based branch-and-bound. The situation changed only in the late nineties with a publication of Balas et al. [4] who show that Gomory cuts can be embedded effectively into a branch-and-cut framework to solve $\{0; 1\}$-MIPs (integral variables are restricted to the values 0 or 1). One of the new observations from this last work, was that adding sets of cuts clearly outperforms adding just a single violated cut in every separation step.

In addition to Gomory cuts, several classes of general cutting planes have been developed over the years and tested with respect to their computational impact. We cannot introduce all these classes in detail here but refer the reader to overviews like the ones given by Marchand et al. [88] and Cornuéjols [32] among others. Nevertheless, in order to have an intuition of the usefulness of the inclusion of a cut, we know give a short description of different types of inequalities.

5.2 Standard Inequalities

Considering the particular approach proposed in this chapter, initially we introduce the Clique and Cover inequalities which according to our criteria could be appropriate for representing certain constraints of the MSPSP.

Clique Cuts: For this type of inequality, a presolve phase is required, for determining a group Q of binary variables, such that no more than one can be non-zero at the same time [79]. This inequality is given by:

$$\sum_{i \in Q} x_i \leq 1 \tag{5.1}$$

The most fundamental work about presolving is the one of Savelsbergh [108] to which the interested reader is referred to. Overall, preprocessing may be useful for identifying infeasibility and redundant constraints, fix variables, strengthening the corresponding LP relaxation, and generating new valid inequalities. Hence, for certain problems, it could be really useful a special-purpose preprocessing which exploits the specific problem structure.

Cover inequalities: If a constraint takes the form of a knapsack constraint (that is, a sum of binary variables with nonnegative coefficients less than or equal to a nonnegative right-hand side), then there is a minimal cover associated with the constraint. A minimal cover is a subset Q of the variables of the inequality such that if all the subset variables were set to one, the knapsack constraint would be violated, but if at least one variable is set to zero, the constraint would be satisfied. This constraint is given by:

$$\sum_{i \in Q} x_i \leq |Q| - 1 \tag{5.2}$$

It is important to notice that the amount of work devoted to cover cuts is huge starting with the seminal work of Balas [3] and Wolsey [116]. In addition, to the inequalities just mentioned, we also present other types of inequalities widely used when applying cutting planes methods.

Gomory Fractional Cuts: These type of inequalities are generated by applying integer rounding on a pivot row in the optimal LP tableau for a (basic) integer variable with a fractional solution value [60, 61].

Flow Cover Cuts: These inequalities are generated from constraints that contain continuous variables, where the continuous variables have variable upper bounds that are zero or positive depending on the setting of associated binary variables. The idea of a flow cover comes from considering the constraint containing the continuous variables as describing a single node in a network where the continuous variables are in-flows and out-flows. The flows will be on or off depending on the settings of the associated binary variables for the variable upper bounds. The flows and the demand at the single node imply a knapsack constraint. That knapsack constraint is then used to generate a cover cut on the flows (that is, on the continuous variables and their variable upper bounds)[99].

5.3 Global description of the two-phase solution approach

The two-phase approach proposed in this section implies the iterative resolution of two integer linear programming (ILP) models.

In the first phase we solve an ILP, which considers a single decision variable related to the starting time of the activities. The activities and skill requirements constraints are approximated by means of cuts on such a variable. Subsequently, we can estimate the starting times of all the activities of the project, obtaining also a lower bound for the makespan.

In the second phase we apply an assignment model for finding a feasible assignment of workers given the starting times defined in the first phase. Then, if the solution of this last model leads to a feasible assignment of workers, we can ensure that we reached an optimal schedule. In the opposite case, it is implied that we have to redefine the starting times of the activities. For such a purpose we aim on preventing an infeasible schedule by generating and adding new cuts to the ILP used in the first phase. Thus, these new cuts are added with the purpose of achieving starting times that enhance a feasible assignment of workers and therefore, an optimal schedule. The resolution of the two solution phases and the inclusion of new cuts are done iteratively until the workers assignment model leads to a feasible solution (see figure 5.1).

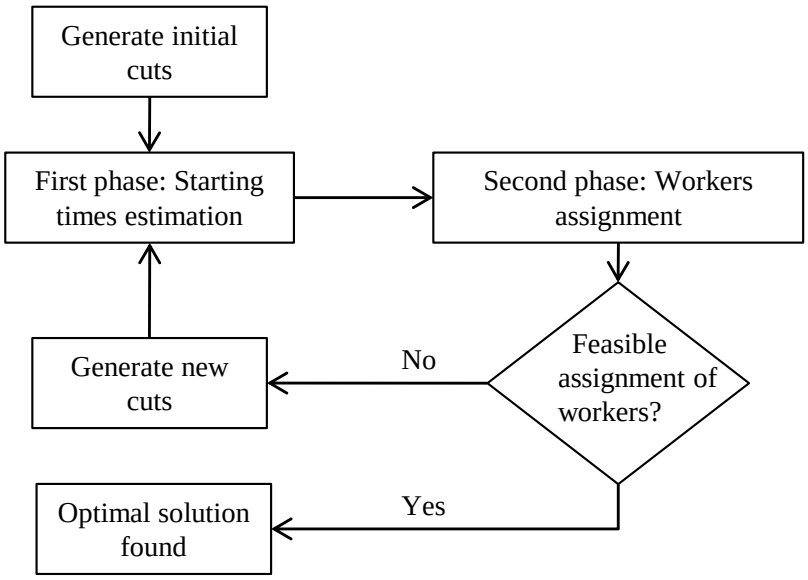

Figure 5.1: Two-phase approach for solving the MSPSP.

It is important to notice that in the proposed approach we do not add cuts to the original formulation of the problem (which includes all the constraints) for solving its linear relaxation and progressively augmenting it with more and more valid inequalities, as it is done in the (pure) cutting planes method. The main purpose of our approach is to propose cuts that are suitable for approximating the activities and skill requirements constraints.

Hence such inequalities are added iteratively to an ILP that initially contains a subset of the constraints of the original formulation of the problem. After adding the cuts, we solve the ILP (not only the linear relaxation). Nevertheless, we aim on identifying inequalities that takes into account the structure of the classic cuts introduced in the previous section.

It is also important to mention that the the proposed two-phase solution method it can be described also as a Benders or combinatorial Benders-like method [70, 73]. Therefore, we can relate the first phase to the Benders master problem and the second phase to the (combinatorial) Benders sub-problem.

5.3.1 First phase: New time indexed model STIMWS

The first step of the solution approach presented in this section consists in solving a time indexed model. Hence, we consider the decision variable z_i^t which takes the value of 1 if an activity A_i starts at time t, or takes the value of 0 otherwise. This ILP (STIMWS) can be considered as a simplified version of the TIMWS introduced in section2.11.2).

Initially, our main concern is to estimate the starting times of the activities, thus the activities and skill requirements constraints are approximated by means of cuts on z_i^t. The resulting ILP (STIMWS) is defined as follows:

$$Z[STIMWS] : Min\, C_{max} = t_N \tag{5.3}$$

$S.t.$

$$\sum_{t\in[0,T]} (z_i^t \cdot t) + p_i \leq \sum_{t\in[0,T]} (z_j^t \cdot t) \quad \forall i \in A, \forall j \in E_i^+ \tag{5.4}$$

$$\sum_{t\in[0,T]} z_i^t \leq 1 \quad \forall i \in A \tag{5.5}$$

$$\sum_{i\in A'} \sum_{s\in[t-p_i+1,t]} z_i^s \cdot b_i^{K'} \leq b(A'; K'; T') \quad \forall t \in T' \tag{5.6}$$

$$\sum_{i\in A'} \sum_{s\in[t-p_i+1,t]} z_i^s \leq l(A'; T') \quad \forall t \in T' \tag{5.7}$$

$$z_i^t \in \{0, 1\} \quad \forall i \in A, \forall t \in [0, T] \tag{5.8}$$

The objective (7.44) is to minimize the completion time (makespan) of the project. Constraint set (7.45) represents the precedence relation between the activities, which implies that the finishing time of an activity must be less or equal than the starting time of its successors. Constraints (7.46) ensure that an activity is performed at most once during the whole planning horizonInequalities (7.47) and (7.48) allows an approximation of the activities and skill requirements.

More precisely, constraints (cuts) (7.47) and (7.48) are formally defined as follows:

Definition 5.1 *(Cumulative cut). Let A' be a subset of activities, K' a subset of skills and T' a set of time periods of the planning horizon. Additionally, we know that for any period*

of T' there is no more than a certain quantity $b(A';K';T')$ of workers simultaneously available for performing activities in A' using at least one skill in K'. We also have that the total number of workers required for performing all the skills in K' related to each activity is given by $b_i^{K'}$. Thereafter, we can state the following cut:

$$\sum_{i \in A'} \sum_{s \in [t-p_i+1,t]} z_i^s \cdot b_i^{K'} \leq b(A';K';T') \quad \forall t \in T'$$

where $b_i^{K'} = \sum_{k \in K'} b_i^k$

For generating the cumulative cuts, we consider that A' contains all the activities of the project and that T' covers the time horizon T. Finally we generate all the possible combinations of skills, and for each generated subset K' we compute the value of $b(A';K';T')$. For defining the value of this last parameter, we assume that all the workers in W are available at a given time t and then, we calculate the maximum number of workers that could use at least one skill in K'.

For not having to add all the possible cumulative cuts to the proposed ILP, we perform a filter where we only add the dominant cuts. For illustrating the previous statement, let us assume that for a given time t we generate a cut C_0 that considers skills S_0, and S_1 with $b(A';S';T')$ equal to H. Such a cut states that we cannot assign more than H workers for using skills S_0 and S_1 at a time t for performing all the activities in A'. Hence, C_0 is defined as follows:

$$\sum_{i \in A'} \sum_{s \in [t-p_i+1,t]} z_i^s \cdot (b_i^0 + b_i^1) \leq H$$

Now, we generate another cut C_1 for time t that covers skills S_0, S_1, and S_2 with a $b(A';S';T')$ equal also to H. Subsequently, C_1 is stated as:

$$\sum_{i \in A'} \sum_{s \in [t-p_i+1,t]} z_i^s \cdot (b_i^0 + b_i^1 + b_i^2) \leq H$$

Given that the limit number of workers H is equal for both cuts, we can state that C_1 is dominant, since it already covers the situation restricted by C_0. Hence, we add only cut C_1 to STIMWS. Moreover, this filtering procedure allows us to reduce significantly the cumulative cuts added to the ILP model. Obtained results show that in average the 75% of all the cumulative cuts generated for a given time-point are dominated by the remaining ones. For further details, this results will be discussed later on in section 5.5.

Moreover, it is important to notice that this cumulative cut was originally considered by [102] for representing the resource constraints in a time indexed model for solving the RCPSP. Nevertheless, in the MSPSP context the utilization of these inequalities does not suffice to ensure a feasible assignment of workers as we will see it at the end of the section.

Definition 5.2 *(Cardinality cut). Let A' be a subset of activities and T' a set of time periods of the planning horizon. Additionally, we state that for any period of T' no more than a certain quantity $l(A';T')$ of activities from A' can be executed in parallel. Clearly, we can state the following cut:*

$$\sum_{i \in A'} \sum_{s \in [t-p_i+1,t]} z_i^s \leq l(A';T') \quad \forall t \in T'$$

Now, if we set $l(A';T')$ equal to one, we can classify this last cut as a clique inequality (7.42). Such an inequality was applied by [41] for solving the RCPSP. In such a work, authors proposed a cutting planes procedure, where a preprocessing phase was initially executed by generating disjunctive sub-problems from greedy heuristics to run some constraint propagation techniques. Hence, with these algorithms, non-overlapping relationships were detected, helping to generate new disjunctive sub-problems of larger sizes, obtaining stronger clique inequalities for the LP.

For generating the cardinality cuts, we perform a pre-processing procedure to define all the couple of activities (A_i, A_j) that according to the precedence relations might be simultaneously processed. Hence, if the parallel execution of such activities lead to a resource conflict, we generate a new cardinality cut, where $A' = \{A_i, A_j\}$. Therefore, the subset T' covers the time-period in which the activities in A' might be simultaneously processed. Finally, we generate each cardinality cut by fixing $l(A';T')$ equal to one. For detecting a resource conflict between the activities in A' we represent the assignment of workers as a min-cost max-flow problem [17]. Then, we solve such a problem with the min-cost max-flow algorithm proposed by [25] (see section 3.2.4).

Additionally, for enforcing the solution of the proposed time indexed model, we use the stable set bound, which was already explained in section 4.1.3 for defining an initial lower bound. Notice that the time horizon (T) was set equal to an upper bound computed with the Taboo Search procedure developed by Bellenguez-Morineau and Néron [17].

As mentioned before, the resolution of STIMWS including the cumulative and cardinality cuts does not suffice to ensure a non-preemptive assignment as we will illustrate on the following example. This last statement, is justified by means of the next example. Let us assume that we obtained the starting times of a given project after solving the proposed time indexed model. Now, let us consider a subset of activities composed by A_6, A_7 and A_8. The skill requirements of each activity are defined as follows:

– A_6 requires one worker that masters skill S_1.
– A_7 requires one worker that masters skill S_0.
– A_8 requires one worker that masters skill S_2.

We take into account two workers: W_0 and W_1. W_0 masters skills S_0 and S_2, while W_1 masters skills S_0 and S_1.

The execution of A_6, A_7 and A_8 is illustrated in figure 5.2. According to the cumulative and cardinality cuts we can state that the parallel execution of A_6 and A_7 is feasible if we assign W_0 to A_7 and W_1 to A_6. On the other hand, the simultaneous processing of A_7 and A_8 is feasible only when assigning W_0 to A_8 and W_1 to A_7. Nevertheless, it is not feasible to find an assignment of workers for executing the three considered activities according to the defined starting times. Such an infeasibility cannot be detected by the proposed cuts, since they don't take into account the interaction between the activities involved. Notice that, this solution can be considered as preemptive in the sense that if W_0 starts A_7 at $t = 8$ he could stop its execution for starting A_8 at $t = 9$. In the following sections we introduce a new cut that aims on preventing this type of infeasibility.

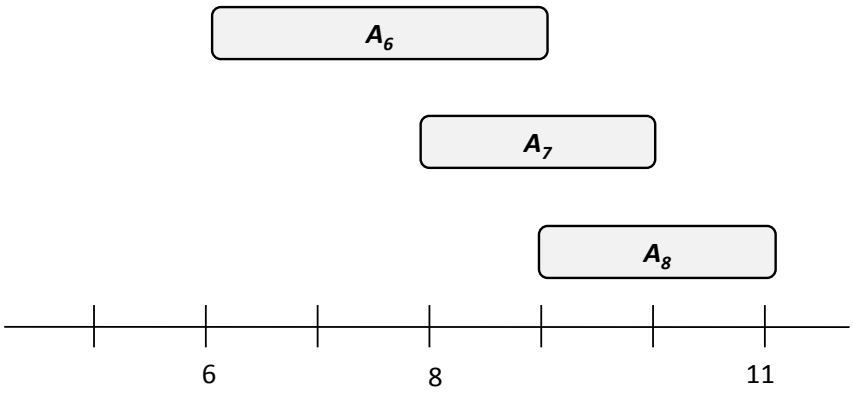

Figure 5.2: Processing among time of activities A_6, A_7 and A_8

5.3.2 Second phase: Workers assignment model AM′(Ω)

In the second phase, we propose a procedure for finding a feasible assignment of workers according to the starting times defined in the first solution phase.

Thus, we apply an assignment model(AM′(Ω)) inspired from the master problem proposed in the column generation approach introduced in section 3.2.1. In this mathematical formalization, a column ω describes the processing attributes of an activity A_i. Such an activity pattern ω is represented by two parameters: (i) α_i^ω which takes the value of 1 if activity A_i is processed in activity pattern ω or 0 otherwise; (ii) $\gamma_{m,t}^\omega$ which takes the value of 1 if worker W_m is assigned to activity pattern ω at time t or 0 otherwise. We assume that the workers assigned to ω satisfy the skills requirement of the related activity.

Notice that Ω covers the enumeration of all the possible combinations of workers that could be assigned to each activity of the project. Hence, before running the previous model we enumerate all the possible combinations of workers that could be assigned to each activity, which are obtained by means of the already mentioned min-cost max-flow algorithm (see section 3.2.4). This implies that we have to solve the min-cost max-flow

problem represented in figure 5.3, assuming that all the workers in W are available. Thus, we generate a new combination of workers for each activity A_i until there are no more combinations of workers that could lead to a feasible resource assignment.

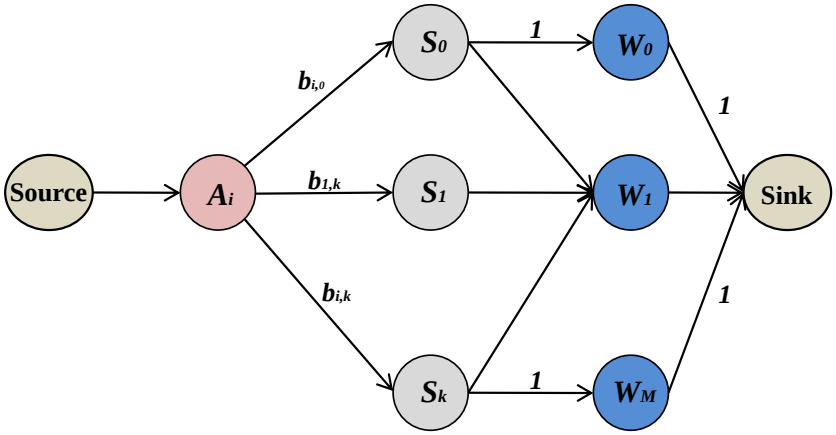

Figure 5.3: Workers assignment for each activity A_i.

Furthermore, the parameters related to each activity pattern are build according to the starting times defined in the first phase of our solution approach.

Thereafter, the decision variables governing the target model are defined by: (i) x_ω which takes the value of 1 if activity pattern ω is selected or 0 otherwise; (ii) v_i is a slack variable that is added to ensure feasibility for any partial selection of activity patterns. Now, the resulting ILP is stated as follows:

$$Z[AM'[\Omega)] : Min \sum_{i \in A} v_i \tag{5.9}$$

$$\sum_{\omega \in [0,\Omega]} (x_\omega \cdot \alpha_i^\omega) + v_i = 1 \quad \forall i \in A \tag{5.10}$$

$$\sum_{\omega \in [0,\Omega]} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{5.11}$$

$$x_\omega \in \{0,1\} \quad \forall \omega \ \in [0,\Omega] \tag{5.12}$$

$$v_i \geq 0 \quad \forall i \in A \tag{5.13}$$

The objective function of AM′(Ω) is to minimize the number of activities with an infeasible assignment of workers. Thus, if v_i is larger than zero, is because it is not possible to find a feasible assignment of workers for activity A_i. Constraint set (7.51), states that only a unique activity pattern can be assigned to any task A_i. Constraint set (7.52) ensures that any operator can carry out at most one activity at a given time. Additionally x_ω and v_i are defined as binary and continuous variables respectively.

If the solution of AM′(Ω) leads to a feasible assignment of workers ($\sum_{i \in A} v_i^* = 0$), we can ensure that the current schedule is non-preemptive, and therefore optimal. In the opposite case, we aim on generating a new cut that could be added to the time indexed model (STIMWS). Subsequently, we continue with the iterative procedure illustrated in figure 5.1.

Hence, let us define this new cut as follows:

Definition 5.3 *(Overlapping subsets cut). Let I be a subset of activities for which is not possible to find a feasible assignment of workers and D a subset of I. In addition, for all $i \in D$, D^i is a subset of activities that are processed simultaneously with an activity A_i ($A_i \in D$). Finally, let t_i be a predefined possible starting time for each activity A_i that belongs to the subset D. Hence, we generate the next cut:*

$$\sum_{i \in D} \sum_{j \in D^i} \sum_{s \in [t_i - p_j + 1, t_i + p_i - 1]} z_j^s + \sum_{i \in D} z_i^{t_i} \leq \sum_{i \in D} \left| D^i \right| + |D| - 1 \tag{5.14}$$

With this last cut, we restrict that at least one activity that belongs to any of the subsets D^i, cannot be performed in parallel with its corresponding activity A_i in D. Notice, that this last cut can be classified as a cover inequality. For understanding this last statement, we remind that in a cover inequality ((7.43)), whenever all the subset variables involved were set to one, the constraint would be violated, but if at least one variable is set as equal to zero, the constraint would be satisfied.

Now, let us consider the next example. Given a certain project, let us assume that after applying the first solution phase we obtained an estimation for the starting times of each activity. Thereafter, we solve AM′(Ω), and assuming that we obtained an infeasible solution, we identify a subset I that contains activities $A_3, A_4, A_5, A_6, A_7, A_8$. Supposing that the execution of the activities in I is represented by figure 5.4, we can identify the next subsets:

$$D = \{A_4, A_7\} \tag{5.15}$$
$$D^4 = \{A_3, A_5, A_6\} \tag{5.16}$$
$$D^7 = \{A_6, A_8\} \tag{5.17}$$

Thus, we have that D^4 corresponds to the activities that are done in parallel with A_4 and D^7 corresponds to the ones that overlap with A_7.

It is important to notice, that there could be other alternative subsets:

$$D = \{A_5, A_6, A_7\} \tag{5.18}$$
$$D^5 = \{A_3, A_4\} \tag{5.19}$$
$$D^6 = \{A_4, A_7\} \tag{5.20}$$
$$D^7 = \{A_6, A_8\} \tag{5.21}$$

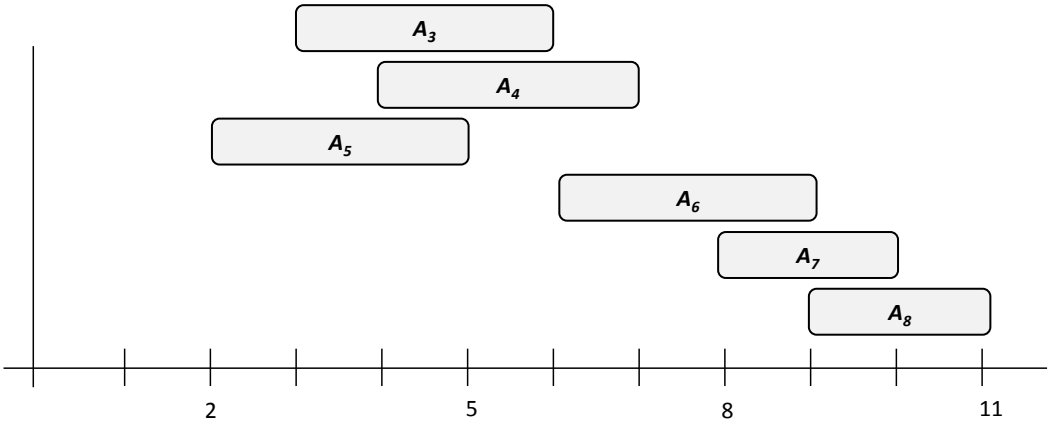

Figure 5.4: Processing of activities in I

Considering $D = \{A_4, A_7\}$ and that $t_4 = 4, t_7 = 8$ (see figure 5.4), we can generate the next overlapping subsets cut:

$$(\sum_{j \in D^4} \sum_{s \in [4-p_j+1, 4+p_4-1]} z_j^s) + (\sum_{j \in D^7} \sum_{s \in [8-p_j+1, 8+p_7-1]} z_j^s) + z_4^4 + z_7^8 \leq \left|D^4\right| + \left|D^7\right| + |D| - 1 \tag{5.22}$$

where $|D^4| + |D^7| + |D| = 7$

With this inequality, we are stating that the current interaction between the activities in D, D_4 and D_7 leads to an infeasible assignment of workers, whenever A_4 and A_7 starts at $t_4 = 4$ and $t_7 = 8$ respectively.

Furthermore, we have to define a procedure for identifying I, D and D_i (where $A_i \in D$) that leads to a "valid" cut. Notice, that a new overlapping subsets cut is considered as valid whenever it restricts a situation that will always lead to an infeasible schedule.

Procedure for identifying subset I

After solving AM′(Ω), we initialize the subset I with the activities with $v_i^* > 0$ and we define a new parameter F equal to the resulting optimal value ($F = \sum_{i \in A} v_i^*$). Notice that constraint set (7.51) does not allows us to detect directly all the activities that should be included in I. Hence, we add the next constraint to AM′(Ω) that forces a feasible assignment of workers for the activities currently included in I.

$$\sum_{i \in I} v_i = 0 \tag{5.23}$$

Moreover, if we consider again our previous example, we can state that A_7 and A_8 could be in conflict for the same resource. Hence, assuming that such a conflict exists, after solving the assignment model we might obtain that $v_7 = 1$ and $v_8 = 0$. Therefore,

we can include initially A_7 in I, although we should include also A_8. Consequently, assuming that currently $I = \{A_3, A_4, A_5, A_6, A_7\}$, we solve again AM′($\Omega$) but we set that $v_3 + v_4 + v_5 + v_6 + v_7 = 0$, hence we will force the model to define $v_8 = 1$, which will allow us to incorporate A_8 in I.

Thereafter, we follow an iterative procedure, where we add this last constraint, solve AM′(Ω) and update I with the activities with $v_i^* > 0$. This procedure is repeated as long as the current objective function of the assignment model is equal to F (see figure 5.5).

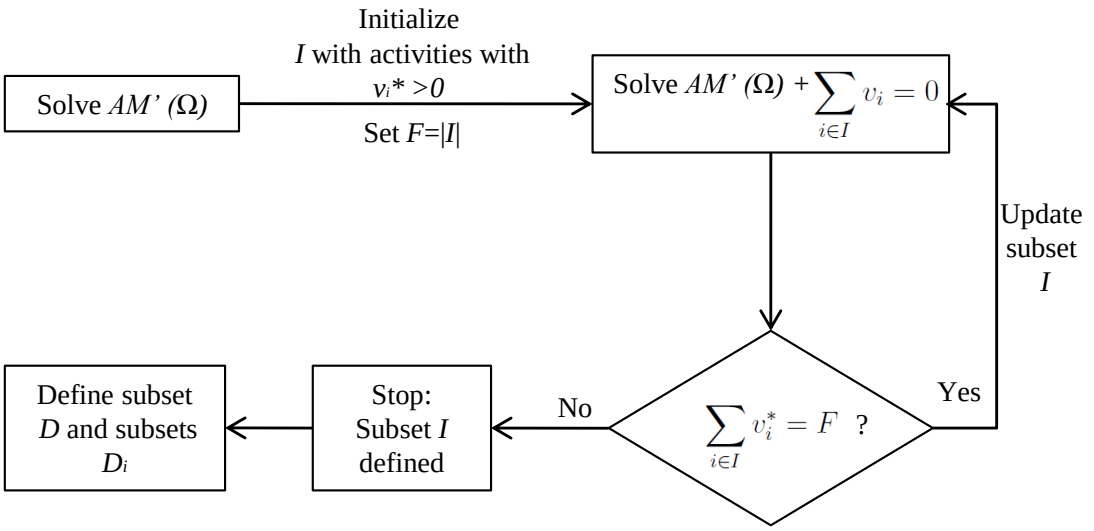

Figure 5.5: Procedure for identifying subset I

Procedure for identifying subset D

There is not a specific criteria for identifying a subset D that will lead us to generate a new valid cut. Hence, initially we part from the assumption that a cut might be stronger when D contains the minimum number of activities. For example if we based on the example that we described previously, if $D = \{A_4, A_7\}$, when generating the new cut we only have to fix the starting time of two activities, but if we consider the alternative subset $D = \{A_5, A_6, A_7\}$ we will have to fix the starting time of three activities. When fixing the starting time of a lower number of activities the cut will have an impact for a larger domain of possible starting times of the remaining activities in I. Thereafter, we give priority on minimizing the number of activities in D. Thus, we use a mathematical model (SC) which enhance the identification of the activities included in the mentioned subset. Such a model considers a decision variable y_i which takes the value 1 if activity A_i is included in the subset D, or 0 otherwise.

Additionally, we use a parameter $P_{i,j}$ which takes a value equal to 1 if A_j is currently executed in parallel with activity A_i, or 0 otherwise.

$$Z[SC] : Min \sum_{i \in I} y_i \tag{5.24}$$

$$\sum_{i \in I} (y_i \cdot P_{i,j}) \geq 1 \quad \forall j \in I \tag{5.25}$$

$$y_i \geq 0 \quad \forall i \in I \tag{5.26}$$

The aim of the model is to minimize the number of activities that will be included in D (7.57). Constraint set (7.58) states that each activity in I is performed in parallel with at least one activity in D.

Furthermore, it is important to notice that the execution of a given subset of activities I could lead to the situation shown in figure 5.6.

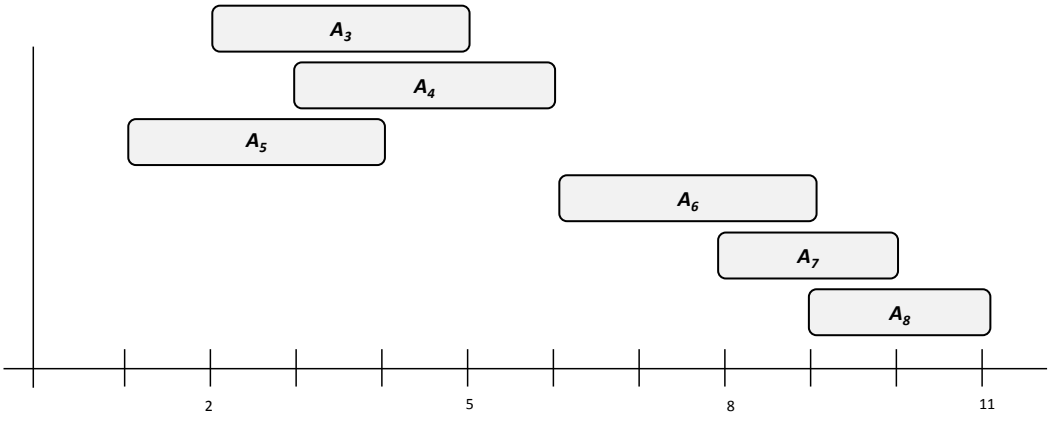

Figure 5.6: Processing among time of activities in I

Thus, we can generate two subsets I ($I = \{A_3, A_4, A_5\}$ and $I = \{A_6, A_7, A_8\}$), each one with a related subset D ($D = \{A_4\}$ and $D = \{A_7\}$). Consequently, we have two independent subsets of activities, that lead to an infeasible assignment of workers. Hence, when a similar situation arises we could generate more than one valid cut in the same iteration.

Overlapping subsets cut validation

Moreover, assuming that we already generated an overlapping subsets cut (which implies that we identified I and D), for validating the resulting inequality, we aim on finding a feasible schedule for the activities in I when fixing the starting times of the activities in D. Additionally, we enforce the overlapping in the execution between each activity A_i in D and the ones in D^i. Subsequently, if we are not able to find a feasible schedule for the activities in I, we can state that the overlapping subsets cut is valid, hence it can be added to the time indexed model. Therefore, we solve a new assignment model

(AM$''(\Omega')$) considering only the activities that belongs to the subset I. In this particular case, we define a new set of activity patterns Ω', which is still related to all the possible combinations of workers that could be assigned to each activity involved in a possible starting time. We fix a starting time t_i for each activity A_i included in D. Additionally, we define the domain (time-windows) of the starting time t_j of each activity A_j in I ($A_j \notin D$) as follows:

$$es_j = t_i - p_j + 1 \quad \forall i \in D, \forall j \in D^i, j \notin D \tag{5.27}$$

$$ls_j = t_i + p_i - 1 \quad \forall i \in D, \forall j \in D^i, j \notin D \tag{5.28}$$

$$es_j \leq t_j \leq ls_j \quad \forall i \in D, \forall j \in D^i, j \notin D \tag{5.29}$$

This previous definition enforces the overlapping in the execution between each activity A_i in D and the ones in D^i. Additionally, we introduce a new parameter β_i^ω which takes the value of t if activity A_i starts at time t in activity pattern ω, or takes the value of 0 otherwise. All the parameters related to this model are build only for the activities in I considering the time-windows previously defined.

Now, the resulting model can be stated as:

$$\sum_{\omega=0}^{\Omega'} (x_\omega \cdot \alpha_i^\omega) = 1 \quad \forall i \in I \tag{5.30}$$

$$\sum_{\omega=0}^{\Omega'} (x_\omega \cdot \beta_i^\omega) = t_i \quad \forall i \in I \tag{5.31}$$

$$\sum_{\omega=0}^{\Omega'} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in M, \forall t \in [0, T] \tag{5.32}$$

$$t_i + p_i \leq t_j \quad \forall i \in I, \forall j \in E_i' \tag{5.33}$$

$$x_\omega \in \{0, 1\} \quad \forall \omega \in \{0, \Omega'\} \tag{5.34}$$

In this new model the starting times are considered as decision variables, thus we consider two new constraints in comparison to the model AM$'(\Omega)$. Constraint set (5.31) recovers the associated starting times, while constraint set (5.33) states the precedence relations of the activities in I (E_i').

Moreover, if the solution of AM"(Ω') is infeasible it implies that the overlapping between each activity A_i in D and the activities in each subset D_i leads to an infeasible schedule, thus, we can restrict such a situation with a new overlapping subsets cut that could be included in STIMWS. In the opposite case, we need to redefine subset D. Thus, we start an iterative procedure where we test all the possible combinations of activities that could be included in D until finding one that we can use for generating a new valid cut.

The global procedure for generating an overlapping subsets cut and adding it to STIMWS is illustrated in figure 5.7.

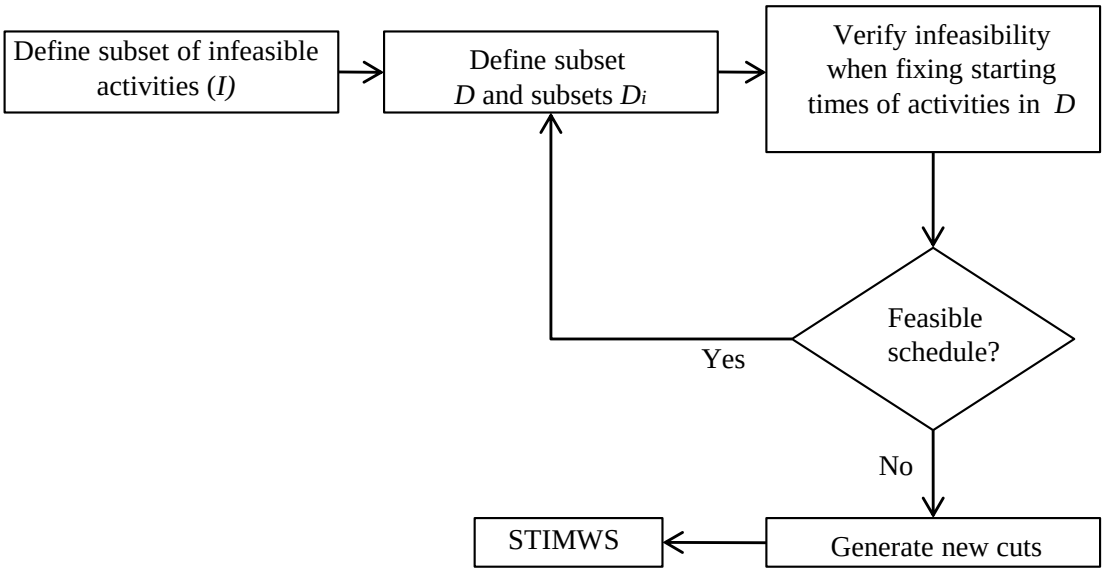

Figure 5.7: Procedure for generating an overlapping subsets cut

5.4 Branch and Bound procedure

For enforcing the performance of the proposed approach we implemented a Branch and Bound (B&B) based on a dichotomic time-window branching strategy which allows us to reduce the domain of the decision variables involved in STIMWS (see section 5.3.1). Thus, with such an approach we aim on accelerating the resolution of the time indexed model and enhancing the generation of new cuts that might lead to an optimal solution. This branching strategy was already explained in section 4.1. With this branching strategy the time-window of the starting time of a given activity A_i is divided in half obtaining two new nodes, corresponding to two new disjoint time-windows for the selected activity.

Moreover, we branch on the activity A_i that after doing a preliminary propagation on the time-windows, leads to a lower number of decision variables z_i^t included in the time indexed model. Therefore, after generating a new node we apply the two-phase solution procedure. Subsequently, we impose a time limit of 30 seconds for the resolution of STIM (first solution phase). Additionally, we also imposed a limit for the number of iterations in which the two-phase solution method may be applied in each node.

5.5 Computational results

Computational experiments were performed using the solver Gurobi OptimizerVersion 4.6. As was done for the previous solution methods, we selected a subset of the available instances for the MSPSP [93] according to their size in terms of number of activities, skills and number of workers. We consider the same set of instances we used for evaluating the Branch and Price approaches explained in section 4.1. The computational results shown in this section corresponds to instances which consider between: 20 and 62 activities,

2 and 15 skills, and 2 and 19 workers. The sizes considered by each group of tested instances (see section 2.10) are reminded as follows:

– Group 1: We studied 110 instances from this group, considering between: 20 and 51 activities, between 2 and 8 skills, and between 5 and 14 workers.
– Group 2: Regarding this group of instances, we include results for 71 instances, which consider between: 32 and 62 activities, 9 and 15 skills, and 5 and 19 workers.
– Group 3: We studied 90 instances from this group, considering between: 22, and 32 activities, 3 and 12 skills, and 4 and 15 workers.

We also tested instances with bigger sizes, but in general terms we were not able to obtain optimal solutions beyond the sizes previously mentioned.

We performed several experiments for evaluating the performance of the methods proposed in this chapter. Therefore, initially we discuss the results related to the resolution of a single iteration of the first phase of the proposed solution approach, which leads to an initial lower bound. Subsequently we introduce the results related to the execution of the two-phase solution approach. Finally, we show the results related to the B&B approach introduced in section 5.4.

Initially, as was already mentioned, in table 5.1 we introduce results related to the execution of a single iteration of the first phase of the proposed solution approach. These results allows us to evaluate the impact on the cumulative and cardinality cuts included initially in the STIMWS for obtaining a strong initial lower bound (LB_0). Thus, first we compare the gap between all the possible cumulative cuts that we could generate and all the dominant cuts, which are the ones that we add to the ILP. Subsequently we introduce the gap against the best known lower bounds (BLB) obtained by Bellenguez-Morineau and Néron [16]. Thereafter, we show the number of instances for which the best known lower bounds were improved, and the number of instances for which the obtained bound was equal to the best known lower bound. In addition, we introduce the number of instances for which the optimal values were previously unknown and that we were able to prove as optimal by comparing the obtained lower bound with the best known upper bound (UB). Finally, we present the average computational time invested for obtaining this initial lower bounds. It is important to outline that we imposed a time limit of ten minutes for calculating LB_0.

Now, results shown in table 5.1 are based only on the 90, 60 and 88 instances, for each group of problems respectively, for which it was possible to obtain a lower bound before the imposed time limit of ten minutes. Hence, we can state that in average, for each group of instances, we are able to obtain a positive deviation against the best known lower bounds, which implies that the lower bound obtained by our proposed approach can be stronger than the one obtained by Bellenguez-Morineau and Néron [16]. In addition, we are able to improve the best known lower bounds for 54, 19 and 83 instances for each of the tested group of instances, investing in average less than 30 seconds per instance. Additionally, we prove optimality for 33, 9 and 52 instances for which the optimal values

	Group of instances		
	Group 1	**Group 2**	**Group 3**
% of filtered cumulative cuts	61,07%	96,06%	75,43%
Average deviation between LB_0 and BLB	4,47%	1,29%	7,56%
Number of lower bounds improved ($LB_0 > BLB$)	54	19	83
Number of lower bounds equal to BLB ($LB_0 = BLB$)	36	40	5
Number of new optimal solutions ($LB_0 = UB$)	33	9	52
Average CPU time (sec)	21,98	26,61	15,13

Table 5.1: Summary of the obtained lower bounds results

were previously unknown. Overall, for the majority of the tested instances we were able to improve or at least reach the best known lower bound obtained by Bellenguez-Morineau and Néron [16]. Moreover, based on the presented results, we can state that when adding only the dominant cumulative cuts, there is a big impact in the percentage of filtered cuts.

Furthermore, with the purpose of achieving an optimal schedule, in table 5.2 we introduce the results obtained from applying the two-phase solution approach explained in section 5.3. Thereafter, we show the total number of obtained optimal solutions within an time limit of thirty minutes. Additionally, we show the number of instances in which we reached optimality but the optimal value was previously unknown given the lower and upper bounds founded in previous work by [16, 17]. Finally, we present the average computational times.

	Group of instances		
	Group 1	**Group 2**	**Group 3**
Total number of obtained optimal solutions	89	55	90
Number of new optimal solutions	41	8	43
Average CPU time (sec)	405,65	478,46	111,71

Table 5.2: Summary obtained results of the two-phase solution approach

Overall, results from table 5.2 show that we are able to reach optimality in 89, 55 and 90 out of the 110, 71 and 90 tested instances for each group of problems. Subsequently, for each group of problems, we also obtained optimal solutions in 57, 19 and 88 out of the 78, 31 and 88 tested instances in which the optimal solution was previously unknown. Furthermore, we can also conclude that the average computational times are acceptable (no more than 8 minutes in average among the three groups of tested instances) given the fact that we are using an exact solution method. Although, it is important to state that the magnitude of the CPU times values are affected by the thirty minutes spent in the instances in which was not possible to find the optimal solution.

More precisely, regarding the instances of the group 1 we were able to solve to optimality 82 out of 100 instances that considers between 20 and 35 activities and 7 out of 10 instances that considers 51 activities. Concerning the group 2 of instances, which is

the one that considers a larger number of activities (up to 62 activities), we were able to achieve optimality in all the instances with 32 activities and in 13 out of 29 instances with 62 activities. Thereby, regarding the instances of the group 3, we were able to solve to optimality all the instances tested. In addition, concerning this last group of instances, we also tested instances with bigger sizes (up to 60 activities), but we were not able to obtain any optimal solution. Overall, it is important to state that when considering instances with a big number of activities ($|A| \geq 60$) and with a time horizon of high magnitude ($T > 50$) it becomes more difficult to reach an optimal solution since our solution method is based on a time indexed model.

Now, in table 5.3 we compare the average percentage of the total computational time spent on both the first solution phase and the second solution phase. We also show the average number of overlapping subsets cuts added to the STIMWS after the resolution of the second phase of our solution approach. Finally, we give the average number of global iterations (first phase + second phase) until either proving optimality or reaching the imposed time limit of thirty minutes.

	Group of instances		
	Group 1	**Group 2**	**Group 3**
First phase % of total CPU time	74,88%	58,87%	64,53%
Second phase % of total CPU time	25,12%	41,13%	35,47%
Average Number of added cuts	0,85	1,84	1,03
Average Number of global iterations	1,56	2,41	1,92

Table 5.3: Performance measures of the two-phase solution approach

Results from table 5.3 show that we invested more time in the resolution of the first phase, given that in average such a procedure takes the 74,88%, 58,87% and 64,53 % of the total computational time for each group of instances. While in the other hand, the resolution of the second solution phase takes the 25,12%, 41,13% and 35,47% of the total computational time for each group of instances. Subsequently, in average, for each group of instances, we generated near to one, two and one overlapping subsets cuts per instance. Additionally, we executed in average near to two global iterations per instance for groups 1, 2 and 3 respectively. These last findings confirms that the resolution of the first iteration of the first solution phase leads to a good approximation of the starting times of the activities. The fact that the average number of global of iterations is mostly lower than 2 (at least for groups 1 and 3) allows us to assume that the starting times estimated in the first iteration could lead directly to a feasible assignment of workers. Additionally, it seems that it is not necessary to add a high number of overlapping subsets cut for reaching a feasible assignment of workers.

Moreover, considering results shown in table 5.2, we can outline that for the 21 and 12 instances of groups 1 and 2 respectively, for which we were not able to prove optimality, in 19 and 8 of such instances, we spend the whole imposed time limit of thirty minutes in the resolution of the first iteration of the first solution phase. Hence, for enforcing the

performance of the proposed approach we implemented the Branch and Bound (B&B) explained in section 5.4. Thus, in table 5.4 we introduce the results related to this last procedure. Thereafter, we show the total number of obtained optimal solutions within a time limit of thirty minutes. Additionally, we show the number of instances in which we reached optimality but the optimal value was previously unknown given the lower and upper bounds founded in previous work by [16, 17]. Finally, we present the average computational times and the average number of expanded nodes.

	Group of instances		
	Group 1	**Group 2**	**Group 3**
Total number of obtained optimal solutions	89	61	89
Number of new optimal solutions	63	23	87
Average CPU time (sec)	338,52	366,05	72,623
Number of evaluated nodes	15,79	15,31	4,08

Table 5.4: Summary obtained results

Therefore, results shown in table 5.4 state that we are able to reach optimality in 94, 60 and 89 out of the 110, 71 and 90 tested instances for each group of problems. Subsequently, for each group of problems, we also obtained optimal solutions in 63, 23 and 87 out of the 78, 31 and 88 tested instances in which the optimal solution was previously unknown. Hence, if we compare with the results obtained from applying the two-phase solution method only in the root node, we can distinguish that with such an approach we reached optimality in 234 out of the 271 instances tested. While, with the B&B approach, we were able to obtain the optimal solution in 243 instances. This last approach allowed us also to obtain 9 instances more for which the optimal solution was previously unknown.

Moreover, regarding the instances of the group 1, we were able to solve to optimality 84 out of 100 instances that considers between 20 and 35 activities and 10 out of 10 instances that considers 51 activities. Concerning the group 2 of instances, which is the one that considers a larger number of activities (up to 62 activities), we were still able to achieve optimality in all the instances with 32 activities and in 18 out of 29 instances with 62 activities. Now, regarding the instances with 62 activities, we can notice that when applying the B&B we were able to increase the number of instances solved to optimality from 13 to 18. Subsequently, regarding the instances of the group 3, we were not able to solve to optimality only one instance among all the ones that we tested. Additionally, concerning this last group of instances, we also tested instances with bigger sizes (up to 60 activities), but we were not able to obtain any optimal solution. In general terms, when using the B&B approach it is still difficult to solve instances with more than 60 activities.

Furthermore, we can also conclude that the average computational times decreased with the B&B approach (no more than 7 minutes in average among the three groups of tested instances). In addition, we can also notice that the number of average expanded nodes per instance is low (15,78; 15,30 and 4,08 for each group of instances). This last

finding is mostly justified by the fact that the majority of instances were optimally solved, hence, optimality was proven early in the process of expanding the tree. Additionally, given the imposed time limit of 30 seconds for the execution of the first solution phase, it could be possible to spend a high amount of computational time in a given node. It is important to notice that considering that we invested in average less than 30 seconds for obtaining the lower bounds shown in table 5.1, we fixed such a value as a time limit for the resolution of a single iteration of the first solution phase in each evaluated node. We also outline, that for the execution of the proposed B&B, we imposed a limit of two global iterations per node for executing the two-phase solution method. This last value is based on the average number of global iterations executed per instance when using the two-phase solution method only in the root node (see table 5.3).

5.6 Conclusion

In this chapter we proposed an iterative approach that generates and adds new cuts to an integer model for solving the MSPSP. We mainly try to deal with the hard constraints of the problem by means of cuts included in a time indexed model. Best obtained results show that the proposed procedure is able to reach optimality in 243 out of 271 studied instances in an acceptable computational time. Additionally we obtained new optimal solutions in 173 out of 197 instances for which the optimal value was previously unknown. Additionally we can conclude that we can solve to optimality several instances with up to 62 activities. Nevertheless, there are still some instances of bigger sizes, than the ones discussed in this chapter for which we were not able to obtain optimal solutions. Future work could be related to the definition of alternative cuts that could allows us to spend less time in the resolution of the time indexed model, but ensuring a good approximation of the starting times of the activities of the project. Subsequently, there could be other efficient ways for identifying the overlapping subsets cut. It could be also interesting to evaluate the impact of the proposed cuts in the LP relaxation. Hence, a cutting planes method, oriented to enforcing the LP relaxation could also be an interesting alternative, as well as the implementation of Branch and Cut approach.

6

Conclusion and Perspectives

The work done in this thesis aimed at analyzing and solving a $\mathcal{NP}$-Hard in the strong sense optimization problem, known as the Multi-Skill Project Scheduling Problem (MSPSP). Such a problem characterizes several features and components of certain real-life planning and scheduling tasks. Particularly, in a MSPSP environment, the inclusion of resources with multiple skills, and the fact that an activity may require the utilization of several skills and resources at the same time, adds a certain complexity that somehow restricts the utilization of the different methods oriented to the resolution of the all the variety of Project Scheduling problems that have been studied among the last years. Hence, initially, in this thesis, we reviewed different problems that shares certain features with the MSPSP and their respective solution methods. Thereafter, among the studied problems, the Multi-Mode Resource Constrained Project Scheduling Problem arises as an alternative for representing the components of the MSPSP by enumerating all the modes related to the execution of each activity of a given project. Nevertheless, when considering projects of certain sizes, the number of modes could be really huge, making impossible to implement the existing methods related to the resolution of the Multi-mode RCPSP.

Furthermore, after given a detailed description of the studied problem, we presented different procedures and approaches for modeling and solving the MSPSP. At first, we introduced different ILP models, based on different perspectives for representing all the components of the MSPSP. More precisely, we initially, presented three time-indexed models which share a similar modeling structures. The main difference between these ILP relies mainly in the disaggregation among the time of certain constraints, obtaining different results regarding linear relaxation and the number of optimal solutions. Additionally, we also considered two new ILP models oriented to different modeling perspectives, which, in general terms were outperformed by the time indexed models.

Subsequently, we studied Column Generation (CG) as an alternative for obtaining strong a linear relaxations without having to enumerate all the decision variables involved in an ILP for solving the MSPSP. Thus, after describing all the concepts related to CG,

we also studied the different decomposition approaches that we could explore taking into account the features of the MSPSP. Thereby, based on some theoretical arguments we selected an activity-based decomposition approach, then, we proposed two alternative restricted master problem (RMP) formulations. We also considered the utilization of Lagrangian relaxation for accelerating the resolution of the proposed RMP. In addition, we also outline that the dual information obtained after solving the RMP allowed us to estimate an assignment cost for each worker at a given time. Subsequently, in the resulting sub-problem we focus on finding a feasible assignment of workers for a given a activity at a certain starting time. Finally, we compared the linear relaxation obtained with the proposed CG approach with the ones achieved by the time-indexed models presented in the second chapter. We also evaluated the two proposed RMP formulations, and the utilization of an approach that combines Lagrangian relaxation and the simplex method for decreasing the computational times of the proposed CG approaches.

In the following chapter, we explored the use of two branching procedures for solving the MSPSP. The first one aimed at obtaining an optimal solution (B&P), while the second one was based on a heuristic approach which allowed us to solve big size instances (RBS). In addition we used one of the CG approaches proposed in the third chapter for calculating a lower bound for each evaluated node. Subsequently, when applying CG in a given node, we used the dual information obtained after solving a RMP, for developing different procedures to get a feasible schedule that leads to a makespan upper bound. Furthermore, regarding the B&P we evaluated two branching strategies, hence, we compared the respective results. In general terms, we were able to solve to optimality small and medium size instances with up to 32 activities, reaching optimality in several problems for which the optimal values were previously unknown. On the other hand, concerning the RBS, we compared different approaches based on a single branching strategy, where we were able to evaluate the impact of applying a step that allows to recover from a previous wrong decisions (recovering step). Obtained results presented a competitive performance with the state of the art available for solving the MSPSP. Finally, regarding the chronological branching strategy, we can outline that we proposed an approach that allowed us to avoid branching on the possible set of workers that could be assigned to a given activity at a certain time point, which could lead to a big increase in the number of generated nodes.

Thereafter, the last method presented in this thesis relies in the resolution of a time indexed model (STIMWS) that contains a subset of the constraints of the original formulation of the MSPSP. Hence, initially we described some inequalities that have been used in the literature for enforcing the linear relaxation of a linear program. Thereby, we identified cuts that could be added for approximating the constraints of the MSPSP that were not included in the STIMWS in the first place. Moreover, the resolution of the time indexed model along with these new inequalities allowed us to compute a strong makespan lower bound. Hence, we were able to improve the best known lower bounds for several instances at a reasonable computational time. Subsequently we proposed an iterative procedure that allowed us to detect new inequalities that could be added to the STIMWS and that eventually could guide us to an optimal solution. In general terms we were able to solve to optimality instances with up to 62 activities, reaching optimality in several problems for which the optimal values were previously unknown.

Now, at last, we can conclude, that the methods introduced in this thesis enhanced optimality for up to 173 out of 197 instances for which the optimal solution was previously unknown. Overall, we have to outline the performance of the Cut Generation approach, which was the one that allowed us to solve to optimality a greater number of instances (up to 243 out of 271 tested instances) investing a reasonable amount of computational time. It is also important to outline that we reached optimality in several instances with up to 62 activities. Therefore, despites that there are things that can still be improved, we can state that the work done in this thesis related to the use of exact methods presented in general terms a satisfactory performance in terms of number of instances solved to optimality and computational times. Additionally, we can also mention that we were able to explore different and new approaches that were not considered before for solving the MSPSP. Now, regarding the Recovering Beam Search it is important to outline, that we were able to efficiently integrate the utilization of integer linear programming with a heuristic approach obtaining competitive results with the state of the art available for solving the MSPSP. Nevertheless, regarding the methods used in this thesis, there are still interesting things to be done. For instance, concerning the utilization of Column Generation there are other decomposition approaches that could be considered. In addition, other branching strategies could also be explored, particularly in the chronological branching squeme, there are other alternatives that could be explored in which more than one activity could be added to the partial schedule of a given node. Concerning the RBS, other approaches could be considered for performing the recovering step and estimating the lower and upper bounds of a given node. Subsequently, given the good performance of the proposed Cut Generation approach, it could be possible to identify new inequalities that enhances a faster resolution of the time indexed model, but ensuring a good approximation of the starting times of the activities of the project.

Finally, we can also identify some additional features to the ones of the MSPSP, that might be interesting to study and that could capture several components that commonly takes place in a real life application. For instance, we can consider new approaches that reacts to non expected events, like the absence of an operator, a sudden interruption of an activity, etc. Additionally, another relevant issue could be related to the planning stage of the project, where estimating the number and types of resources required to execute a project may be essential elements in the decision making process. Other optimization criteria may be considered, in particular minimizing the lateness or the tardiness given the interest in the context of Project Scheduling. Additionally, there are other objectives based on costs assignment related to resources and/or activities and/or time that could also be taken into account as it is done in the Multi-mode RCPSP.

Bibliography

[1] D. Applegate, R. Bixby, V. Chvatal, and W. Cook. *The traveling salesman problem: a computational study*. Princeton University Press, 2007.

[2] C. Artigues, S. Demassey, and E. Néron. *Resource-constrained project scheduling*. Wiley Online Library, 2008.

[3] E. Balas. Facets of the knapsack polytope. *Mathematical Programming*, 8(1):146–164, 1975.

[4] E. Balas, S. Ceria, G. Cornuéjols, and N. Natraj. Gomory cuts revisited. *Operations Research Letters*, 19(1):1–9, 1996.

[5] P. Baptiste, C. Le Pape, and W. Nuijten. Satisfiability tests and time-bound adjustmentsfor cumulative scheduling problems. *Annals of Operations Research*, 92:305–333, 1999.

[6] P. Baptiste, C. Le Pape, and W. Nuijten. *Constraint-based scheduling: applying constraint programming to scheduling problems*, volume 39. Springer Netherlands, 2001.

[7] F. Barahona and R. Anbil. The volume algorithm: producing primal solutions with a subgradient method. *Mathematical Programming*, 87(3):385–399, 2000.

[8] F. Barahona and D. Jensen. Plant location with minimum inventory. *Mathematical Programming*, 83(1):101–111, 1998.

[9] J. Bard and H. Purnomo. Preference scheduling for nurses using column generation* 1. *European Journal of Operational Research*, 164(2):510–534, 2005.

[10] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations research*, pages 316–329, 1998.

[11] M. Bazaraa, H. Sherali, and C. Shetty. *Nonlinear programming: theory and algorithms*. LibreDigital, 2006.

[12] J. Beasley. Lagrangian relaxation. In *Modern heuristic techniques for combinatorial problems*, pages 243–303. John Wiley & Sons, Inc., 1993.

[13] S. Begur, D. Miller, and J. Weaver. An integrated spatial dss for scheduling and routing home-health-care nurses. *Interfaces*, pages 35–48, 1997.

[14] J. Beliën and E. Demeulemeester. On the trade-off between staff-decomposed and activity-decomposed column generation for a staff scheduling problem. *Annals of Operations Research*, 155(1):143–166, 2007.

[15] O. Bellenguez-Morineau. Methods to solve multi-skill project scheduling problem. *4OR: A Quarterly Journal of Operations Research*, 6(1):85–88, 2008.

[16] O. Bellenguez-Morineau and E. Néron. Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. *Practice and Theory of Automated Timetabling V: 5th International Conference, PATAT 2004,Pittsburgh, PA, USA, revised selected papers*, pages 229–243, 2005.

[17] O. Bellenguez-Morineau and E. Néron. Tabu search for the multi-skill project scheduling problem. *to appear, Journal of Operations and Logistics*, 2011.

[18] S. Bertels and T. Fahle. A hybrid setup for a hybrid scenario: combining heuristics for the home health care problem. *Computers & Operations Research*, 33(10):2866–2890, 2006.

[19] D. Bertsekas. *Nonlinear programming*. Athena Scientific Belmont, MA, 1999.

[20] R. Bixby, J. Gregory, I. Lustig, R. Marsten, and D. Shanno. Very large-scale linear programming: a case study in combining interior point and simplex methods. *Operations Research*, pages 885–897, 1992.

[21] D. Bredström and M. Rönnqvist. A branch and price algorithm for the combined vehicle routing and scheduling problem with synchronization constraints. *Discussion Papers*, 7, 2007.

[22] P. Brucker, A. Drexl, R. Möhring, K. Neumann, and E. Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3–41, 1999.

[23] P. Brucker, B. Jurisch, and A. Krämer. Complexity of scheduling problems with multi-purpose machines. *Annals of Operations Research*, 70:57–73, 1997.

[24] P. Brucker and S. Knust. Lower bounds for resource-constrained project scheduling problems. *European Journal of Operational Research*, 149(2):302–313, 2003.

[25] R. Busacker and P. Gowen. A procedure for determining a family of minimum-cost-flow patterns. *Operations Research Office Technical Reports*, 15, 1961.

[26] J. Carlier and B. Latapie. Un méthode arborescente pour résoudre les problèmes cumulatifs. *RAIRO. Recherche opérationnelle*, 25(3):311–340, 1991.

[27] Y. CHANG, H. MATSUO, and S. ROBERT. A bottleneck-based beam search for job scheduling in a flexible manufacturing system. *THE INTERNATIONAL JOURNAL OF PRODUCTION RESEARCH*, 27(11):1949–1961, 1989.

[28] Z. Chen and W. Powell. A column generation based decomposition algorithm for a parallel machine just-in-time scheduling problem. *European Journal of Operational Research*, 116(1):220–232, 1999.

[29] N. Christofides, R. Alvarez-Valdés, and J. Tamarit. Project scheduling with resource constraints: A branch and bound approach. *European Journal of Operational Research*, 29(3):262–273, 1987.

[30] W. Cook. Fifty-plus years of combinatorial integer programming. *50 Years of Integer Programming 1958-2008*, pages 387–430, 2010.

[31] J. Cordeau, G. Laporte, F. Pasin, and S. Ropke. Scheduling technicians and tasks in a telecommunications company. *Journal of Scheduling*, pages 1–17, 2010.

[32] G. Cornuéjols. Valid inequalities for mixed integer linear programs. *Mathematical Programming*, 112(1):3–44, 2008.

[33] I. Correia, L. Lourenço, and F. Saldanha-da Gama. Project scheduling with flexible resources; formulation and inequalitites. *OR Spectrum*, 34:635–663, 2012.

[34] H. Crowder, E. Johnson, and M. Padberg. Solving large-scale zero-one linear programming problems. *Operations Research*, 31(5):803–834, 1983.

[35] G. Dantzig. Discrete-variable extremum problems. *Operations Research*, 5(2):266–288, 1957.

[36] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, pages 393–410, 1954.

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

[38] S. De and A. Lee. Flexible manufacturing system (fms) scheduling using filtered beam search. *Journal of Intelligent Manufacturing*, 1(3):165–183, 1990.

[39] F. Della Croce, M. Ghirardi, and R. Tadei. Recovering beam search: Enhancing the beam search approach for combinatorial optimization problems. *Journal of Heuristics*, 10(1):89–104, 2004.

[40] F. Della Croce and V. T'kindt. A recovering beam search algorithm for the one-machine dynamic total completion time scheduling problem. *Journal of the Operational Research Society*, 53(11):1275–1280, 2002.

[41] S. Demassey, C. Artigues, and P. Michelon. Constraint-propagation-based cutting planes: An application to the resource-constrained project scheduling problem. *INFORMS Journal on computing*, 17(1):52–65, 2005.

[42] E. Demeulemeester and W. Herroelen. A branch-and-bound procedure for the multiple resource-constrained project scheduling problem. *Management science*, pages 1803–1818, 1992.

[43] G. Desaulniers, J. Desrosiers, and M. Solomon. *Column generation*, volume 5. Springer-Verlag New York Inc, 2005.

[44] A. Dohn, E. Kolind, and J. Clausen. The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, 36(4):1145–1157, 2009.

[45] A. Ernst, H. Jiang, M. Krishnamoorthy, B. Owens, and D. Sier. An annotated bibliography of personnel scheduling and rostering. *Annals of Operations Research*, 127(1):21–144, 2004.

[46] B. Esteve, C. Aubijoux, A. Chartier, and V. T'kindt. A recovering beam search algorithm for the single machine just-in-time scheduling problem. *European Journal of Operational Research*, 172(3):798–813, 2006.

[47] P. Eveborn, P. Flisberg, and M. Ronnqvist. Laps care–an operational system for staff planning of home care. *European Journal of Operational Research*, 171(3):962–976, 2006.

[48] C. Fábián. Bundle-type methods for inexact data. *Central European Journal of Operations Research*, 8:35–55, 2000.

[49] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. Vehicle routing with time windows and split deliveries. *Proceedings of Odysseus*, 2003.

[50] M. Fırat and C. Hurkens. An improved mip-based approach for a multi-skill workforce scheduling problem. *Journal of Scheduling*, pages 1–18, 2011.

[51] M. Fisher. The lagrangian relaxation method for solving integer programming problems. *Management science*, pages 1861–1871, 2004.

[52] M. Fox. Constraint-directed search: A case study of job-shop scheduling. Technical report, DTIC Document, 1983.

[53] B. Franck, K. Neumann, and C. Schwindt. Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling. *OR Spectrum*, 23(3):297–324, 2001.

[54] M. Garey, D. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of operations research*, pages 117–129, 1976.

[55] S. Gélinas and F. Soumis. Dantzig-Wolfe Decomposition for Job Shop Scheduling. *Column Generation*, pages 271–302, 2005.

[56] A. Geoffrion. Lagrangean relaxation for integer programming. *Approaches to Integer Programming*, pages 82–114, 1974.

[57] M. Ghirardi and C. Potts. Makespan minimization for scheduling unrelated parallel machines: A recovering beam search approach. *European Journal of Operational Research*, 165(2):457–467, 2005.

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

[59] J. Goffin and J. Vial. Convex nondifferentiable optimization: A survey focused on the analytic center cutting plane method. *Optimization Methods and Software*, 17(5):805–867, 2002.

[60] R. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64(5):275–278, 1958.

[61] R. Gomory. An algorithm for the mixed integer problem. Technical report, DTIC Document, 1960.

[62] M. Grötschel and M. Padberg. On the symmetric travelling salesman problem i: inequalities. *Mathematical Programming*, 16(1):265–280, 1979.

[63] W. Gutjahr, S. Katzensteiner, P. Reiter, C. Stummer, and M. Denk. Competence-driven project portfolio selection, scheduling and staff assignment. *Central European Journal of Operations Research*, 16(3):281–306, 2008.

[64] S. Hartmann and D. Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010.

[65] C. Heimerl and R. Kolisch. Scheduling and staffing multiple projects with a multi-skilled workforce. *OR spectrum*, 32(2):343–368, 2010.

[66] M. Held and R. Karp. The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, 1(1):6–25, 1971.

[67] M. Held, P. Wolfe, and H. Crowder. Validation of subgradient optimization. *Mathematical programming*, 6(1):62–88, 1974.

[68] S. Ho and D. Haugland. A tabu search heuristic for the vehicle routing problem with time windows and split deliveries. *Computers & Operations Research*, 31(12):1947–1964, 2004.

[69] S. Hong. *A linear programming approach for the traveling salesman problem*. PhD thesis, Johns Hopkins University, 1972.

[70] J. Hooker and G. Ottosson. Logic-based benders decomposition. *Mathematical Programming*, 96(1):33–60, 2003.

[71] D. Huisman, R. Jans, M. Peeters, and A. Wagelmans. Combining column generation and lagrangian relaxation. *Column generation*, pages 247–270, 2005.

[72] I. Ioachim, J. Desrosiers, F. Soumis, and N. Belanger. Fleet assignment and routing with schedule synchronization constraints. *European Journal of Operational Research*, 119(1):75–90, 1999.

[73] V. Jain and I. Grossmann. Algorithms for hybrid milp/cp models for a class of optimization problems. *INFORMS Journal on computing*, 13(4):258–276, 2001.

[74] R. Jans and Z. Degraeve. An industrial extension of the discrete lot-sizing and scheduling problem. *IIE Transactions*, 36(1):47–58, 2004.

[75] K. Jansen and L. Porkolab. Polynomial time approximation schemes for general multiprocessor job shop scheduling* 1. *Journal of Algorithms*, 45(2):167–191, 2002.

[76] B. Jaumard, F. Semet, and T. Vovor. A generalized linear programming model for nurse scheduling. *European Journal of Operational Research*, 107(1):1–18, 1998.

[77] H. Jiang, M. Krishnamoorthy, and D. Sier. Staff scheduling and rostering: theory and applications, Part I and II. *Annals of Operational Research*, 128:1–4, 2004.

[78] E. Johnson, G. Nemhauser, and M. Savelsbergh. Progress in linear programming-based algorithms for integer programming: An exposition. *INFORMS Journal on Computing*, 12(1):2–23, 2000.

[79] E. Johnson, M. Padberg, et al. Degree-two inequalities, clique facets and biperfect graphs. 1981.

[80] B. Kallehauge. Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers & Operations Research*, 35(7):2307–2330, 2008.

[81] S. Kesen, S. Das, and Z. Güngör. A genetic algorithm based heuristic for scheduling of virtual manufacturing cells (vmcs). *Computers & Operations Research*, 37(6):1148–1156, 2010.

[82] R. Kolisch and S. Hartmann. 7 heuristic algorithms for the resource-constrained project scheduling problem: Classification and computational analysis. *Project scheduling: Recent models, algorithms, and applications*, 14:147, 1999.

[83] R. Kolisch and R. Padman. An integrated survey of deterministic project scheduling. *Omega*, 29(3):249–272, 2001.

[84] R. Kolisch and A. Sprecher. Project scheduling problem library–psplib. *línea]< http://www. bwl. uni-kiel. de/Prod/psplib/>. Consulta*, 13, 2000.

[85] H. Li and K. Womer. Scheduling projects with multi-skilled personnel by a hybrid milp/cp benders decomposition algorithm. *Journal of Scheduling*, 12(3):281–298, 2009.

[86] B. Lowerre. The harpy speech recognition system. 1976.

[87] M. Lübbecke and J. Desrosiers. Selected topics in column generation. *Operations Research*, pages 1007–1023, 2005.

[88] H. Marchand, A. Martin, R. Weismantel, and L. Wolsey. Cutting planes in integer and mixed integer programming. *Discrete Applied Mathematics*, 123(1):397–446, 2002.

[89] A. Mason and M. Smith. A nested column generator for solving rostering problems with integer programming. In *International conference on optimisation: techniques and applications*, pages 827–834. Citeseer, 1998.

[90] A. Mehrotra, K. Murphy, and M. Trick. Optimal shift scheduling: A branch-and-price approach. *Naval Research Logistics (NRL)*, 47(3):185–200, 2000.

[91] 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*, pages 714–729, 1998.

[92] T. Morton and D. Pentico. *Heuristic scheduling systems: with applications to production systems and project management*, volume 3. Wiley-Interscience, 1993.

[93] E. Néron and D. Baptista. Heuristics for multi-skill project scheduling problem. *International Symposium on Combinatorial Optimization (CO'2002)*, 2002.

[94] E. Néron and J. Carlier. Du flow-shop hybride au probleme cumulatif. 1999.

[95] P. Ow and T. Morton. Filtered beam search in scheduling. *The International Journal Of Production Research*, 26(1):35–62, 1988.

[96] P. Ow and T. Morton. The single machine early/tardy problem. *Management Science*, pages 177–191, 1989.

[97] P. Ow and S. Smith. Viewing scheduling as an opportunistic problem-solving process. *Annals of Operations Research*, 12(1):85–108, 1988.

[98] M. Padberg and G. Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[99] M. Padberg, T. Van Roy, and L. Wolsey. Valid linear inequalities for fixed charge problems. *Operations Research*, 33(4):842–861, 1985.

[100] J. Patterson, R. Slowinski, F. Talbot, and J. Weglarz. An algorithm for a general class of precedence and resource constrained scheduling problems. *Advances in project scheduling*, pages 3–28, 1989.

[101] B. Polyak. A general method of solving extremum problems. In *Soviet Mathematics Doklady*, volume 8, pages 593–597, 1967.

[102] A. Pritsker, L. Watters, and P. Wolfe. Multiproject scheduling with limited resources: A zero-one programming approach. *Management Science*, pages 93–108, 1969.

[103] S. Ropke and J. Cordeau. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science*, 43(3):267–286, 2009.

[104] L. Rousseau, M. Gendreau, G. Pesant, and Q. Centre for Research on Transportation (Montréal. *The synchronized vehicle dispatching problem*. Citeseer, 2003.

[105] I. Sabuncuoglu and M. Bayiz. Job shop scheduling with beam search. *European Journal of Operational Research*, 118(2):390–412, 1999.

[106] I. Sabuncuoglu and S. Karabuk. A beam search-based algorithm and evaluation of scheduling approaches for flexible manufacturing systems. *IIE transactions*, 30(2):179–191, 1998.

[107] M. Salani and I. Vacca. Branch and Price for the Vehicle Routing Problem with Discrete Split Deliveries and Time Windows. *European Journal of Operational Research*, 2011.

[108] M. Savelsbergh. Preprocessing and probing techniques for mixed integer programming problems. *ORSA Journal on Computing*, 6(4):445–454, 1994.

[109] M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45(6):831–841, 1997.

[110] P. Toth and D. Vigo. *The vehicle routing problem*, volume 9. Society for Industrial Mathematics, 2002.

[111] J. Valente and R. Alves. Beam search algorithms for the single machine total weighted tardiness scheduling problem with sequence-dependent setups. *Computers & Operations Research*, 35(7):2388–2405, 2008.

[112] V. Valls, Á. Pérez, and S. Quintanilla. Skilled workforce scheduling in service centres. *European Journal of Operational Research*, 193(3):791–804, 2009.

[113] J. van den Akker, G. Diepen, and J. Hoogeveen. A column generation based destructive lower bound for resource constrained project scheduling problems. *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 376–390, 2007.

[114] J. Van Den Akker, C. Hurkens, and M. Savelsbergh. Time-indexed formulations for machine scheduling problems: Column generation. *INFORMS Journal on Computing*, 12(2):111–124, 2000.

[115] M. Van den Akker, H. Hoogeveen, and S. de Velde. Combining column generation and Lagrangean relaxation to solve a single-machine common due date problem. *INFORMS Journal on Computing*, 14(1):37, 2002.

[116] L. Wolsey. Faces for a linear inequality in 0–1 variables. *Mathematical Programming*, 8(1):165–178, 1975.

[117] L. Wolsey. *Integer programming*. Wiley, New York, 1998.

[118] L. Wolsey. Integer programming. *IIE Transactions*, 32:273–285, 2000.

[119] A. Zanette, M. Fischetti, and E. Balas. Lexicography and degeneracy: can a pure cutting plane algorithm work? *Mathematical programming*, 130(1):153–176, 2011.

7

NOUVELLES METHODES POUR LE PROBLEME DE GESTION DE PROJET MULTI-COMPETENCE

7.1 Introduction

La gestion des ressources joue un rôle important dans la compétitivité d'une entreprise. En particulier, la bonne gestion des ressources humaines est essentiel pour les organisations. Subséquemment, l'ordonnancement et la répartition des tâches entre les ressources disponibles sont fonctions qui ont lieu sur une situation quotidienne dans toute organisation. Ainsi, les problèmes d'ordonnancement apparaissent dans tous les domaines de l'économie. Citons la construction, l'informatique, l'industrie et l'administration. Dans la majorité des tâches de planification et d'ordonnancement, les contraintes plus dures sont liées aux ressources limitées. Pour cette raison, l'allocation des ressources est un élément important dans diverses tâches liées à l'ordonnancement et à la planification dans un environnement réel. Les méthodes utilisées pour effectuer ces tâches doivent être conçues pour fournir une bonne utilisation des ressources, en tenant compte de sa capacité, coût et disponibilité. Il existe différents environnements de production qui traitent les aspects mentionnés précédemment. Dans cette thèse, nous nous concentrons dans un cas particulier, qui implique la utilisation des ressources dans un environnement de gestion des projets. Ces caractéristiques sont considérées par le problème d'ordonnancement de projet à moyens limités (RCPSP) [22].

Le RCPSP considère un projet avec un certain nombre d'activités qui doivent être affectées à un ensemble de ressources. Il existe entre l'activités des relations de précédence, qui signifient classiquement qu'une tâche ne peut débuter que si certaines autres tâches sont finies. Ces relations permettent de représenter l'organisation du projet par un graphe de précédence. L'intérêt pour l'extension des applications pratiques liées au RCPSP,

a conduit à envisager des extensions différentes comprenant certaines variantes liées à des situations réelles [22, 83, 93, 2, 64]. Dans le RCPSP ont été considérés de critères d'optimisation différents, parmi lesquels la minimisation du makespan a été largement considérée.

Lorsque vous traitez avec des ressources humaines, il existe plusieurs types de entreprises dans lequel l'utilisation des membres du personnel qui maîtrise plusieurs compétences sont requises. Subséquemment, nous nous concentrons sur une extension particulière du RCPSP, qui est connu comme le problème de gestion de projet multi-compétence (MSPSP). Ce problème a été initialement proposé par Néron et Baptista [93]. Dans le MSPSP les ressources sont des membres du personnel qui maîtrise plusieurs compétences. Ainsi, un certain nombre de travailleurs doit être affecté pour utiliser chaque compétence requise par une activité. L'objectif est trouver un ordonnancement qui minimise le makespan. Les applications pratiques peuvent être liés aux centres d'appels, la construction de bâtiments et le développement de logiciels.

Par ailleurs, nous accorderons une importance particulière aux méthodes exactes pour résoudre le MSPSP, puisqu'il y a encore un certain nombre d'instances pour lesquelles l'optimalité doit encore être prouvée. Les meilleurs résultats ont été obtenus à ce jour par Bellenguez-Morineau et Néron [17] avec la utilisation d'une approche heuristique. En outre, ces derniers auteurs ont également trouvé des bornes inférieures pour les différentes instances disponibles. Donc, nous proposons de nouvelles méthodes pour résoudre le MSPSP en s'appuyant sur la programmation linéaire. Dans cette thèse nous avons développé, en particulier des méthodes associées à la génération de colonnes, relaxation lagrangienne, génération des coupes et des méthodes arborescentes et des méthodes heuristiques de type beam-search.

7.2 Description du problème et formulation mathématique

Le MSPSP est un problème d'ordonnancement de projets, qui est principalement composé de trois éléments: activités, ressources et compétences. Ces éléments sont détaillés cidessous:

Activités

Un projet est composé d'un ensemble d'activités $A = \{A_0,\ ldots, A_N\}$. Dans cet ensemble, on définit aussi deux activités fictives A_0 et A_N pour représenter le début et la fin du projet, respectivement. Il existe entre l'activités des relations de précédence, qui signifient classiquement qu'une tâche ne peut débuter que si certaines autres tâches sont finies [64]. Ainsi, chaque activité A_i dispose d'un ensemble correspondant de successeurs E_i^+ et de prédécesseurs E_i^-. Ces relations permettent de représenter l'organisation du projet par un graphe de précédence (G) où une activité est représenté par un noeud et la relation de précédence entre deux activités est représentée par un arc orienté (A_i, A_j) où

$A_j \in E_i^+$ [83]. Cet arc représente également la durée minimale du temps entre la date du début de A_i et le début de un de ses successeurs directs. Par la suite, ce durée de temps est indiquée par p_i, qui correspond également au durée opératoire de lactivité A_i.

Toutes les activités doivent être planifiées afin que la durée totale du temps du projet (makespan) soit minimisée. La date de début d'une activité A_i est notée par t_i, donc sa date de finalisation sera donné par $t_i + p_i$. Les activités sont supposées non-préemptives: une fois que l'exécution a commencé, elles se poursuit sans interruption jusqu'à la fin.

Ressources et compétences

Dans le MSPSP les ressources sont personnes, donc, nous pouvons nous assurer que les ressources sont des moyens de production renouvelables, mais limités en capacité. Chacune des personnes affectées maîtrise une ou plusieurs compétences parmi celles nécessaires aux activités du projet. Nous considérons un ensemble W de M personnes et un ensemble S de K compétences. Ensuite, pour chaque personne W_m ($W_m \in W$) et chaque compétence S_k ($S_k \in S$), on a $MS_{m,k} = 1$ si P_m maîtrise S_k, et 0 sinon.

Par la suite, un certain nombre de personnes doit être affecté à chacune des compétences requises pour effectuer une activité donnée. Les contraintes liées à cette notion de compétence sont alors [93]:

- Pour chaque activité A_i et chaque compétence S_k, il existe une donnée $b_{i,k}$ qui est égale au nombre de personnes qui devront exercer la compétence S_k lors de l'exécution de l'activité A_i, $b_{i,k}$ est nul si la compétence S_k n'est pas nécessaire à A_i.
- Une personne ne peut exercer qu'une compétence qu'elle maîtrise, i.e. si $MS_{m,k} = 1$.
- Une personne W_m ne peut faire qu'une seule chose à un instant donné t.
- Chaque personne choisie pour répondre à un besoin de l'activité A_i durant son exécution lui est affectée durant tout l'intervalle de temps ($[t_i, t_i + p_i[$) et ne peut assurer aucun travail sur cet intervalle de temps.

Par la suite, un exemple est présente ci-dessous afin d'illustrer le type de problème que l'on peut traiter. Cet exemple est basé sur un projet contenant 4 activités, 4 personnes et 3 compétences. On retrouvera dans le tableau 7.1 la description de chaque compétence. Le tableau 7.2 contient les besoins des différents activités de ce projet. Puis, dans le tableau 7.3 sont résumées les compétences maîtrises par chacune des personnes qui ont été affectées au projet. Ensuite, la figure 7.1 représenté le graphe de précédence du projet correspondant à cet exemple. Enfin, la 7.2 présente un des ordonnancement réalisable pour ce problème.

S_0	Programmeur
S_1	Designer de base de données
S_2	Webmaster

Table 7.1: Liste des compétences

	S_0	S_1	S_2
A_1	-	1	1
A_2	1	-	1
A_3	2	1	-
A_4	-	1	-

Table 7.2: Besoins des activités

	S_0	S_1	S_2
W_0	-	1	1
W_1	1	-	1
W_2	1	-	-
W_3	1	-	1

Table 7.3: Compétences maitrisées par chaque personne ($MS_{m,k}$)

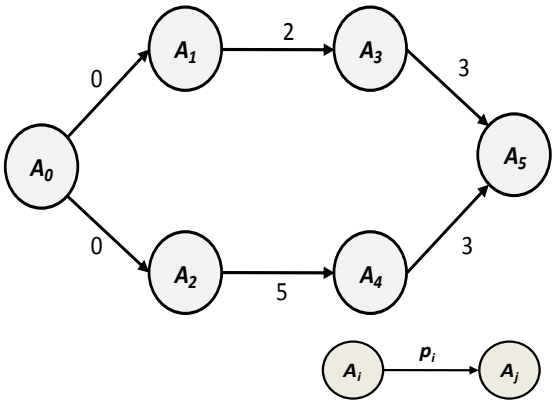

Figure 7.1: Graphe G

Notez que A_0 et A_5 sont des activités supplémentaires fictives qui représente le début et la fin du projet, respectivement. Le poids sur les arcs représente la durée opératoire de chaque activité.
Enfin, en tenant compte que le RCPSP classique est déjà un problème d'optimisation $\mathcal{NP}$-difficile, donc on peut déduire que le MSPSP est $\mathcal{NP}$-difficile au sens fort [2]. De ce fait, ces deux problèmes sont équivalents si chaque personne maîtrise seulement une compétence.

7.2.1 Travaux antérieurs relatifs à MSPS

Malgré le fait que la notion de compétences joue un rôle important dans le domaine de l'affectation du personnel [77], il n'est pas régulièrement pris en compte dans le domaine

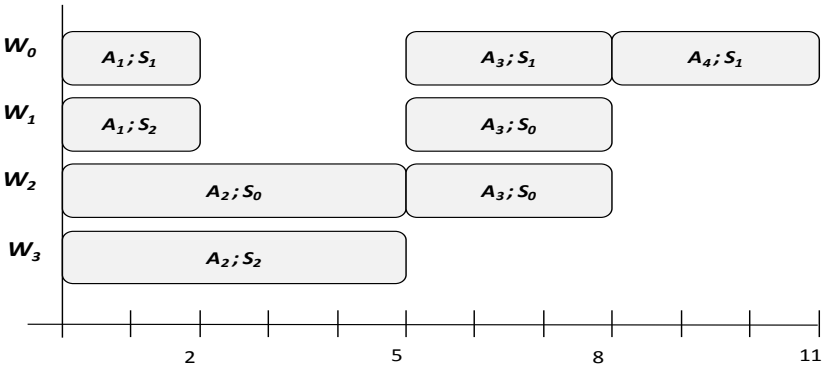

Figure 7.2: Exemple d'une solution optimale

de la gestion de projet. Ainsi, concernant le MSPSP, on peut souligner le travail effectué par Bellenguez-Morineau et Néron [15, 16, 17], qui ont développé différentes procédures pour déterminer des bornes inférieures et supérieures pour le makespan. En outre, Cordeau et al. [31] ont développé une heuristique constructive pour résoudre le problème du planification des techniciens et des interventions pour les télécommunications. Pour résoudre ce dernier problème, plus récemment, Firat et Hurkens [50] ont développé une méthode de résolution qui utilise un modèle basé sur un modèle de programmation en nombres entiers.

En outre, il existe des méthodologies intéressantes dans la littérature de la gestion de projet qui considère des ressources humaines qui maîtrise plusieurs compétences. Par exemple, Heimerl et Kolisch [65] ont proposée un programme linéaire mixte en nombres entiers pour résoudre un problème multi-projet où le ordonnancement de chaque projet est déjà fixée. Li et Womer [85] ont développé un algorithme hybride basé sur la modélisation entier mixte et la programmation par contraintes pour résoudre un problème d'ordonnancement de projet avec personnel polyvalent. Gutjahr et al. [63] ont proposée une heuristique gloutonne et une méthode hybride basée sur l'utilisation de règles de priorité, d'un algorithme de colonie de fourmis et d'un algorithme génétique pour résoudre un problème liée a la sélection et ordonnancement des projets. Plus récemment, Correia et al. [33] ont présenté une formulation linéaire mixte en nombres entiers et plusieurs inégalités supplémentaires pour une variante du RCPSP où les ressources sont flexibles, c'est à dire, chaque ressource maîtrise plusieurs compétences.

7.2.2 Données utilisées

Les instances disponibles pour le MSPSP ont été générés par Bellenguez-Morineau [15] et sont basées sur des graphes de précédence préexistants qui proviennent dinstances de RCPSP. Les instances considérées sont réparties en trois groupes:

– Groupe 1: Se compose d'instances basée sur les graphes des instances proposées par Baptiste et al. [5], Patterson et al. [100] et Néron [94]. Nous avons pour ce groupe 185

instances, avec un nombre d'activités entre 8 et 50. Le nombre de compétences a été généré aléatoirement entre 3 et 8. Le nombre de personnes varie quant à lui entre 5 et 22.

– Groupe 2: Contient 174 instances proviennent dinstances de RCPSP classique de la PSPlib [82] et de Kolisch et Sprecher [84]. Nous avons pris des graphes contenant 30, 60 et 90 activités. Le nombre de compétences a été généré aléatoirement entre 9 et 18. Les personnes affectées au projet sont entre 5 et 30.
– Groupe 3: Se compose de 198 instances basée sur des graphes provenant d'instances de RCPSP multi-mode de la PSPlib [82] et de Kolisch and Sprecher [84]. Ils contiennent 12, 14, 16, 18, 20, 22, 32, 62 et 92 activités. Le nombre de compétences a été généré aléatoirement entre 3 et 12. Le nombre de personnes varie quant à lui entre 4 et 15.

7.2.3 Programme linéaire en nombres entiers

Afin de formaliser ce problème, nous avons proposé cinq modèles de programmation linéaire en nombres entiers (PLNE), dont trois basés sur la indexation du temps. Dans ces trois modèles, la variable principale est un variable booléenne à trois paramètres: le temps, la activité et le ressource. Il s' agit d'une adaptation du modèle développé par Pritsker et al. [102] pour résoudre le RCPSP. Après avoir comparé les performances de chaque modèle, ici nous montrons le plus représentatif (TIMWS). Ce modèle a été aussi inspirée de celui proposé par Bellenguez-Morineau et N´eron (2005). Les variables de décision sont définis par:

$x_{i,m}^t$ 1 la personne W_m commence lactivité A_i à la date t, 0 sinon;
$y_{i,m}^k$ 1 si la personne W_m exerce la compétence S_k lors de lexécution de lactivité A_i, 0 sinon;
z_i^t 1 si lactivité A_i commence à la date t, 0 sinon.

Compte tenu de l'utilisation de z_i^t, nous pouvons représenter le temps de début d'une activité A_i $commesuit$:

$$t_i = \sum_{t\in[0,T]} (z_i^t \cdot t) \quad \forall i \in A \tag{7.1}$$

Par la suite, ci-dessous, nous présentons la formulation mathématique associée:

$$Z[TIMWS] : Min\ C_{max} = t_N \tag{7.2}$$

$S.t.$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E_i^+ \tag{7.3}$$

$$es_i \leq t_i \leq ls_i \quad \forall i\ A \tag{7.4}$$

$$\sum_{t\in[0,T]} x_{i,m}^t \leq 1 \quad \forall i \in A, \forall m \in W \tag{7.5}$$

$$\sum_{i \in A} \sum_{d \in [t-p_i+1,t]} x_{i,m}^{d} \leq 1 \quad \forall t \in [0,T], \forall m \in W \tag{7.6}$$

$$x_{i,m}^{t} \leq z_i^t \quad \forall i \in A, \forall m \in W, \forall t \in [0,T] \tag{7.7}$$

$$x_{i,m}^{t} + 1 \geq z_i^t + \sum_{k \in S} y_{i,m}^{k} \quad \forall i \in A, \forall m \in W, \forall t \in [0,T] \tag{7.8}$$

$$y_{i,m}^{k} \leq MS_m^k \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{7.9}$$

$$\sum_{m \in W} y_{i,m}^{k} = b_i^k \quad \forall i \in A, \forall k \in S \tag{7.10}$$

$$\sum_{t \in [0,T]} x_{i,m}^{t} = \sum_{k \in S} y_{i,m}^{k} \quad \forall i \in A, \forall m \in W \tag{7.11}$$

$$z_i^t \in \{0,1\} \quad \forall i \in A, \forall t \in [0,t] \tag{7.12}$$

$$x_{i,m}^{t} \in \{0,1\} \quad \forall i \in A, \forall m \in W, \forall t \in [0,T] \tag{7.13}$$

$$y_{i,m}^{k} \in \{0,1\} \quad \forall i \in A, \forall m \in W, \forall k \in S \tag{7.14}$$

La fonction objectif (7.2) est de minimiser le makespan du projet. L'ensemble des contraintes (7.3) représente la relation de précédence entre les activités. L'ensemble des contraintes (7.4) indique que le temps de début de chaque activité doit être comprise dans une fenêtre de temps prédéfinie. L'ensemble des contraintes (7.5) précise qu'une activité ne peut commencer quune fois au maximum. L'ensemble des contraintes (7.6) indique qu'une personne peut commencer une activité au maximum une fois. Les ensembles de contraintes (7.7) et (7.8) précise la synchronisation de la date de début de tous les personnes affectés à une activité. L'ensemble des contraintes (7.9) indique qu' une personne ne peut exercer quune compétence qu'elle maîtrise. L'ensemble des contraintes (7.10) précise que pour chaque activité les besoins en compétences doivent être satisfaits. L'ensemble des contraintes (7.11) indique qu'une personne doit exercer une compétence pour une activité à laquelle elle participe. Enfin, les ensembles de contraintes (7.12), (7.13) et (7.14) précise que les variables de décision sont binaires.

En ce qui concerne la contrainte (2.4), il est important de noter que es_i (resp. ls_i) désigne une borne inférieure (resp. supérieure) pour la date de début associé à l'activité A_i. Cette fenêtre de temps $(es_i; ls_i)$ est par exemple tout simplement induite par le graphe de précédence, et une borne supérieure donnée (UB) pour le makespan. Par conséquent, les fenêtres de temps pour chaque activité $A_i \forall i \in A$ sont initialement définis comme suit:

Les plus petite dates de début (es_i) sont calculés de la façon suivante:

$$es_0 = 0$$
$$es_i = max_{\forall j \in E_i^-}\{es_j + p_j\} \quad \forall i \in A$$

Les plus grand dates de début (ls_i) sont calculés de la façon suivante:

$$ls_N = UB$$

$$ls_i = min_{\forall j \in E_i^+}\{ls_j - p_i\} \quad \forall i \in A$$

Il est important de mentionner que l'horizon de planification (T) a été défini comme égal à une borne supérieure (UB) calculée avec la méthode tabou développé par Bellenguez-Morineau and Néron [17].

7.3 Bornes inferieures avec Génération de colonnes

Dans cette section, nous étudions et proposons une méthode de génération de colonnes (CG), qui est une procédure qui consiste à résoudre itérativement un programme linéaire (RMP) selon certains critères d'arrêt. Plus précisément, dans la CG, nous décomposons le problème en plusieurs sous-problèmes qui contiennent moins de contraintes, qui peuvent être résolus plus efficacement et indépendamment les uns des autres [10]. Enfin, nous présentons des résultats expérimentaux, dans lequel on compare la relaxation linéaire obtenu avec la génération de colonnes contre la relaxation linéaire obtenu par le PLNE présenté dans la section précédente.

7.3.1 Génération de colonnes

Travaux antérieurs relatifs à la Génération de colonnes

Jusqu'à présent, la génération de colonnes (CG) n'a pas été utilisée pour résoudre spécifiquement le problème de gestion de projet multi-compétence (MSPSP). Néanmoins, il a été utilisé en combinaison avec d'autres techniques d'optimisation pour résoudre d'autres types de problèmes d'ordonnancement de projet. En particulier, Brucker et Knust [22] ont développé une approche pour trouver des bornes inférieures pour le RCPSP en utilisant des techniques de propagation par contraintes et une méthode de CG. En outre, Van den Akker et al. cite van2007 ont présenté une borne inférieure basée sur une approche par génération de colonnes pour résoudre certaines extensions du RCPSP.

D'autre part, la CG a été largement utilisé sur le problème de tournées de véhicules (VRP) et plusieurs extensions associées [72, 103, 49, 21, 80, 107, 44] qui partage des caractéristiques similaires avec le MSPSP.

En outre, la génération de colonnes a également été utilisé pour résoudre différent problèmes d'ordonnancement du personnel [76, 90, 14, 9, 89]. Enfin, la CG a également été utilisé pour résoudre d'autres types de problèmes d'ordonnancement comme par exemple le Job Shop Scheduling Problem [28, 114, 115, 55].

Introduction à la génération de colonnes et décomposition du problème

La génération de colonnes est une méthode pour résoudre efficacement les programmes linéaires de grande taille. Elle repose sur la décomposition de Dantzig-Wolfe [37], qui consiste à décomposer l'ensemble des contraintes en deux sous-ensembles: le problème maître (MP) y le sous-problème (MP). L'idée centrale est que les programmes linéaires

de grande taille ont trop de variables (ou colonnes) pour qu'on puisse les représenter toutes de manière explicite. A l'optimum, la plupart des variables sont hors base et, très souvent, la plupart d'entre elles sont nulles, c'est-à-dire que seul un (petit) sous-ensemble de variables doit être pris en compte pour résoudre le problème. Une méthode utilisant la génération de colonnes initialise le programme linéaire avec un sous-ensemble de colonnes de petite taille. Le mécanisme de la génération de colonnes consiste alors à générer, au sein d'un algorithme à plusieurs étapes, les variables qui sont susceptibles d'améliorer la solution courante, c'est-à-dire celles qui ont des coûts réduits négatifs.

Notez que la décomposition du problème initial en un problème maître et un sous-problème est possible grâce à l'exploitation d'une structure spécifique de la formulation du orignal problème où les sous-problème conduit à une problème d'optimisation plus " facile " comme par exemple le problème de plus court chemin ou le problème du sac à dos. Cela implique que décider comment décomposer un problème particulier jouera un rôle important pour obtenir des solutions efficaces [14].

Pour le MSPSP, la décomposition peut être fait sur les ressources ou les activités (tâches). Dans ce cas, le sous- problème consiste à trouver un ordonnancement réalisable pour une seule ressource (véhicule, travailleur, etc)[14, 103, 49, 113, 44, 55]. D'autre part, la décomposition sur les activités consiste à trouver une affectation réalisable des personnes pour une seule activité. Compte tenu des caractéristiques du MSPSP, la décomposition sur les ressources peut conduire à un problème maître plus difficile à résoudre, par conséquent, dans notre approche de génération de colonnes, nous considérons une décomposition sur les activités.

7.3.2 Approche proposée avec l'utilisation de la génération de colonnes

Nous avons considéré une approche de décomposition sur les activités, dans lequel nous avons comparé deux formulations des problèmes de maître, qui conduisent à la même sous-problème. Nous présentons ici celle qui nous a permis d'obtenir de meilleurs résultats.

Problème Maître (MP)

L'idée de base de notre approche de génération de colonnes est basée sur une reformulation indexée sur le temps du problème. Dans cette nouvelle formalisation mathématique, une colonne ω décrit les attributs de l'exécution d'une activité A_i. Ensuite, une colonne ω est représenté par un triplet $[A_i(\omega), t(\omega), W(\omega)]$ ou $A_i(\omega)$ indique l'activité liée à ω, $t(\omega)$ représente sa date de début, et $W(\omega)$ indique le sous-ensemble des opérateurs affectés à A_i. Nous supposons que les personnes affectés à ω répondre aux besoins en compétences de l'activité associée. Plus précisément, on peut définir les paramètres correspondants de la manière suivante:

Paramètres

α_i^ω 1 l'activité A_i est traitée sur la colonne ω, 0 sinon;

β_i^ω t i l'activité A_i commence à la date t sur la colonne ω, 0 sinon;

$\gamma^{\omega}_{m,t}$ 1 si la personne W_m est affectée à la colonne ω à la date t, 0sinon.

En outre, on note Ω comme l'ensemble de toutes les colonnes possibles. Les variables de décision liées au modèle proposé sont définies par:

Variables

x_{ω} 1 si la colonne ω est choisi, 0 sinon;

t_i Date de début de l'activité A_i;

La formulation mathématique associée est définie de la manière suivante:

$$Z[MP] : Min\ t_N \tag{7.15}$$

$$S.t.$$

$$\sum_{\omega\in[0,\Omega]} (x_{\omega} \cdot \alpha^{\omega}_i) = 1 \quad \forall i \in A \tag{7.16}$$

$$\sum_{\omega\in[0,\Omega]} (x_{\omega} \cdot \beta^{\omega}_i) = t_i \quad \forall i \in A \tag{7.17}$$

$$\sum_{\omega\in[0,\Omega]} (x_{\omega} \cdot \gamma^{\omega}_{m,t}) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{7.18}$$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E^{+}_i \tag{7.19}$$

$$es_i \leq t_i \leq ls_i \quad \forall i\ A \tag{7.20}$$

$$x_{\omega} \in \{0,1\} \quad \forall \omega\ \in [0,\Omega] \tag{7.21}$$

La fonction objectif (7.15) est minimiser le makespan du projet. L'ensemble des contraintes (7.16) indique que seulement une colonne peut être affecté à chaque activité A_i. L'ensemble des contraintes (7.17) récupère les dates de début associés. L'ensemble des contraintes (7.18) précise que tout opérateur peut effectuer au plus une activité à un moment donné. L'ensemble des contraintes (7.19) représente la relation de précédence entre les activités. Enfin, l'ensemble des contraintes (7.20) indique que le temps de début de chaque activité doit être comprise dans une fenêtre de temps prédéfinie.

En outre, le problème maître(MP) est obtenue en relâchant les contraintes liées aux variables de décision x_{ω}.

Définition du problème maître restreint (RMP)

Compte tenu de la formulation du problème maître, pour tout ensemble partiel de colonnes $\bar{\Omega} \subseteq \Omega$ nous pouvons définir le problème maître restreint($RMP(\bar{\Omega})$) de la manière suivante:

$$Z[RMP[\bar{\Omega}]] : Min\ t_N + (L \cdot \sum_{i\in A} s_i) + (L \cdot \sum_{i\in A} u_i) \tag{7.22}$$

$$S.t.$$

$$\sum_{\omega\in[0,\bar{\Omega}]} (x_\omega \cdot \alpha_i^\omega) + s_i = 1 \quad \forall i \in A \tag{7.23}$$

$$\sum_{\omega\in[0,\bar{\Omega}]} (x_\omega \cdot \beta_i^\omega) + u_i = t_i \quad \forall i \in A \tag{7.24}$$

$$\sum_{\omega\in[0,\bar{\Omega}]} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{7.25}$$

$$t_i + p_i \leq t_j \quad \forall i \in A, \forall j \in E_i^+ \tag{7.26}$$

$$es_i \leq t_i \leq ls_i \quad \forall i \in A \tag{7.27}$$

$$x_\omega \geq 0 \quad \forall \omega \in [0,\bar{\Omega}] \tag{7.28}$$

Dans cette formulation, u_i et s_i sont variables d'écart positifs qui sont ajouter au MP afin de garantir la faisabilité d'une sélection partielle de colonnes. Comme L est défini comme une constante positive, une solution réalisable est garanti si les variables d'écart sont égales à zéro.

En supposant qu'une solution optimale du $RMP(\bar{\Omega})$ a été calculé avec un solveur standard, les multiplicateurs correspondants (variables duales) aux contraintes (7.23),(7.24),(7.18) sont définis comme suit:

π_i	Variables duales associées à l'ensemble de contraintes (7.23);
λ_i	Variables duales associées à l'ensemble de contraintes (7.24);
μ_m^t	Variables duales associées à l'ensemble de contraintes (7.25).

Par la suite, le coût réduit associé à une colonne donnée $\bar{\omega}$ liée à l'exécution d'une activité A_i à la date de début t, sont définies de la manière suivante:

$$r_{i,t} = 0 - \pi_i - (\lambda_i \cdot t) - \sum_{m\in W} \sum_{\theta\in[0,T]} (\mu_m^\theta \cdot \gamma_{m,\theta}^\omega) = r_{i,t}^1 + r_{i,t}^2 \tag{7.29}$$

Où:

$$r_{i,t}^1 = -\pi_i - (\lambda_i \cdot t) \tag{7.30}$$

$$r_{i,t}^2 = - \sum_{m\in W} \sum_{\theta\in[0,T]} (\mu_m^\theta \cdot \gamma_{m,\theta}^\omega) \tag{7.31}$$

Sous-problème (SP)

Étant donné que l'activité A_i commence à l'instant t, chaque personne affecté doit travailler pendant les instants $t, t+1, .., t+pi-1$, donc le coût total de affectation de une personne W_m est égal à:

$$\sigma_m(t) = - \sum_{\theta=t}^{t+p_i-1} \mu_m^\theta \tag{7.32}$$

Avant de présenter le sous-problème (SP), il faut définir les variables de décision de la façon suivante:

Variables

y_m 1 si la personne W_m est affectée à l'activité A_i, 0 sinon;

z_m^k 1 si la personne W_m exerce la compétence S_k lors de l'exécution de l'activité A_i, 0 sinon .

Par la suite, trouver une nouvelle colonne relative à l'exécution d'une activité A_i, qui commence au temps t, avec un coût minimal réduite, conduit au sous-problème suivante:

Formulation du Sous-problème (SP)

$$Z[SP] : Min\ r_{i,t}^2 = \sum_{m \in W} (\sigma_m(t) \cdot y_m) \tag{7.33}$$

$$S.t.$$

$$\sum_{m \in W} z_m^k = b_{i,k} \quad \forall k \in S \tag{7.34}$$

$$y_m = \sum_{k \in S} z_m^k \quad \forall m \in W \tag{7.35}$$

$$y_m \in \{0,1\} \quad \forall m \in W \tag{7.36}$$

$$z_m^k \in \{0,1\} \quad \forall m \in W, \forall k \in S \tag{7.37}$$

Dans cette formulation, l'objectif est minimiser le coût total de trouver une affectation des personnes pour effectuer l'activité A_i à un instant t. L'ensemble des contraintes (7.34) indique que les besoins en compétences doivent être satisfaits. L'ensemble des contraintes (7.35) garantit qu'une personne affectée utilise une seule compétence. En fin l'ensemble des contraintes (7.36) and (7.37) précise que les variables de décision sont binaires.

En outre, après l'obtention de la valeur de $r_{i,t}^2$ nous pouvons calculer le coût réduit ($r_{i,t} = r_{i,t}^1 + r_{i,t}^2$) associé à une colonne donnée. Par conséquent, si $r_{i,t} < 0$, alors, la colonne correspondant est candidat à entrer dans la base car son coût réduit négative diminuera la fonction objective du RMP($\bar{\Omega}$) actuel. Par conséquent, cette colonne peut être ajoutée à l'ensemble actuel des colonnes en définissant:

$$\bar{\Omega} \leftarrow \bar{\Omega} \cup \bar{\omega} \tag{7.38}$$

$$\alpha_i^{\bar{\omega}} = 1 \tag{7.39}$$

$$\beta_i^{\bar{\omega}} = t \tag{7.40}$$

$$\gamma_{m,t}^{\bar{\omega}} = y_m \quad \forall m \in W, \forall \theta \in [t, t + p_i - 1] \tag{7.41}$$

Résolution du sous- problème (SP)

Avec le sous-problème (SP) on cherche à résoudre un problème d'affectation à coût minimum que l'on modélise à l'aide du graphe F_c [17], présenté sur la figure 7.3. Ce graphe se compose d'un premier étage de sommets $S_k \in S$ correspondant aux différentes compétences S_k requises par l'activité A_i que l'on cherche à ordonnancer à la date de début t, puis d 'un second étage contenant un sommet $W_m \in W$ pour chaque personne qui maîtrise au moins une des compétences requises par l'activité A_i.

Ce graphe comporte un arc entre le sommet source et un sommet $S_k \in S$ dont la capacité maximum est égale à $b_{i,k}$, le nombre de personnes requises par A_i pour exercer la compétence S_k. Il existe un arc entre un sommet $S_k \in S$ et un sommet $W_m \in W$ si la personne W_m maîtrise la compétence S_k. La capacité maximum de cet arc est alors de 1 car une personne ne peut répondre qu' à une unité de besoin. De même, il existe un arc entre un sommet $W_m \in W$ et le puits du graphe, dont la capacité est égale à 1, car une personne ne peut être affectée qu'à un seul besoin à un instant donné. Aux arcs de ce dernier type, on associé également un coût unitaire de affectation $\sigma_m(t)$ pour chaque personne W_m.

Nous utilisons ensuite l'algorithme de Busacker et Gowen [25] afin de rechercher un flot maximum à coût minimum dans ce graphe, dont on peut minimiser le coût total d'affectation $r^2_{i,t}$.

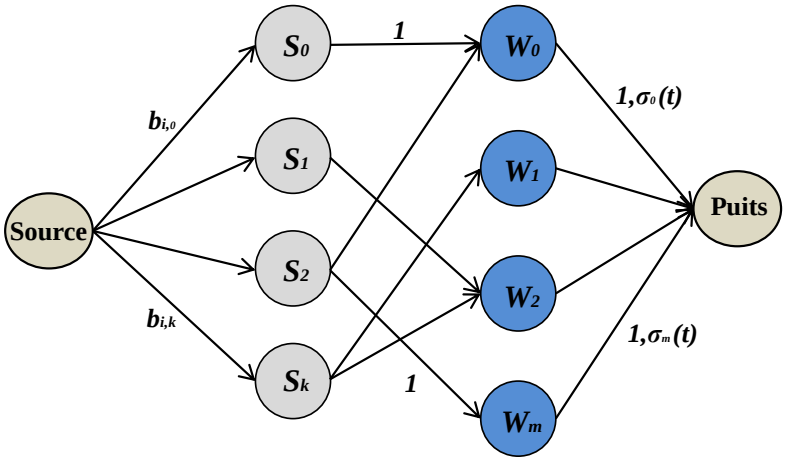

Figure 7.3: Graphe F_c: Affectation des compétences pour l'activité A_i.

Initialisation de l'ensemble de colonnes

Pour la première itération de la méthode CG, il faut initialiser le sous-ensemble de colonnes $\bar{\Omega}$ pour résoudre RMP($\bar{\Omega}$), selon l'ordonnancement obtenu par la recherche Tabou (TS) développée par [17].

7.3.3 Résolution du problème maître restreint (RMP)

Au départ, nous utilisons la méthode du simplexe pour résoudre la relaxation linéaire résultant du problème maître restreint (RMP). Une autre approche utilisée est de résoudre la relaxation lagrangienne qui consiste à relaxer certaines contraintes et à pénaliser leur violation en insérant un terme dans la fonction objectif. Par conséquent, nous avons développé une approche de relaxation lagrangienne pour accélérer la résolution du RMP. Par la suite, nous utilisons la méthode de sous-gradient [66] pour obtenir les multiplicateurs de Lagrange, qui nous permet d'estimer les variables duales nécessaires pour pouvoir calculer le coût réduit associé à une colonne.

7.3.4 Resultats Experimentaux

Les tests expérimentaux ont été effectués avec le solveur Gurobi Optimizer Version 4.6. Nous avons considéré un sous-ensemble de 271 instances de celles présentées dans la section 7.2.2. Les instances considérées sont réparties en trois groupes:

- Groupe 1: Nous avons étudié 110 instances de ce groupe, en tenant compte: entre 20 et 51 activités, entre 2 et 8 compétences, et entre 5 et 14 personnes.
- Groupe 2: En ce qui concerne ce groupe des instances, nous incluons les résultats pour 71 instances qui considèrent entre: 32 et 62 activités, 9 et 15, les compétences et les 5 et 19 personnes.
- Groupe 3: Dans cette section, nous avons étudié 90 instances qui considère entre: 22 et 32 activités, 3 et 12 compétences, et les 4 et 15 personnes.

Par la suite, dans le tableau 7.4 on compare la relaxation linéaire obtenu avec la génération de colonnes contre la relaxation linéaire obtenu par le PLNE présenté dans la section précédente (TIMWS). Pour l'approche de génération de colonnes, nous comparons les résultats obtenus en utilisant uniquement la méthode du simplexe (CG) pour résoudre le RMP contre les résultats obtenus en utilisant la relaxation lagrangienne (CGLR) pour résoudre le RMP. Par ailleurs, au départ, nous introduisons l'écart moyen entre la borne inférieure obtenue avec chaque modèle évaluées contre les plus connus bornes inférieures (BLB) obtenus par Bellenguez-Morineau et Néron [16]. Notez que les écarts ont été calculés par: $(LB - BLB)/BLB$), où LB représente la borne inférieure obtenue avec TIMWS, CG et CGLR. Par la suite, nous comparons également les temps moyens de calcul requis par chaque modèle testé pour l'obtention de leur respectif borne inférieure.

		Groupes d'instances		
		Groupe 1	**Groupe 2**	**Groupe 3**
Déviation moyenne entre LB et BLB	TIMWS	-36,87%	-7,12%	-25,21%
	CG	-10,80%	-4,96%	-6,17%
	$CGLR$	-10,80%	-4,96%	-6,17%
Temps de calcul moyen (s)	TIMWS	13,78	10,19	3,25
	CG	11,37	9,88	5,10
	$CGLR$	7,07	7,97	3,42
Nombre de colonnes générées	CG	738,05	1181,65	1817,77
	$CGLR$	1181,19	1645,07	2571,8

Table 7.4: Comparaison entre TIMWS, CG et CGLR

Les résultats présentés dans le tableau 7.4 nous permet d'affirmer que, pour chacun des groupes des instances testé, CG et CGLR sont en mesure d'obtenir une relaxation linéaire plus forte que celui obtenu par TIMWS. En outre, on peut en effet constater que l'utilisation de relaxation lagrangienne nous a permis d'accélérer la résolution du problème maître restreint (RMP).

7.4 Branch and price et Recovering Beam Search

Dans cette section, nous présentons deux méthodes de recherche arborescent. Dans le premier on effectue une recherche exhaustive (exact), tandis que dans le deuxième méthode, nous explorons seulement un certain nombre de noeuds (heuristique). Les deux méthodes proposées sont basées sur l'utilisation de la génération de colonnes (CG) décrite dans la section précédente pour estimer la borne inférieure d'un noeud donné. Dans un premier temps, nous présentons l'approche exacte, qui consiste en l'élaboration d'une procédure de Branch and Bound, qui est communément connu sous le nom de Branch and Price (B&P), compte tenu de l'utilisation de la CG pour obtenir une borne inférieure pour chaque noeud évalué. Par la suite, nous avons aussi exploré différentes stratégies de branchement et des stratégies pour réduire la taille de l'arbre de recherche. De ce fait, nous introduisons les résultats obtenus. En outre, nous introduisons une approche de recherche arborescent connu comme Recovering Beam Search approach, dans lequel on explore qu'une partie du parcours arborescent. La solution est construite chronologiquement, les bornes utilisant la génération de colonnes. Enfin, nous présentons les expériences et les résultats de calcul respectifs.

7.4.1 Branch and Price (B&P)

Introduction

Comme nous l'avons déjà mentionné, B&P combine l'utilisation de la CG avec une procédure de Branch and Bound. Par conséquent, il est important de rappeler au lecteur que l'application de génération de colonnes implique la résolution itérative d'un programme linéaire (RMP). Maintenant, il est important de noter que la solution du RMP répond à toutes les contraintes d'un problème maître, sauf pour les contraintes d'intégralité. Dans le cas où la relaxation linéaire du problème maître ne conduit pas à une solution entière optimale, une stratégie de branchement doit être appliqué pour diriger la solution dans l'intégralité. Différentes stratégies ont été mises au point de branchement [109]. Habituellement, la ramification se fait sur les variables du problème initial ou sur les variables du sous- problème. Dans la plupart des cas, une ramification du type 1-0 sur les variables du problème maître n'est pas conseillé [10]. Selon la structure du MSPSP et l'approche de génération de colonnes proposée, nous avons exploré deux stratégies de branchement. La première consiste à séparer sur l'intervalle de temps d'exécution de chaque activité et la deuxiéme sur l'ordre de leurs exécution [2].

Stratégies de branchement

Fenêtre de temps dichotomique

Cette stratégie de branchement a également été utilisé dans une procédure de Branch-and-Bound proposé par Bellenguez-Morineau et Néron citebellenguez pour résoudre le

MSPSP. Il est basé sur la réduction de la fenêtre de temps (es_i, ls_i) d'une activité donnée par le biais d'une recherche dichotomique, inspiré de Carlier et Latapie [26]. Ainsi, selon certains critères une activité A_i est sélectionnée. Par la suite, la fenêtre de temps de la date de début de la activité A_i est divisé en deux pour obtenir deux nouveaux noeuds, correspondant à deux nouvelles fenêtres de temps disjoints pour l'activité sélectionnée. La figure suivante montre un exemple pour générer deux nouveaux noeuds, après de l'application de la génération de colonnes sur un noeud donné et après avoir sélectionné une activité A_i.

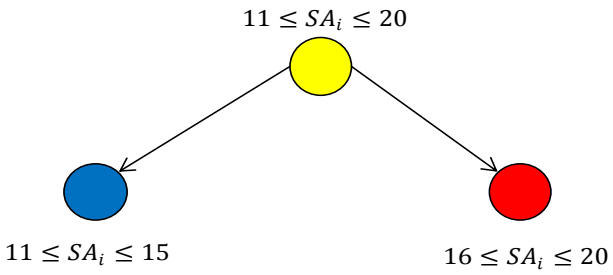

Figure 7.4: Example of the dichotomic time-windows branching strategy.

Le critère pour sélectionner le activité A_i a un impact primordial sur le performance de l'algorithme de Branch-and-Price, par conséquent, plusieurs options ont été envisagées. Des tests préliminaires effectués sur un sous-ensemble prédéfini des instances montrent que les deux critères suivants, conduisent à une meilleure performance en termes de nombre d'instances résolues à l'optimalité dans un temps limite de 30 minutes:

1. Activité avec le plus grand nombre de ressources en conflit avec l'exécution d'autres activités. Ces conflits apparaissent lorsqu'il existe un sous-ensemble d'activités qui, en fonction de leur fenêtre de temps actuelle, peuvent être exécutées simultanément. Néanmoins, leur traitement en parallèle n'est pas possible en raison de la capacité des ressources disponibles.
2. Activité qui mène à la plus grande réduction dans le fenêtres de temps entre toutes les activités: Une telle réduction peut simplement être évalué avec une propagation préliminaire sur le graphe de précédence.

Séparation chronologique

Dans cette stratégie de ramification, un nouveau noeud comprend l'addition d'au moins une activité A_j à un ordonnancement partiel. Étant donné qu'il existe plusieurs types de systèmes chronologiques de branchement [2], nous avons d'abord exploré la possibilité d'ajouter seulement qu'une activité à un ordonnancement partiel. Dans ce schéma de branchement, d'abord, nous définissons un ensemble d'activités admissibles EL composée par l'ensemble des activités disponibles qui ont déjà eu ses prédécesseurs ordonnancés. Par conséquent, un noeud est créé pour chaque activité A_j dans EL. Cette activité

est ajoutée dès que possible, à un instant t en respectant les contraintes de précédence et de ressources. Remarquez, que cette procédure considère toutes les activités disponibles pour construire des nouvelles ordonnancements partiels.

La génération d'un nouveau ordonnancement partiel implique fixer des dates de début des activités concernées. Par conséquent, l'affectation des ressources correspondant est modélisé par la recherche dun flot maximum à coût minimum (voir figure 7.5). Si nous n'obtenons pas une affectation réalisable des personnes, nous résolvons le problème maître décrit dans la section 7.3.2, mais nous ne considérons que les activités incluses dans le ordonnancement partiel actuel et on fixe leurs respectives dates de début.

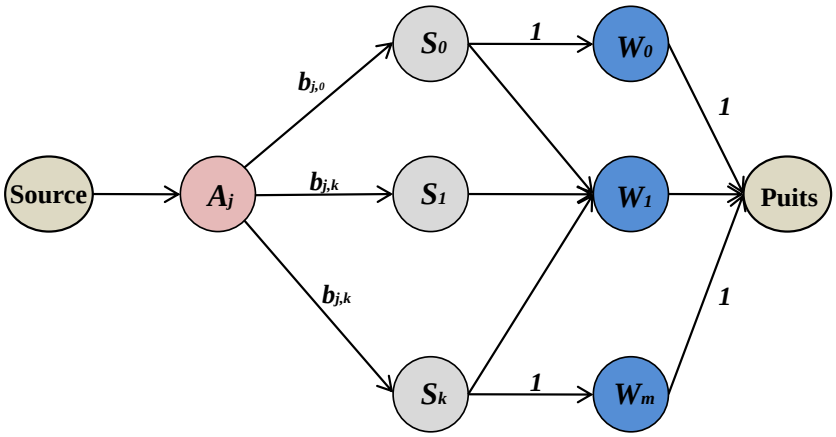

Figure 7.5: Graph F_a skills assignment for the candidate activity A_j without considering an assignment cost for each worker.

Bornes

L'utilisation des bornes inférieures et supérieures est un élément clé pour réduire l'espace de recherche lors de l'application de la procédure de Branch-and-Price. Par conséquent, nous proposons dans cette section différentes bornes inférieures et supérieures, lesquelles sont présentés comme suit:

Propagation des temps fenêtres et des bornes inférieures avec la génération de colonnes: Pour créer un nouveau noeud nous mettons à jour les fenêtres de temps pour les dates de début des activités en propageant sur le graphe de précédence. Le makespan résultant est une borne inférieure de la durée totaled du projet. Par la suite, la résolution du RMP donne une meilleure borne inférieure pour un noeud donné.

Borne inférieure basée sur le graphe de compatibilité: Cette borne inférieure est basé sur la notion de graphe de compatibilité (G_c) et il a été proposé par Mingozzi et al. [91] et adaptée pour le MSPSP par Bellenguez-Morineau et N'eron [16].

Borne supérieure basée sur la résolution d'un programme linéaire en nombres entiers (PLNEUB): Cette approche consiste d'abord à résoudre le RMP associée à un noeud N de l'arbre de recherche, et après le processus de génération de colonnes a convergé, en ré-intégrant la contrainte d'intégralité sur les variables x_ω liées au sous-ensemble actuel de colonnes Ω. La solution de cette PLNE permet d'obtenir une borne supérieure pour le noeud actuel.

Borne supérieure basée sur une liste de priorités sur les coûts réduits (RCPL): Cette borne supérieure est obtenue en appliquant un algorithme de liste et est effectué après la fin du processus de la génération de colonnes associé au noeud actuel. Il est obtenue avec la construction d'une liste d'activités admissibles EL, composée par l'ensemble des activités disponibles qui ont déjà eu ses prédécesseurs ordonnancés. La première activité disponible dans EL est sélectionné pour être exécutée dès que possible. Par la suite, l'affectation de personnes à un instant t est déterminée en appliquant le même algorithme déjà utilisé pour résoudre le sous-problème (SP). Quand il ya plusieurs activités qui pourraient être ordonnancés à un instant t, nous sélectionnons l'activité et le groupe des personnes qui conduit à une coût réduit $r_{i,t}$ inférieure.

Stratégies pour réduire l'espace de recherche

Différentes stratégies ont été développées pour réduire l'espace de recherche:

Borne inférieure basée sur le graphe de compatibilité: Cette stratégie consiste à comparer la borne inférieure basée sur le graphe de compatibilité avec la meilleure borne supérieure courante UB. Par conséquent, on peut savoir si le noeud courante peut permettre d'atteindre une solution meilleure que la meilleure solution connue, dans le cas contraire le noeud est coupé.

Interdire l'exécution en parallèle de certaines paires des activités: Après avoir généré un nouveau noeud et mettre à jour les fenêtres de temps de toutes les activités, nous vérifions si l'exécution d'un couple d'activités doit se faire simultanément. Par conséquent, si l'exécution en parallèle de ces activités n'est pas possible en raison de la capacité des ressources disponibles, la réalisation du projet avec les fenêtres de temps courantes n'est pas réalisable, donc nous pouvons couper le noeud généré.

Mise à jour de la relations de précédence: Après avoir généré un nouveau noeud, nous identifions les paires d'activités qui, en principe, pourraient être faites en parallèle, mais en raison de leur fenêtres de temps et de la capacité des ressources, une activité doit être effectué après l'autre (ce qui implique la création d'une nouvelle relation de précédence entre ces activités).

Utilisation de la génération de colonnes: Un noeud est coupée si la borne inférieure obtenue par la génération de colonnes est plus grand ou égal à la borne supérieure courant.

Resultats Experimentaux

Les tests expérimentaux ont été effectués avec le solveur Gurobi Optimizer Version 4.6. Nous avons considéré un sous-ensemble de 271 instances de celles présentées dans la section 7.2.2. Les groups d'instances considérées sont rappelé comme suit:

- Groupe 1: Nous avons étudié 110 instances de ce groupe, en tenant compte: entre 20 et 51 activités, entre 2 et 8 compétences, et entre 5 et 14 personnes.
- Groupe 2: En ce qui concerne ce groupe des instances, nous incluons les résultats pour 71 instances qui considèrent entre: 32 et 62 activités, 9 et 15, les compétences et les 5 et 19 personnes.
- Groupe 3: Dans cette section, nous avons étudié 90 instances qui considère entre: 22 et 32 activités, 3 et 12 compétences, et les 4 et 15 personnes.

Nous avons également testé des instances avec de plus grandes tailles, mais nous n'avons pas obtenu des solutions optimales au-delà des dimensions mentionnées précédemment.

Maintenant, avant de présenter tous les résultats obtenus, nous rappelons que nous avons appliqué l'approche de relaxation lagrangienne pour résoudre le problème maître restreint (RMP). Les résultats présentés dans la section 7.3.4 reflète que l'utilisation de la relaxation lagrangienne nous a permis d'accélérer la résolution du problème maître restreint (RMP).

En outre, le tableau 7.5 montrent des résultats qui comparent le Branch and Price (B&P) avec le schéma de séparation de fenêtre de temps dichotomique (B&PTW) contre le B&P avec le schéma de séparation chronologique (B&PCB). Cette comparisson se fait en termes de temps de calcul moyen, nombre de solutions optimales obtenues dans un temps limite de 30 minutes et l'écart moyen entre le meilleur borne supérieure (UB) et la borne inférieure (LB) finale obtenus avec B&PTW et B&PCB. Les écarts ont été calculés avec $(UB - LB)/LB$.

		Groupes d'instances		
		Groupe 1	**Groupe 2**	**Groupe 3**
Temps de calcul moyen (s)	B&PTW	1081,07	647,17	1016,34
	B&PCB	1151,46	714,75	1195,86
Nombre de solutions optimales	B&PTW	48	47	46
	B&PCB	42	44	40
Déviation moyen entre UB et LB	B&PTW	16,86%	5,38%	8,97%
	B&PCB	17,03%	5,2%	8,7%

Table 7.5: Comparaison de performance entre B&PTW et B&PCB

Ces résultats montrent que le B&PTW nous avons obtenu la solution optimale pour 141 instances(48, 47 et 46 pour chaque groupe d'instances testées. D'autre part, avec le B&PCB, nous avons obtenu la solution optimale pour 126 instances. En outre, le B&PTW

présente une meilleure performance que B&PCB en termes de temps de calcul moyen par instance. Néanmoins les deux approches présentent un comportement similaire en termes de l'écart moyen entre UB et LB.

En outre, il est important de préciser que la grandeur des temps de calcul moyen montré dans le tableau ref tableau: BandPComparissonFrench sont influencés par les trente minutes consommées dans les instances pour lesquels n'a pas été possible de trouver la solution optimale. Ainsi, le tableau 7.6 montre les temps de calcul requis dans les 141 et 126 instances, pour lesquelles la solution optimale a été obtenue avec B&PTW et B&PCB respectivement.

		Groupes d'instances		
		Groupe 1	**Groupe 2**	**Groupe 3**
Temps de calcul moyen (s)	B&PTW	148,95	57,09	283,02
	B&PCB	166,55	57,55	218,21
Nombre de nouvelles solutions optimales	B&PTW	16	11	45
	B&PCB	10	8	39

Table 7.6: Comparaison de performance entre B&PTW et B&PCB pour les instances pour lesquelles la solution optimale a été obtenue

En outre, ce dernier tableau montre également que parmi les 271 instances testés avec B&PTW et B&PCB ont été en mesure de trouver des solutions optimales pour 72 et 57 instances où une telle solution était jusqu'à présent inconnue étant donné les bornes inférieures et supérieures trouvés par Bellenguez-Morineau et Néron [16, 17].

7.4.2 Recovering Beam Search

Dans cette section, nous introduisons une heuristique appelee Recovering Beam Search (RBS) avec lequelle nous essayons d'exploiter la structure des procédures de Branch and Price présentées dans la section précédente 7.4.1. Le recherche RBS est une amélioration d'un algorithme très connu de recherche arborescent appelé Beam Search (BS) [96]. Dans le RBS un nombre donné de noeuds sont générés à chaque niveau de l'arbre de recherche. En outre, le RBS considère une phase de recuperation des erreurs potentielles à chaque niveau de l'arborescence. D'une manière générale, la méthode proposée est une algorithme heuristique de recherche arborescent qui intègre la résolution de différents modèles mathématiques pour résoudre des instances de plus grandes tailles que ceux discutés dans les sections précédentes.

Recovering Beam search (RBS)

Le recherche RBS est une amélioration d'un algorithme très connu de recherche arborescent appelé Beam Search (BS). Lapproche de beam search (BS) est une heuristique bien établie qui est issue du monde de lintelligence artificielle [97]. La méthode repose sur une recherche arborescente tronquée couplée à une stratégie de type en largeur dabord

où seuls les w noeuds les plus prometteurs à une profondeur donnée sont effectivement explorés: w est appelé largeur de faisceau. Lévaluation des noeuds est généralement réalisée en deux temps: (i) une première phase de filtrage permet de sélectionner à un coût modeste un certain nombre de noeuds qui sont ensuite évalués de façon plus fine et w dentre eux sont conservés pour la suite de lexploration de larbre.

Linconvénient majeur de BS est quune erreur dans le processus dévaluation, conduisant à la fermeture dun noeud menant à une solution optimale ou proche de loptimum, est irréversible. La solution retournée par le processus global peut ainsi être fortement éloignée de la solution optimale dans les cas les moins favorable. le processus. Une façon de éviter cette dernière situation a été proposée par Della Croce et al. [40]. Il sagit dinsérer dans le processus global une étape de récupération (Recovering Beam Search) qui va rechercher, à chaque niveau delarbre de recherche, des solutions partielles dominantes par rapport à celles sélectionnées par le faisceau. Cette phase de récupération est réalisée par lapplication dopérateurs déchange aux solutions partielles examinées parle faisceau et la confrontation des diverses solutions ainsi obtenues entre elles. Les solutions retenues pour la poursuite de la recherche dépendront des conditions de dominance, forcément dépendantes du problèmes, définies au préalable.

Approche de recovering beam search proposée

Afin d'appliquer la RBS à l'MSPSP, il est nécessaire de spécifier leurs principales composantes: le schéma de branchement, procédure de filtrage, bornes supérieures et inférieures, et la phase de recuperation des erreurs potentielles. Le RBS proposée considère également une étape supplémentaire de réparation exécuté dans la phase de branchement. Cette étape supplémentaire est également expliqué dans le résumé suivant des principaux éléments de l'approche proposée.

Schéma de branchement

Pour ce type d'algorithmes de recherche arborescent, généralement, dans le domaine de l'ordonnancement, les stratégies de branchement sont basées sur la construction progressive d'un ordonnancement partiel. Par la suite, nous considérons une approche de branchement chronologique, dans lequel une activité est ajouté à un ordonnancement partiel pour générer un nouveau noeud [2]. Dans cette stratégie de ramification, un nouveau noeud comprend l'addition d'au moins une activité A_j à un ordonnancement partiel. Étant donné qu'il existe plusieurs types de systèmes chronologiques de branchement [2], nous avons d'abord exploré la possibilité d'ajouter seulement qu'une activité à un ordonnancement partiel. Dans ce schéma de branchement, d'abord, nous définissons un ensemble d'activités admissibles EL composée par l'ensemble des activités disponibles qui ont déjà eu ses prédécesseurs ordonnancés. Par conséquent, un noeud est créé pour chaque activité A_j dans EL. Cette activité est ajoutée dès que possible, à un instant t en respectant les contraintes de précédence et de ressources. Remarquez, que cette procédure considère toutes les activités disponibles pour construire des nouvelles ordonnancements partiels. Néanmoins, il existe d'autres procédures basées sur l'ajout d'une activité à un

ordonnancement partiel [2]. Ce schéma de branchement a également été pris en compte dans la procédure de B&PCB introduite dans la section 7.4.1.

La génération d'un nouveau ordonnancement partiel implique fixer les dates de début des activités concernées. Par conséquent, l'affectation des ressources correspondant est modélisé par la recherche dun flot maximum à coût minimum. Si nous n'obtenons pas une affectation réalisable des personnes, nous exécutons une étape de réparation dans lequel nous résolvons le problème maître décrit dans la section 7.3.2, mais nous ne considérons que les activités incluses dans le ordonnancement partiel actuel et on fixe leurs respectives dates de début.

Procédure de filtrage

Pour filtrer les noeuds qui seront évalués, si il ya deux noeuds x et y avec des ordonnancements partielles qui contient le même sous-ensemble d'activités, nous gardons le noeud lié à une makespan partielle inférieur. Cela implique, par exemple, que si le makespan partiel lié au noeud x est plus grand que le makespan partiel lié au noeud y, x est coupée de l'arbre. En outre, nous appliquons également les stratégies pour couper noeuds expliqués dans la section 7.4.1. Par la suite, les noeuds enfants restants sont inclus dans un ensemble B.

Bornes supérieures et inférieures

Dans le but de calculer la fonction de coût de l'évaluation d'un noeud généré, qui est égal à $V = (1-\alpha)\cdot LB+\alpha\cdot UB$, nous devons déterminer une borne inférieure (LB) et une borne supérieure (UB). De plus, α est un paramètre défini par des essais expérimentaux ($0 \leq \alpha \leq 1$). Ainsi, nous utilisons les mêmes méthodes présentées dans la section 7.4.1.

Phase de recuperation des erreurs potentielles

Maintenant, étant donné que nous conservons au plus w noeuds, à chaque niveau de l'arbre, l'étape de récupération consiste à considérer à la fois une réaffectation des personnes et une redéfinition des temps de début des activités qui pourraient conduire à détecter un autre ordonnancement partiel qui domine l'actuel. En outre, au lieu de considérer les différentes permutations d'ordonnancement partiel courant, nous résolvons un programme linéaire en nombres entiers (PLNE) qui nous permet de rechercher directement un nouveau ordonnancement partielle qui pourrait dominer l'actuel. Cette PLNE, est basée sur la formulation du problème maître introduit dans la section 7.3.2 mais nous ne considérons que les activités incluses dans le ordonnancement partiel actuel.

Par la suite, nous avons considéré différentes conditions de dominance. Les notions de telles conditions sont présentées comme suit: (i) Au moins une activité commence plus tôt, aucune autre activité retardé; (ii)au moins une activité commence plus tôt, d'autres activités pourraient être retardées.

Resultats Experimentaux

Les tests expérimentaux ont été effectués avec le solveur Gurobi Optimizer Version 4.6. Nous considérons les trois groupes d'instances étudiés dans les sections précédentes, néanmoins, ici nous montrons des résultats pour des instances avec de plus grande taille. Les groups d'instances considérées sont présentés comme suit:

- Groupe 1: Nous avons étudié 113 instances de ce groupe, en tenant compte: entre 20 et 51 activités, entre 2 et 8 compétences, et entre 5 et 14 personnes.
- Groupe 2: En ce qui concerne ce groupe des instances, nous incluons les résultats pour 94 instances qui considèrent entre: 32 et 92 activités, 9 et 15, les compétences et les 5 et 19 personnes.
- Groupe 3: Dans cette section, nous avons étudié 177 instances qui considère entre: 22 et 92 activités, 3 et 12 compétences, et les 4 et 15 personnes.

Par ailleurs, en ce qui concerne les paramètres nécessaires pour l'application de la méthode proposée, nous avons essayé de déterminer les valeurs qui pourraient donner le meilleur équilibre entre la qualité de la solution et l'effort du temps de calcul. Les valeurs suivantes ont été considérées pour les deux paramètres pris en compte lors de l'application de la RBS: (i)$\alpha = \{0, 1, 0, 2, \ldots, 0, 9\}$; (ii) $w = \{1, 2, \ldots, 8\}$. Ensuite, pour identifier une valeur unique pour w et α, l'algorithme proposé a été appliqué à un sous-ensemble d'instances présélectionnés. Compte tenu les valeurs de la fonction objective et les temps de calcul obtenues, les valeurs de w et α qui ont donné le meilleur équilibre entre la qualité de la solution et l'effort du temps de calcul sont $w = 3$ et $\alpha = 0, 1$.

En outre, dans le but d'analyser l'impact de la phase de recuperation des erreurs potentielles, nous avons comparé différentes approches de type Beam Search. Au départ, nous avons considéré une procédure Beam Search (BS), qui considère toutes les étapes décrites précédemment, sauf la phase de recuperation des erreurs potentielles. Ensuite, nous avons testé différentes procédures de RBS dans lequelles différents conditions de dominance ont été évalués. Des tests préliminaires montrent que la règle de dominance qui stipule que au moins une activité commence plus tôt et d'autres activités pourraient être retardées, donne de meilleurs résultats que les autres conditions de dominance. Ainsi, ici nous montrons les résultats de la procédure RBS qui considère la condition de dominance mentionné.

Le Tableau 7.7 présente une comparaison entre la BS et la RBS en termes de temps de calcul moyen, la déviation moyenne entre le UB obtenue et la meilleure borne inférieure connue (BLB), la déviation moyenne entre le UB obtenue et la borne supérieure obtenue par la recherche tabou développé par Bellenguez-Morineau et Néron [17](BUB) et le nombre total d'instances résolus de façon optimale.

Les résultats obtenus montrent qu'il n'y a pas une différence signficative dans les temps de calcul entre les deux procédures. En outre, le RBS a surperformé le BS en termes de la déviation moyenne entre la borne supérieure obtenu (UB) et la meilleure borne inférieure connue (BLB) et la borne supérieure obtenue par Bellenguez-Morineau et Néron (BUB). En outre, avec la BS, nous avons obtenu la solution optimale pour 124 d'instances,

		Groupes d'instances		
		Groupe 1	**Groupe 2**	**Groupe 3**
Temps de calcul moyen	BS	136,55	167,53	182,91
	RBS	161,91	167,59	184,27
Déviation moyenne entre UB et BLB	BS	3,99%	6,10%	5,47%
	RBS	3,71%	5,88%	5,25%
Déviation moyenne entre UB et BUB	BS	1,44%	2,83%	1,96%
	RBS	1,20%	2,63%	1,77%
Nombre de solutions optimales	BS	37	40	47
	RBS	45	41	50

Table 7.7: Comparaison des performances entre BS et RBS

alors que avec la RBS, nous avons obtenu la solution optimale pour 136 d'instances. Par conséquent, nous pouvons affirmer que la phase de recuperation des erreurs potentielles conduit à de meilleurs résultats pour le makespan.

7.5 Génération de coupes

Dans cette section, nous explorons une autre approche pour résoudre le MSPSP. En termes généraux, les méthodes de résolution étudiés dans les sections précédentes sont liées à la résolution des modèles mathématiques qui intègre la définition de les dates de début des activités et l'affectation des personnes dans une seule phase de solution. Par conséquent, dans cette section, nous explorons la possibilité d'utiliser une approche en deux phases pour résoudre le MSPSP. La première phase consiste à définir les dates de début des activités du projet. Par la suite, dans la deuxième phase, nous nous concentrons sur la recherche d'une affectation réalisable des personnes, qui permet l'exécution des activités en fonction de les dates de début définies dans la première phase. La résolution de cette procédure en deux phases se fait de façon itérative dans lequel de nouvelles inégalités (coupes) sont générés et inclus dans un PLNE pour assurer un ordonnancement optimal.

7.5.1 Travaux antérieurs relatifs à la génération de coupes

Tout programme linéaire en nombres entiers (PLNE) peut être résolu sans ramification. Pour ce faire, on peut résoudre itérativement le problème de séparation, qui consiste à trouver une coupe violé [78]. Une coupe est une inégalité valide qui n'est pas une partie de la formulation actuelle et qui n'est pas satisfait par la solution optimale du programme linéaire (PL). Par conséquent, une coupe qui n'est pas satisfait par la solution optimale du PL est appelée une coupe violés. Par la suite, l'ajout d'une inégalité violée dans le PL pourrait aider à améliorer la relaxation linéaire. Par la suite, il est concevable de résoudre un PLNE en résolvant sa relaxation linéaire et intégrer progressivement des nouvelles inégalités valides [35, 36]. Afin d'avoir une intuition des caractéristiques de l'inégalité, ci-dessous nous donnons une brève description des différents types d'inégalités.

7.5.2 Certains types d'inégalités

Compte tenu de l'approche proposée dans cette section, nous présentons d'abord deux types d'inégalités, lesquelles pourrait être approprié pour représenter certaines contraintes du MSPSP.

Inegalités de clique: Pour ce type d'inégalité, une phase de pré-traitement est nécessaire, pour déterminer un groupe de Q variables binaires, où pas plus d'une variable peut être non nulle [79]. Cette inégalité est décrite comme suit:

$$\sum_{i \in Q} x_i \leq 1 \tag{7.42}$$

Inégalités de couverture: Si une contrainte prend la forme d'une contrainte de type sac à dos (qui est une somme de variables binaires à coefficients positifs ou nuls, inférieur ou égal à un non négatif à droite), alors il ya une couverture minimale associée à la contrainte. Une couverture minimale est un sous-ensemble Q des variables de l'inégalité, où, si tous les variables dans Q sont égaux à un, la contrainte de type sac à dos serait violée, mais si au moins une variable est fixée à zéro, la contrainte serait satisfaite [3, 116]. Cette contrainte est donnée par:

$$\sum_{i \in Q} x_i \leq |Q| - 1 \tag{7.43}$$

7.5.3 Description globale de l'approche en deux phases

L'approche en deux étapes proposée dans cette section implique la résolution itérative de deux modèles de programmation linéaire en nombres entiers (ILP).

Dans la première phase, nous utilisons un PLNE, qui considère une variable de décision unique concernant les dates de début des activités. Les contraintes liées à la satisfaction des besoins en compétences de chaque activité sont estimés par des coupes sur cette variable. Par la suite, on peut estimer les dates de début de toutes les activités du projet et obtenir une borne inférieure pour le makespan.

Dans la seconde phase on applique un modèle d'affectation pour trouver une affectation réalisable des personnes compte tenu des dates de début définis dans la première phase. Ensuite, si la solution de ce dernier modèle conduit à une affectation réalisable des personnes, nous pouvons nous assurer que nous avons obtenu un ordonnancement optimal. Dans le cas contraire, il est implicite que nous devons redéfinir les dates de début des activités. Par conséquent, nous essayons d'éviter un ordonnancement irréalisable en générant et en ajoutant de nouvelles coupes dans le PLNE utilisé dans la première phase. Ainsi, ces nouvelles coupes sont ajoutés avec le but d' obtenir des dates de début qui conduit à une affectation réalisable des personnes et donc, à un ordonnancement optimal. La résolution des deux phases et l'inclusion de nouvelles coupes sont effectuées de manière

itérative jusqu'à le modèle d'affectation des personnes conduit à une solution réalisable (voir la figure 7.6).

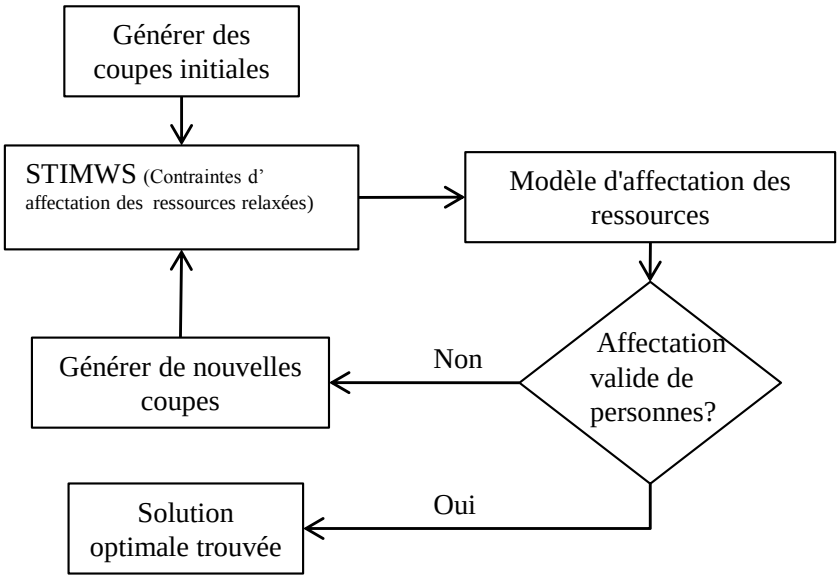

Figure 7.6: Approche en deux phases pour résoudre le MSPSP.

Première phase: PLNE indexé sur le temps

La première étape de l'approche présentée dans cette section consiste à résoudre une nouvelle modèle indexé sur le temps. Par conséquent, nous considérons que la variable de décision z_i^t qui prend le valeur de 1 si lactivité A_i commence à la date t, ou prend la valeur de 0 sinon. Cette PLNE (STIMWS) est une version simplifiée du modèle TIMWS introduit dans la section 7.2.3).

Au départ, notre principale préoccupation est d'estimer les dates de début des activités, donc les contraintes liées à la satisfaction des besoins en compétences de chaque activité sont estimés par des coupes sur z_i^t. Le PLNE résultante (STIMWS) est défini comme suit:

$$Z[STIMWS] : Min\ C_{max} = t_N \tag{7.44}$$

$S.t.$

$$\sum_{t\in[0,T]} (z_i^t \cdot t) + p_i \leq \sum_{t\in[0,T]} (z_j^t \cdot t) \quad \forall i \in A, \forall j \in E_i^+ \tag{7.45}$$

$$\sum_{t\in[0,T]} z_i^t \leq 1 \quad \forall i \in A \tag{7.46}$$

$$\sum_{i\in A'} \sum_{s\in[t-p_i+1,t]} z_i^s \cdot b_i^{K'} \leq b(A'; K'; T') \quad \forall t \in T' \tag{7.47}$$

$$\sum_{i \in A'} \sum_{s \in [t-p_i+1,t]} z_i^s \leq l(A';T') \quad \forall t \in T' \tag{7.48}$$

$$z_i^t \in \{0,1\} \quad \forall i \in A, \forall t \in [0,T] \tag{7.49}$$

La fonction objectif (7.44) est minimiser le makespan du projet. L'ensemble des contraintes (7.45) représente la relation de précédence entre les activités. L'ensemble des contraintes (7.46) précise qu'une activité ne peut commencer quune fois au maximum. L'ensemble des contraintes (7.47) et (7.48) permet une approximation des besoins en compétences de chaque activité.

Plus précisément, les contraintes (coupes) (7.47) et (7.48) sont définis comme suit:

Definition 7.1 *(Coupe cumulative). Pour un sous-ensemble dactivités (A'), de compétences (K') et de périodes (T'): Pour un instant donné on ne peut pas affecter plus de $b(A';K';T')$ personnes simultanément disponibles. Par la suite, nous pouvons introduire l'inégalité (coupe) respective:*

$$\sum_{i \in A'} \sum_{s \in [t-p_i+1,t]} z_i^s \cdot b_i^{K'} \leq b(A';K';T') \quad \forall t \in T'$$

où: $b_i^{K'} = \sum_{k \in K'} b_i^k$

Pour générer les coupes cumulatives, nous considérons que A' contient toutes les activités du projet et que T' couvre l'horizon de planification T. Enfin, nous générons toutes les combinaisons possibles de compétences, et pour chaque sous-ensemble généré K', on calcule la valeur de $b(A', K', T')$. Pour définir la valeur de ce dernier paramètre, nous supposons que tous les personnes dans W sont disponibles à un instant donné t et ensuite, nous calculons le nombre maximum de personnes qui pourraient utiliser au moins une compétence dans K'.

En outre, il est important de noter que cette réduction cumulative a été considéré par [102] pour représenter les contraintes de ressources dans un modèle en temps indexé pour résoudre le RCPSP. Néanmoins, dans le contexte du MSPSP l'utilisation de ces inégalités ne suffisent pas pour assurer une affectation réalisable des personnes.

Definition 7.2 *(Coupe de cardinalité). Pour un sous-ensemble dactivités (A') et périodes (T'): Pas plus dune certaine quantité $l(A';T')$ dactivités ne peut être effectuées en parallèle à un instant donné. Par la suite, nous pouvons introduire l'inégalité (coupe) respective:*

$$\sum_{i \in A'} \sum_{s \in [t-p_i+1,t]} z_i^s \leq l(A';T') \quad \forall t \in T'$$

Maintenant, si nous fixons $l(A', T')$ égal à 1, on peut classer cette dernière coupe comme une inégalité de clique (7.42). Cette inégalité a été appliquée par [41] pour résoudre le RCPSP.

Pour générer des coupes de cardinalité on considère plusieurs ensembles A' qui correspondent aux paires dactivités en disjonction à cause des contraintes de ressource. De la même façon T' est définie par rapport à la fenêtre de temps du début dactivités dans chaque ensemble A'.

Deuxième phase: modèle d'affectation des personnes AM$'(\Omega)$

Dans la deuxième phase, nous proposons une procédure pour trouver une affectation réalisable des personnes en fonction des dates de début définis dans la première phase.

Ainsi, nous appliquons un modèle d'affectation (AM$'(\Omega)$) inspiré du problème maître proposé dans l'approche de génération de colonnes introduit dans la section 7.3.2.

Notez que Ω couvre l'énumération de toutes les combinaisons possibles de personnes qui pourraient être affectées à chaque activité du projet. Par conséquent, avant d'exécuter le modèle(AM$'(\Omega)$), nous générons toutes les combinaisons possibles de personnes qui pourraient être affectées à chaque activité, au moyen de la recherche dun flot maximum à coût minimum (voir la section 7.3.2).

En outre, les paramètres relatifs à chaque colonne ω sont construits selon les dates de début définis dans la première phase. Par la suite, les variables de décision de ce modèle sont définies par: (i)x_ω qui prend la valeur de 1 si la colonne ω est sélectionnée ou 0 dans le cas contraire; (ii) v_i est une variable d'écart, qui est ajouté pour assurer, si l'exécution d'une activité est réalisable. Le PLNE respectif se présente comme suit:

$$Z[AM'[\Omega)] : Min \sum_{i \in A} v_i \tag{7.50}$$

$$\sum_{\omega \in [0,\Omega]} (x_\omega \cdot \alpha_i^\omega) + v_i = 1 \quad \forall i \in A \tag{7.51}$$

$$\sum_{\omega \in [0,\Omega]} (x_\omega \cdot \gamma_{m,t}^\omega) \leq 1 \quad \forall m \in W, \forall t \in [0,T] \tag{7.52}$$

$$x_\omega \in \{0,1\} \quad \forall \omega \ \in [0,\Omega] \tag{7.53}$$

$$v_i \geq 0 \quad \forall i \in A \tag{7.54}$$

La fonction objective du AM$'(\Omega)$ est de minimiser le nombre de activités pour lesquelles il n'a pas été possible de trouver une affectation réalisable des personnes. Ainsi, si v_i est supérieur à zéro, c'est parce qu'il n'est pas possible de trouver une affectation réalisable des personnes pour l'activité A_i. Comme il a déjà été expliqué dans la section 7.3.2, l'ensemble des contraintes (7.51) indique que seulement une colonne peut être affecté à chaque activité A_i et l'ensemble des contraintes (7.52) précise que tout opérateur peut effectuer au plus une activité à un moment donné.

Si la solution du AM$'(\Omega)$ conduit à une affectation réalisable des personnes ($\sum_{i \in A} v_i^* = 0$), cela implique que l'ordonnancement actuel, il est non-préemptif, et donc optimale.

Dans le cas contraire, nous cherchons à générer une nouvelle coupe qui pourraient être ajouté dan le modèle indexé sur le temps (STIMWS). Par la suite, nous continuons avec la procédure itérative illustré dans la figure 7.6.

Par conséquent, nous définissons cette nouvelle coupe comme suit:

Definition 7.3 *(Coupe de Overlapping subsets). Pour obtenir cette nouvelle coupe, d'abord nous devons définir trois nouveaux sous-ensembles:* I, D *et* D^i. I *es un sous-ensemble d'activités pour lesquelles nous n'avons pas obtenu une affectation valide des personnes et* D *est un sous-ensemble de* I. *En outre, pour tout* $i \in D$, D^i *correspond au sous-ensemble d'activités qui sont traités simultanément avec l'activité* A_i *inclus dans D (*A_ißD*). Enfin,* t_i *est une date de départ possible pour chaque activité* A_i *qui appartient au sous-ensemble* D. *Par conséquent, nous générons la coupe suivante:*

$$\sum_{i \in D} \sum_{j \in D^i} \sum_{s \in [t_i - p_j + 1, t_i + p_i - 1]} z_j^s + \sum_{i \in D} z_i^{t_i} \leq \sum_{i \in D} \left|D^i\right| + |D| - 1 \tag{7.55}$$

Cette inégalité stipule que au moins une activité parmi lun des sous ensembles D^i, ne peut être exécutée en parallèle avec son activité A_i correspondante dans D. Remarquez, que cette dernière coupe peut être classée comme une inégalité de couverture.

Procédure pour identifier le sous-ensemble I

Après de avoir résolu AM′(Ω), on initialise le sous-ensemble I avec les activités avec $v_i^* > 0$ et on définit un nouveau paramètre F égale à la valeur optimale résultante($F = \sum_{i \in A} v_i^*$). Notez que l'ensemble de contraintes (7.51) ne nous permet de détecter directement l'ensemble des activités qui devraient être inclus dans I. Par conséquent, nous ajoutons la contrainte suivante en AM′(Ω) qui oblige une affectation réalisable des personnes pour les activités inclus actuellement dans I.

$$\sum_{i \in I} v_i = 0 \tag{7.56}$$

Par la suite, nous suivons une procédure itérative, où nous ajoutons cette dernière contrainte, pour résoudre AM′(Ω) Ensuite, nous mettons à jour le sous-ensemble I avec les activités avec $v_i^* > 0$. Cette procédure est répétée tant que la fonction objectif actuel du modèle d'affectation soit égale à F (voir la figure 7.7).

Procédure pour identifier le sous-ensemble D Dans notre approche nous donnons la priorité à réduire au minimum le nombre d'activités dans D. Ainsi, nous utilisons un modèle mathématique (SC) qui nous permet d'identifier les activités incluses dans D. Ce modèle prend en compte une variable de décision y_i $quiprendunevaleurgale1sil'activit\mathrm{A}_i$ est inclus dans le sous-ensemble D, ou prend une valeur égale à 0 dans le cas contraire.

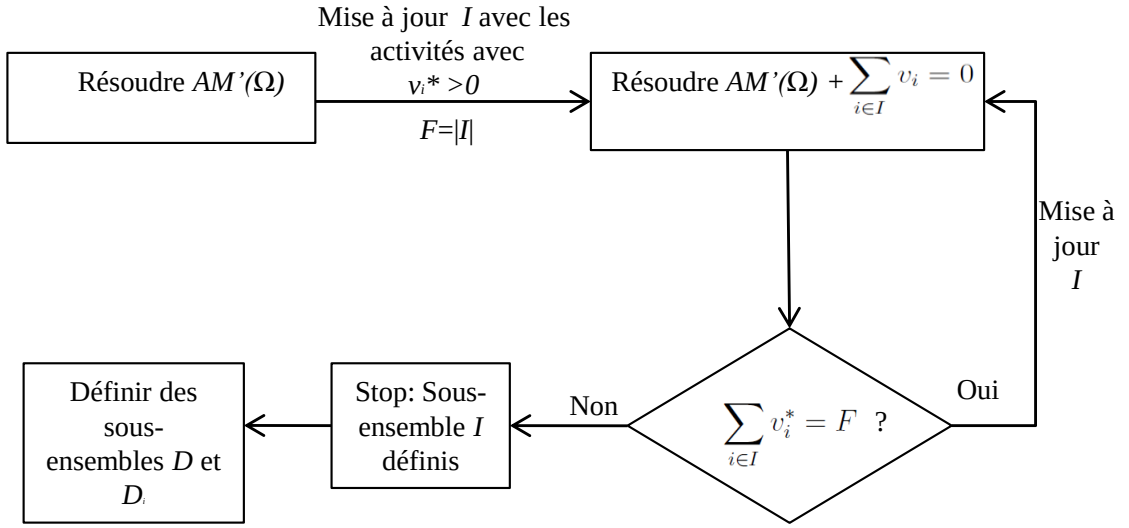

Figure 7.7: Procédure pour identifier le sous-ensemble I

$$Z[SC] : Min \sum_{i \in I} y_i \tag{7.57}$$

$$\sum_{i \in I} (y_i \cdot P_{i,j}) \geq 1 \quad \forall j \in I \tag{7.58}$$

$$y_i \geq 0 \quad \forall i \in I \tag{7.59}$$

Le but du modèle est de minimiser le nombre d'activités qui seront inclus dans D (7.57). L'ensemble de contraintes (7.58) stipule que chaque activité dans I est effectué en parallèle avec au moins une activité dans D.

Par ailleurs, en supposant que nous avons généré une coupe de overlapping subsets (cela implique que nous avons déjà identifié I et D), pour valider l'inégalité résultante, nous cherchons à trouver un ordonnancement réalisable pour les activités dans I lors de la fixation de les dates de début des activités dans D. En outre, on force l'exécution en parallèle entre chaque activité $A_i \in D$ et toutes les activités dans D^i. Par la suite, si nous ne sommes pas en mesure de trouver un ordonnancement réalisable pour les activités en I, on peut affirmer que la coupe de overlapping subsets est valide, donc il peut être ajouté au modèle indexé sur le temps introduit dans la première phase de notre approche. La procédure globale pour générer une coupe de overlapping subsets et de l'ajouter dans STIMWS est illustré dans la figure 7.8.

7.5.4 Procedure de Branch and Bound

Pour améliorer la performance de l'approche proposée nous avons implémenté un Branch and Bound (B&B) basée sur l'utilisation de la stratégie branchement: fenêtre de temps

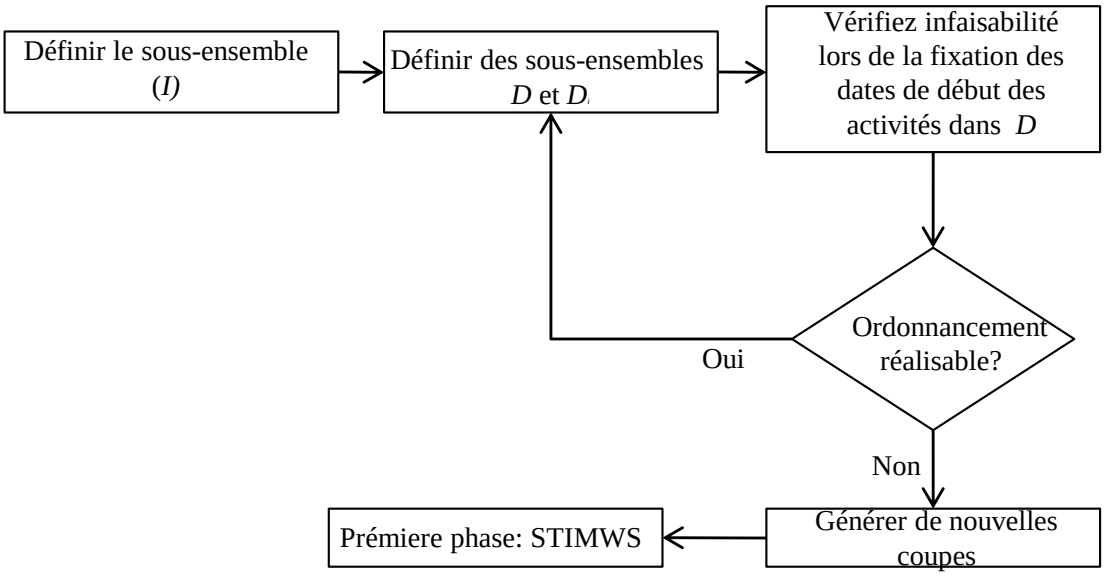

Figure 7.8: Procédure pour générer une coupe de overlapping subsets

dichotomique. Cette stratégie de branchement a déjà été expliqué dans la section 4.1. Ainsi, avec une telle approche, nous visons à accélérer la résolution du modèle indexé sur le temps et générer de nouvelles coupes qui pourraient conduire à une solution optimale. En outre, après avoir fait une propagation préliminaire sur le fenêtres de temps des actitivites du projet, nous branche sur le activité A_i que, conduit à une diminution du nombre de variables de décision z_i^t inclus dans le modèle indexé sur le temps (STIMWS). Par conséquent, après avoir généré un nouveau noeud, nous appliquons la procédure de résolution en deux phases.

7.6 Resultats Experimentaux

Les tests expérimentaux ont été effectués avec le solveur Gurobi Optimizer Version 4.6. Nous avons considéré un sous-ensemble de 271 instances de celles présentées dans la section 7.2.2. Les groups d'instances considérées sont rappelé comme suit:

- Groupe 1: Nous avons étudié 110 instances de ce groupe, en tenant compte: entre 20 et 51 activités, entre 2 et 8 compétences, et entre 5 et 14 personnes.
- Groupe 2: En ce qui concerne ce groupe des instances, nous incluons les résultats pour 71 instances qui considèrent entre: 32 et 62 activités, 9 et 15, les compétences et les 5 et 19 personnes.
- Groupe 3: Dans cette section, nous avons étudié 90 instances qui considère entre: 22 et 32 activités, 3 et 12 compétences, et les 4 et 15 personnes.

Nous avons également testé des instances avec de plus grandes tailles, mais nous n'avons pas obtenu des solutions optimales au-delà des dimensions mentionnées précédemment.

En outre, le tableau 7.8 montre les résultats obtenus avec l'approche de génération de coupes présentée dans cette section. Les résultats sont présentés en termes de: (i) nombre de solutions optimales obtenues dans un temps limite de 30 minutes, (ii) nombre de nouvelles solutions optimales (pour les instances où la solution optimale était jusqu'à présent inconnue), (iii) temps de calcul moyen.

	Groups d'instances		
	Groupe 1	**Groupe 2**	**Groupe 3**
Nombre de solutions optimales	89	61	89
Nombre de nouvelles solutions optimales	63	23	87
Temps de calcul moyen (s)	338,52	366,05	72,623

Table 7.8: Résumé des résultats obtenus

Par conséquent, les résultats présentés dans le tableau 7.8 montre que nous sommes en mesure d'atteindre l'optimalité en 94, 60 et 89 instances pour chaque groupe de problèmes. Ensuite, pour chaque groupe de problèmes, on a également obtenu des solutions optimales en 63, 23 et 87 sur les 78, 31 et 88 instances testés pour lequelles la solution optimale était jusqu'à présent inconnue. Enfin, on peut distinguer que avec l'approche de génération de coupes, nous avons obtenu la solution optimale pour 243 sur les 271 instances testés, et que nous avons pu obtenir 173 nouvelles solutions optimales.

7.7 Conclusions et Perspectives

Les travaux réalisés au course de cette thése portant sur un probléme de gestion de projet multi-compétence (noté MSPSP), qui s'avére $\mathcal{NP}$-difficile. Nous travaux se sont donc tout d'abord portés sur une définition précise du probléme, ainsi que d'une modéle utilisable pour celui-ci. Par la suite, nous avons mis au point des bornes inférieures afin d'évaluer la durée minimum d'une instance de MSPSP. Par la suite, nous avons proposé différentes méthodes approches. Donc, ci-dessous, nous présentons les principales conclusions liées à chacune des approches proposées.

Au départ, nous pouvons stipuler que nous avons introduit différents modèles de programmation linéaire en nombres entiers (PLNE). Ensuite, en ce qui concerne l'approche de génération de colonnes, nous avons considéré différentes alternatives pour résoudre le problème maitre relaxé et nous avons amélioré la relaxation linéaire obtenue avec les modèles de programmation linéaire en nombre entiers. Par la suit, en ce qui concerne l'approche de B&P nous avons comparé deux stratégies de branchement et nous avons obtenu une solution optimale pour 72 instances où une telle solution était jusqu'à présent inconnue. Ensuite, lors de l'utilisation de la RBS, il était possible de traiter des instances plus importantes et nous pouvons affirmer que la méthode de solution proposé ont une performance compétitive. Enfin, en ce qui concerne la génération de coupes nous avons

développé une procédure itérative qui ajoute différents types de coupes à un modèle indexé sur le temps et nous avons obtenu une solution optimale pour 173 instances où une telle solution était jusqu'à présent inconnue.

Enfin, parmi les nombreuses perspectives de recherche, nous envisageons de proposer des méthodes qui prendraient en compte un cout associé aux personnes selon les compétences qu'elles maîtrisent. Par la suite, nous souhaiterons également mettre en place des nouvelles approches qui réagissent aux événements non prévus. On peut alors tout-àfait envisager de considérer d'autres critères d'optimisation peuvent être envisagés. On peut également envisager l'utilisation de Resource leveling pour examiner l'utilisation déséquilibrée des ressources et estimer le nombre et les types de ressources nécessaires pour exécuter un projet.

l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

ECOLE DES MINES DE NANTES

Thèse de Doctorat

Carlos Montoya

New methods for the Multi-Skill Project Scheduling Problem
Nouvelles méthodes pour le Problème de Gestion de Projet Multi-Compétence

Résumé

Dans cette Thèse, nous avons introduit plusieurs procédures pour résoudre le problème d'ordonnancement du projet multi-compétences (MSPSP). L'objectif est de trouver un ordonnancement qui minimise le temps de terminaison (makespan) d'un projet, composé d'un ensemble d'activités. Les relations de précédences et les contraintes de ressource seront considérées. Dans ce problème, les ressources sont des membres du personnel qui maîtrisent plusieurs compétences. Ainsi, un certain nombre de travailleurs doit être affecté pour utiliser chaque compétence requise par une activité. Par ailleurs, nous accorderons une importance particulière aux méthodes exactes pour résoudre le MSPSP, puisqu'il y a encore un certain nombre d'instances pour lesquelles l'optimalité doit encore être prouvée. Néanmoins, pour traiter des instances plus importantes, nous implémentons une approche heuristique.

Mots clés

Optimisation, Ordonnancement des Projets, Ressources Multi-compétences, Méthodes Exactes, Méthodes Arborescentes.

Abstract

In this Phd Thesis we introduce several procedures to solve the Multi-Skill Project Scheduling Problem (MSPSP). The aim is to find a schedule that minimizes the completion time (makespan) of a project, composed of a set of activities. Precedence relations and resource constraints are considered. In this problem, resources are staff members that master several skills. Thus, a given number of workers must be assigned to perform each skill required by an activity. Furthermore, we give a particular importance to exact methods for solving the Multi-Skill Project Scheduling Problem (MSPSP), since there are still several instances for which optimality is still to be proven. Nevertheless, with the purpose of solving big sized instances we also developed and implemented a heuristic approach.

Key Words

Optimization, Project Scheduling, Multi-skilled Resources, Exact Methods, Search Tree Methods.