HAL open science

Operational research approach for optimising the operations of a nuclear research laboratory

Oliver Polo Mejia

To cite this version:

Oliver Polo Mejia. Operational research approach for optimising the operations of a nuclear research laboratory. Automatic Control Engineering. Institut national des sciences appliquées de Toulouse, 2019. English. NNT : . tel-02309284

HAL Id: tel-02309284

https://hal.science/tel-02309284v1

Submitted on 9 Oct 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ FÉDÉRALE TOULOUSE MIDI-PYRÉNÉES

Délivré par :
l'Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue le *19/09/2019* **par :**

Oliver Javier POLO MEJIA

Operational research approach for optimising the operations of a nuclear research laboratory

---

JURY

Stéphane DAUZÈRE-PÉRÈS	Professeur IMT Mines Saint-Etienne, Gardanne	Président du Jury
Maria DI MASCOLO	Directrice de Recherche, CNRS, Grenoble	Rapporteure
Ameur SOUKHAL	Professeur des Universités, Tours	Rapporteur
Olivier DUGNE	Ingénieur CEA, Marcoule	Examinateur
Philippe LABORIE	Ingénieur IBM Research, Paris	Examinateur
Christian ARTIGUES	Directeur de Recherche, CNRS, Toulouse	Directeur de Thèse
Pierre LOPEZ	Directeur de Recherche, CNRS, Toulouse	Directeur de Thèse
Marie-Christine ANSELMET	Ingénieur CEA, Cadarache	Invitée
Virginie BASINI	Chef adjointe du CEA/SETC, Cadarache	Invitée

École doctorale et spécialité :
EDSYS : Génie Industriel 4200046
Unité de Recherche :
LAAS-CNRS Toulouse et CEA Cadarache
Directeur(s) de Thèse :
Christian ARTIGUES et *Pierre LOPEZ*
Rapporteurs :
Maria DI MASCOLO et *Ameur SOUKHAL*

Acknowledgements

I would like to express my sincere gratitude to Dr. Christian Artigues and Dr. Pierre Lopez, my research supervisors, for their patient guidance, enthusiastic encouragement and useful criticisms of this research work. Thank you so much for comforting me during the moments of doubts. I must thank Dr. Marie-Christine Anselmet for welcoming me and guiding me at the LECA-STAR during the first two years of this thesis. I would also like to thank Dr. Virginie Basini, for taking over the research project and advising me during the last year of my PhD. Without her (and her amazing planning skills), this dissertation would not have been ready in such a short time.

Thanks to all the members of the LEPC and especially to Ing. Olivier Descombin and Ing. Quentin Elie. I can not fail to mention the engineering apprentices from number 1 up to number 6: Thibaud, Benoit, Jerôme, Yannis, Anysia and Jordane. Thank you all for making my stay at the LECA-STAR enjoyable (sorry for the numbers, guys).

A special thank you to the planning team at LECA-STAR who were always there to answer my questions. I must express my deepest gratitude to Mrs Céline Ayme, who gave me her time to explain to me the operation of the laboratory. I could not ask for a better office partner (I am really going to miss the chocolates during the winter).

My deep gratitude to all the members of the ROC team at LAAS-CNRS for welcoming me during my visits to the laboratory.

Thanks to my family who, despite the distance, was always there to support me and never let me lose sight of my goals.

Finally, I would like to thank all the people that indirectly were part of my formation. To all my teachers at the Universidad del Norte and at the ENIM, thank you. Also, thanks to Luis and Rosnen, my French teachers. Thanks to the Colombian government and the French embassy for the scholarship that allowed me to come to France.

Contents

Introduction **1**

1 Industrial Context **5**
1.1 The nuclear research facility under study 6
1.2 Importance of the operational schedule 8
1.3 Analysis of the current scheduling process 9
1.3.1 Scheduling characteristics 9
1.3.2 Management of scheduling requests and schedule generation 10
1.4 Nuclear and R&D activities scheduling 11
1.5 Proposed improvement approach 12
1.5.1 Scheduling engine 13
1.5.2 Management information system 14
1.6 Concluding remarks 15

I State of the Art **17**

2 Combinatorial Optimisation Problems **19**
2.1 Some classical combinatorial optimisation models 20
2.1.1 Knapsack problems 20
2.1.2 Network and graph problems 22
2.1.3 Location and routing problems 23
2.1.4 Scheduling problems 25
2.2 Classical solution methods 27
2.2.1 Mixed-Integer Linear Programming 27
2.2.2 Constraint Programming 30
2.2.3 (Meta)Heuristic methods 31

3 Project Scheduling Problems **37**
3.1 Resource-Constrained Project Scheduling Problem (RCPSP) 38
3.1.1 Some variants and extensions 39
3.1.2 Modelling and solution approaches 41
3.2 Multi-Skill Project Scheduling Problem (MSPSP) 44

II Proposed Methods **49**

4 MSPSP with penalty for preemption **51**
4.1 Problem description 52

4.2 MILP formulations . . . 53
4.2.1 Model MSWP1 . . . 53
4.2.2 Model MSWP2 . . . 56
4.2.3 Model MSWP3 . . . 57
4.2.4 Model MSWP4 . . . 58
4.3 Computational experiments . . . 60
4.4 Limited theoretical analysis . . . 62
4.5 Concluding remarks . . . 63

5 MSPSP with Partial Preemption 65
5.1 Problem description . . . 65
5.2 MILP formulations . . . 68
5.2.1 Model MSPP1 . . . 68
5.2.2 Model MSPP2 . . . 71
5.2.3 Theoretical comparison . . . 73
5.3 CP formulation . . . 74
5.4 Computational Experiments . . . 78
5.4.1 Testing MILP formulations . . . 78
5.4.2 Testing CP formulation . . . 81
5.5 Concluding remarks . . . 82

6 Heuristic Methods for the MSPSP-PP 85
6.1 Flow problem for technicians allocation . . . 86
6.2 Greedy Algorithm: Serial Generation Scheme . . . 88
6.3 Binary-Tree-based Local Search Algorithm . . . 91
6.4 Greedy Randomised Adaptive Search Procedure . . . 93
6.5 Large neighbourhood search . . . 97
6.6 Computational experiments . . . 100
6.7 Concluding remarks . . . 107

III Industrial Experience 111

7 Industrial application 113
7.1 Scheduling engine – GUI . . . 113
7.1.1 Data entry . . . 114
7.1.2 Schedule generation . . . 116
7.1.3 Schedule display . . . 117
7.1.4 GUI tests . . . 117
7.2 Implementation of the management information system . . . 119
7.3 Concluding remarks . . . 120

Conclusions and perspectives **123**

Bibliography **127**

List of Tables

4.1 Parameters for the MSPSP with penalty for preemption 54
4.2 Distribution of preemption types per set of instances 61
4.3 Feasible solutions after 15 minutes 61
4.4 Wilcoxon signed-ranks test 62

5.1 Distribution of preemption types for instances of the MSPSP-PP 79
5.2 Results for all configurations of Model MSPP1 without warm start 79
5.3 Results for all configurations of Model MSPP1 with warm start 80
5.4 Results for Model MSPP2 without warm start 80
5.5 Results for Model MSPP2 using warm start 80
5.6 Results for the CP formulation 81
5.7 Results for CP model with warm start 82

6.1 Distribution of preemption types for instances of the MSPSP-PP 100
6.2 Average gap for the greedy algorithm per priority rule 101
6.3 Wilcoxon signed-rank test for gap equality by priority list 102
6.4 Wilcoxon test for equality between greedy algorithm and CP 102
6.5 Results for local search algorithm with self-adaptive P_{max} 103
6.6 Results for GRASP algorithm 105
6.7 Statistical test for improvement due to the intensification component . . . 105
6.8 Results for single sweep LNS using MILP and CP 106
6.9 Results for multi-sweep LNS using MILP 107
6.10 Results for multi-start LNS 107
6.11 Results for LNS after GRASP 108

7.1 Characteristics of instances for GUI test 118

List of Figures

1.1 LECA-STAR facility 7
1.2 Hot cells and teleoperation zone 8
1.3 Integrated scheduling support system 12
1.4 Project guideline 13

2.1 A non-exhaustive metaheuristics classification. 32

3.1 Characteristics of the classical RCPSP 39
3.2 Characteristics of the MSPSP 45

5.1 Characteristics of the Multi-Skill Project Scheduling Problem with Partial Preemption (MSPSP-PP). 66
5.2 Example of an MSPSP-PP instance. 67

6.1 Flow graph for the MSPSP 86
6.2 Flow graph for the MSPSP-PP 87
6.3 Example of a binary tree 92
6.4 Time window right shift 99
6.5 Gap and time evolution in function of P_{max} for all instances 103
6.6 Gap and time evolution in function of P_{max} by preemption type 104

7.1 Scheduling engine GUI - main window 114
7.2 Technicians data management 115
7.3 Resources data management 115
7.4 Activities requests 116
7.5 Indicating forbidden/allowed scheduling periods 116
7.6 Gantt chart for a generated schedule 118
7.7 Form used for testing the management information system 120

Introduction

To ensure the development of French nuclear industry, the Alternative Energies and Atomic Energy Commission or CEA (in French: Commissariat à l'Énergie Atomique et aux Énergies alternatives), must ensure that its key research facilities are working in the best way. The LECA-STAR is a nuclear research laboratory playing an essential role in the CEA's projects, since it ensures the execution of most of the post-irradiation experiences over nuclear fuel. Given the characteristics of the activities carried out at the LECA-STAR, an improvement on the scheduling process could be beneficial for ensuring the best performance of the facility, and thus for facing the new economic challenges.

As the first step on this improvement process, this PhD project aims to identify how combinatorial optimisation techniques could be applied for optimising the weekly scheduling process at the LECA-STAR. Combinatorial optimisation techniques have been already used to schedule nuclear-related activities such as nuclear power plant construction, nuclear waste placement or nuclear power plants outages and maintenance. Activities carried out at the LECA-STAR are very close to those of R&D projects, where the order and types of activities to be carried out during the project can vary enormously due to the results obtained in the early stages. However, in the same way that for nuclear-related scheduling activities, most of the literature of R&D project scheduling deals with broad scheduling horizons. Scheduling over a short scheduling horizon, as in this PhD project, reduces the uncertainty of the activities to be scheduled, thus allowing us to use standard scheduling methods.

In Chapter 1, we describe the industrial context of this thesis. We show the importance that a good scheduling process has for hot laboratories such as the LECA-STAR. After describing the current scheduling process and the main characteristics of the laboratory operations, we present the approach we propose to improve the weekly scheduling process. This approach requires the implementation of a management information system, which will support the scheduling engine that exploits the models and algorithms developed in this dissertation. We then present the characteristics that each of these elements must have.

After presenting the characteristics of the facility, we present in Part I of this document a literature review related to the problem at hand. We present in Chapter 2 an overview of Combinatorial Optimisation Problems and their solution methods. We do later a more detailed review of the Resource-Constrained Project Scheduling Problem (RCPSP) in Chapter 3. This literature review leads to conclude that the scheduling problem at the LECA-STAR can be modelled as a variant of the well-known Multi-Skill Project Scheduling Problem (MSPSP). This problem assumes that the preemption of activities is not allowed. However, the possibility of preempting the activities is crucial for modelling the LECA-STAR scheduling problem. We must then do some modifications of the classical version and allow the preemption of the activities.

In Part II, we describe the models and algorithms proposed to schedule the activities at the LECA-STAR. As a first approach, we propose in Chapter 4, an MSPSP with penalty for preemption. In this variant, preemption is allowed for some of the activities, but a penalty is applied every time an activity is preempted. The idea behind this penalty is to decrease the number of times an activity is stopped, thus limiting the impact on productivity linked to the fact of resuming the activity. We present four discrete-time based Mixed Integer/Linear Programming (MILP) models for the problem. The first three models are original and inspired by the time-indexed formulations for the (preemptive) RCPSP; the fourth one is an adaptation of a model proposed in the literature for the preemptive MSPSP. Computational experiments on the performance of the proposed MILP models are also presented in this chapter. Finally, we present a limited theoretical analysis of the strength of the proposed models.

For some critical activities, safety constraints force us to ensure that a subset of resources remain allocated to the activity when it is preempted (what we define as partial preemption). The MSPSP with penalty for preemption does not fulfil these safety constraints. Traditional preemptive scheduling models cannot represent this behaviour since they assume that all resources are released during the preemption periods. The only way to model activities having these safety constraints was to declare them as "non-preemptive". However, this decision can increase the project makespan, especially in our case study, where the activities may have restrictive time-windows and the availability/capacity of the resources vary over time. Aiming to overcome this inconvenience, and to put the first stone for the application of the concept partial preemption that is lacking in the scientific literature, we propose in Chapter 5 a new variant of the MSPSP that better represents the behaviour of our laboratory: the MSPSP with partial preemption (MSPSP-PP). We present various MILP (together with their theoretical comparison) and Constraint programming formulations for the proposed model. Experimental tests are also carried out to analyse the performance of each of the different models.

The industrial applications of this PhD project required that we could find good scheduling solutions in short time. The MILP and CP models could take too long to find good solutions for industrial-size instances. That is why we present various heuristic methods for the MSPSP-PP in Chapter 6. First, we present a greedy algorithm that uses priority rules for generating the schedule and a flow problem for allocating the technicians. Then, a tree-based local search algorithm, partially inspired by the Limited Discrepancy search, is described. A greedy randomised adaptive search procedure, combining the greedy and local search algorithms, is also presented. Finally, we present a large neighbourhood search algorithm, a hybrid procedure combining exact and heuristic methods.

Part III is dedicated to the industrial application. Thus, in Chapter 7, we describe a standalone graphical user interface that allows testing the accuracy of the schedules generated by the methods proposed in Chapters 5 and 6. In this final chapter, we also present the main issues faced during the deployment phase of the management

information system, along with some advices that should facilitate the implementation of the results of this thesis in the future.

Chapter 1

Industrial Context

Contents

1.1 The nuclear research facility under study 6

1.2 Importance of the operational schedule 8

1.3 Analysis of the current scheduling process 9

1.3.1 Scheduling characteristics 9

1.3.2 Management of scheduling requests and schedule generation 10

1.4 Nuclear and R&D activities scheduling 11

1.5 Proposed improvement approach 12

1.5.1 Scheduling engine 13

1.5.2 Management information system 14

1.6 Concluding remarks 15

The Alternative Energies and Atomic Energy Commission or CEA (in French: Commissariat à l'Énergie Atomique et aux Énergies alternatives), is a French public government-funded research organisation created in 1945 to implement a major political project: "To develop all applications resulting from atomic sciences". Today, nuclear power remains a central topic of CEA research projects. However, to accomplish its mission, the organisation has had to broaden the scope of its research in physics, chemistry and biology, and develop new knowledge in microelectronics, materials and new energy technologies. Since its creation, the CEA has been a significant player in research, development and innovation, serving major strategic and industrial issues in France. As part of this mission, the CEA develops new scientific knowledge and transfers technological innovations to the industrial world; intervening today in different areas such as defence and security, low carbon energies (nuclear and renewable), information technologies, and basic research in material sciences and life sciences.

The department of nuclear fuel studies (DEC in short for French), within the CEA's division of nuclear energy, has as mission to acquire, integrate and capitalise the knowledge relating to the design, manufacture, characterisation and study of the behaviour (in all existing operating modes: normal, incidental and accidental mode) of nuclear fuel, as well as those related to the downstream nuclear fuel life cycle. These activities are carried out by combining numerical simulation and experimentation, particularly in experimental reactors and on large instruments, which implies that the department also

pilots the experimental fuel irradiation programs. To carry out its research programs, the DEC operates various facilities required for executing experiments pre- and post-irradiation. The LECA-STAR is the research facility in charge of carrying out all the post-irradiation experiences. Because of the strategic importance of this facility for the development of nuclear fuels, one must ensure its optimal operation. That is why the LECA-STAR has been the object of study of this research project.

In the following of this chapter, we present first in Section 1.1 the characteristics of the nuclear research facility under study. The operational schedule importance in such a facility is discussed in Section 1.2. The current scheduling process, identified as the central axis for improvement, is described in Section 1.3. In Section 1.4, we present some applications of Operations Research scheduling techniques on nuclear and research fields. Finally, in Section 1.5, we present the proposed improvement approach for the scheduling process at LECA-STAR.

1.1 The nuclear research facility under study

The LECA-STAR is a nuclear research facility located on the CEA Cadarache site, in operation since 1964 for its oldest part. It is in charge of the characterisation of irradiated fuel from different types of nuclear plants and reconditioning of spent fuels before storing. This laboratory plays an essential role in the support and development of the nuclear French industry; implementing state-of-the-art characterisation techniques in a complex, very restrictive and regulated environment in terms of safety and security. The LECA-STAR is constituted by two joined buildings: the active fuel testing laboratory (LECA) and the treatment, sanitation and reconditioning station (STAR).

The main activities of the LECA are post-irradiation destructive and non-destructive inspections: non-destructive measurements, optical microscopy and macroscopy, scanning electron microscopy, electron microprobe, secondary ion mass spectrometry, image analysis, quantitative gamma spectrometry, X-ray diffraction structural studies, and heat treatments. The STAR, on the other hand, was designed to treat and recondition spent fuel, and to perform destructive or non-destructive examinations on PWR (Pressurised Water Reactor) and SFR (Sodium Fast Reactor) spent fuel. Since 2010, STAR has been operating the Verdon laboratory, designed to carry out studies on accidental behaviour during severe accident conditions.

The LECA-STAR is classed as a “hot laboratory” (a laboratory designed for the use of radioactive substances). Hot laboratories are very special environments, with many safety and security constraints. The nuclear environment and the large number of restrictions and regulations make the scheduling process difficult, since tasks, which are considered easy in other industries, will become complex requiring much more time (preparation, execution) and resources. Indeed irradiated nuclear fuel must continuously be confined to dedicated areas (hot cells) to protect workers from the radiation. The operations carried out on the irradiated fuel are then made using teleoperation (carried

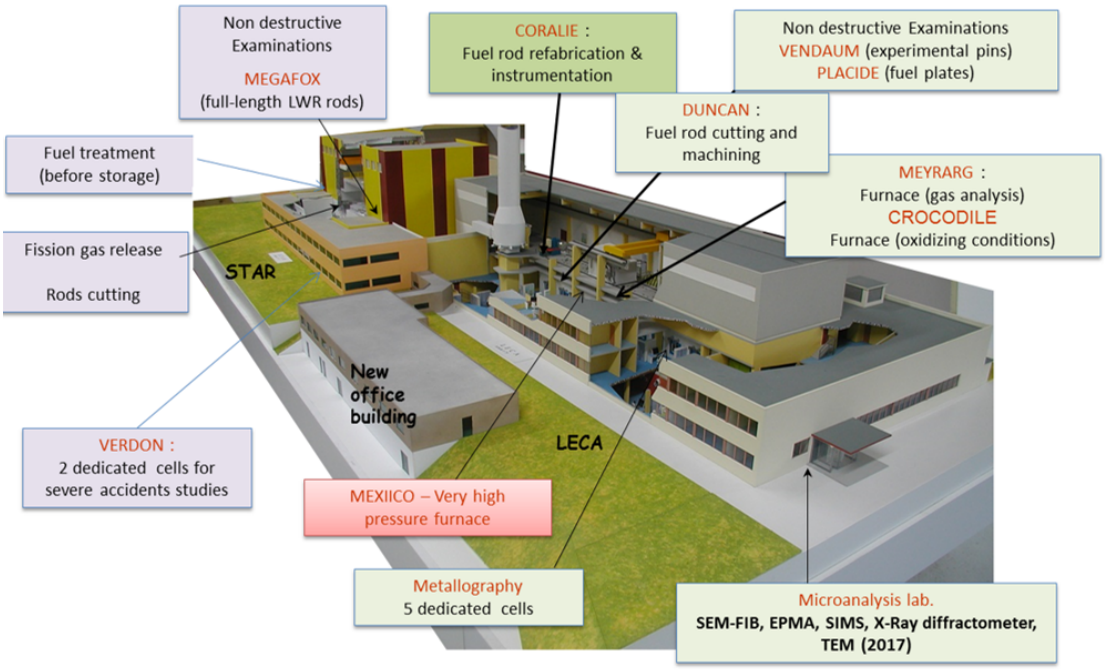

Figure 1.1: LECA-STAR facility

out from outside the hot cell via telemanipulator arms) by trained staff. Teleoperation makes any operation more complicated and time-consuming than whether it was done in a traditional workshop.

To ensure the execution of the experiments, and the normal running of the facility, an important number of maintenance (preventive and curative) activities must also be carried out. Again, the nuclear environment adds a complexity layer to these activities, requiring important preparation works. Maintenance over the manipulator arms, for example, require previous cleaning activities (decontamination) for reducing the radioactivity level within the hot cell. Receipt and shipment of nuclear fuel casks and disposal of effluents also need to be done to maintain operative the LECA-STAR. More than 100 activities are carried every week at the facility, and it welcomes around 110 employees every day among maintenance personnel, experimenters, R&D engineers, hot cells technicians and support staff. The number of activities, and the possible interaction between them, make the scheduling process in the facility have a significant impact on the performance of the whole laboratory and consequently on the progress of the research projects.

Because scheduling is also an important matter for nuclear safety, the LECA-STAR has decided to optimise its planning process. This PhD thesis is the first step in this improvement process, and it has as main focus the optimisation of the weekly schedule of activities within the facility by using Operations Research and, more precisely,

Figure 1.2: Hot cells and teleoperation zone

combinatorial optimisation techniques; see Chapter 2.

1.2 Importance of the operational schedule

Having a good schedule is crucial since it allows the decision makers in the facility to have guidance and a pathway to better focus the effort on critical activities, and to make better use of the limited resources. Additionally to these apparent functions, the schedule may have other indirect impacts such as:

Employees motivation: A schedule that is not precise enough, or which requires a large number of modifications after having been shared in its applicable version, can lead to loss of motivation from employees. Indeed it may exist a direct relationship between the accuracy of the planning (percentage of planning unchanged) and the commitment of employees. Employees will be very reluctant to comply with a schedule that is far from reality (causing even more delays); on the other hand, they may feel strongly motivated to respect the deadlines of a realistic schedule, thus increasing productivity.

Improvement of the working environment: A good schedule helps to maintain the harmony of employees in the workplace, removing sources of stress. By implementing a good scheduling strategy, confusion due to schedule changes or errors can be avoided.

Knowledge of real-time usage status of resources: Changing schedules can lead operators to make changes in the order in which the tasks are executed, without communicating them. This results in the impossibility of knowing in real time the actual, or at least an approximate, state of the load rate of the employees and the availability of the resources and the staff. Adequate scheduling should reduce the need to make changes on scheduled activities, allowing the planner to have a more realistic view of the current state of the facility and thus better manage the scheduling for future activities.

Improvement of the safety: Finally, but not less important, an adequate schedule plays an essential role in the safety of hot laboratories. An exhaustive schedule allows ensuring the respect of nuclear regulations and operational constraints, since it allows us to understand the possible interaction that can happen between different activities. In the nuclear environment, scheduling error could cause incidents (or accidents in the worst case) or endanger the staff. It is then important to be systematically sure of taking into account all constraints during the scheduling process.

1.3 Analysis of the current scheduling process

In this section, we present some of the operational characteristics of the current scheduling process at the nuclear research facility. We first describe the main characteristics of the activities and the resources that are taken into account during the schedule generation. Then we discuss how scheduling requests are treated and how the final schedule is generated.

1.3.1 Scheduling characteristics

Every week more than 100 activities are carried out at LECA-STAR, including maintenance (preventive or curative), experimental activities, nuclear transport and regulatory controls. The laboratory runs its operations continuously from Monday morning to Friday night (108 hours). However, not all activities can be scheduled at any moment due to the absence (known in advance) of some resources and staff during specific periods. Due to this "calendarisation" (see Chapter 3), we can not guarantee the continuous execution of activities with a duration larger than the work shifts. Additionally, sometimes we must preempt (stop an activity in process to continue it later) non-critical activities (such as certain experimental activities) to give priority to more critical activities (such as nuclear transports that have stringent constraints for scheduling). Allowing the preemption is then necessary for modelling our problem. A preempted activity can be resumed later by a set of resources different from the one used to start it. On the other hand, there is a subset of particular activities (set $\overline{NP}$ of non-preemptive activities) that must be executed without interruption due to safety and operational constraints.

Activities carried out at LECA-STAR require the allocation of technicians (staffs)

with particular skills and authorisations to be executed. In other words, not all technicians can execute all activities. Thus, activities are defined by their need for resources (equipment, machines, building) and their need for skills. Each technician has a specific set of skills it masters (we assume this mastering is done at the same level for each skill), and its periods of presences/absences (calendarisation) are defined in advance (and are not subject to change in this study). Technicians can be allocated to only one activity at a time, but they can execute several skills per activity at the same time. The same technician can, for example, be responsible for recording the movement of nuclear material in the database (not everyone can do it), and at the same time, he performs a cut of the sample. For operational reasons, we must, however, guarantee the allocation of a minimal number of technicians for an activity. For instance, activities in contaminated or isolated zones require at least two members of staff for surveillance.

The presence of time windows is also important in the nuclear facility. Nuclear regulation, for example, requires to carry out a series of periodic tests to ensure the proper functioning of the machines. These tests must be scheduled and executed before a deadline. Additionally, some of the activities are carried out in partnership with other laboratories, and they may be subject to the reception of a sample on a fixed contractual date, and the activity cannot start before such date. In this case, we say that the activity is subject to a release date. Sometimes an experimental or maintenance activity may require a set of setup activities. In this case, a precedence relationship exists between these activities (i.e. activity i cannot start before activity l is completed).

1.3.2 Management of scheduling requests and schedule generation

Today, the collection of information concerning the activity scheduling requests is made based on an intensive exchange of e-mails. This approach is cumbersome and complex to manage, given the large number of activities (and its respective constraints) to be scheduled. It is clear that the use of e-mails is not the most efficient way of collecting the information, and that it poses a significant risk of information loss, as the information is difficult to extract from e-mails that the planners have to handle. In addition, the lack of standardisation of e-mails further complicates the process of collecting information. Requesters continue to prefer the use of open and informal e-mails, which most of the time could not provide the level of information required for scheduling, including the resources involved and prerequisites. Having clear and timely information is, therefore, the first step in the scheduling process; that is why an improvement in the collecting information approach must be done.

Once collected and analysed all the requests, the planning team proceeds to schedule all the activities and to allocate the respective technicians as needed. At the start time of the thesis, this schedule is generated “by hand”, i.e. no automatic planning generation tool is used. The main objective of the planning team is to find a schedule that ensures that all the activities are finished as soon as possible. Manually generating the schedule cannot ensure that the right distribution of activities and resources has

been chosen. The manual process becomes even more complicated when the number of resources or activities to manage increases, along with the possibility to omit some of the numerous constraints unintentionally, which as stated before could lead to safety issues. Additionally, this is a time-consuming process. The use of combinatorial optimisation techniques can help the planning team to choose the best schedule (regarding a given objective) while ensuring that all constraints are systematically taken into account.

The scheduling of the activities for the following week must be constructed and validated during a meeting of the heads of research, heads of maintenance and engineers responsible for activities that take place at the end of the week. Although the planners prepare an interim scheduling before this meeting, it is common that changes must be made during the meeting due to new information or the status of the administrative progress of the documentation necessary to carry out an activity. It is then crucial to be able to have a new feasible schedule within a few minutes. Heuristic methods (Chapter 6) are a good choice to exploit the rigorousness of combinatorial optimisation models while obtaining good quality (not necessarily optimal) schedules in a short time. Once generated and validated, an offline version of the schedule is then transmitted to all employees.

1.4 Nuclear and R&D activities scheduling

A literature review allows identifying some applications of scheduling models within a nuclear environment, all of them for broad scheduling horizons. Chen *et al.* [44], for example, proposed a heuristic method to solve the nuclear power plant construction scheduling problem, that integrates building construction scheduling and reactor installation scheduling. The PhD dissertation by Petersen [137] presents various methods aiming to schedule the removal of spent nuclear fuel from reactor sites in the USA. His objective was to reduce the amount of time the shutdown reactors keep the spent fuel on site, and thus reduce the total system costs for the federal government. Johnson *et al.* [88] developed a mixed integer program for scheduling the nuclear waste placement in the Mountain repository in Nevada, USA, as a case study. Their model determines where to place each waste package of a specific type in a given period to minimise heat load concentration within a repository. In France, *Electricité de France (EDF)*, the largest European producer of electricity, has used combinatorial optimisation techniques to schedule outages and maintenance of nuclear power plants [61, 89].

The activities carried out at the LECA-STAR are very close to the classical R&D project. Scheduling R&D project is a complex process. This complexity lies mainly in the fact that the order and types of activities to be carried out during the project can vary enormously due to the results obtained in the early stages. To handle the R&D Project Scheduling Problem [153], researchers have used different techniques to include the uncertain in the scheduling models such as robust optimisation [81, 82, 95] or fuzzy optimisation [72, 130, 136]. In the same way that for nuclear-related scheduling

activities, most of the literature of R&D project scheduling deals with broad scheduling horizons.

Working with a relatively short scheduling horizon, which is the case in this PhD project, may reduce the subjectivity of the activities to be scheduled in a research project, thus allowing the use of an elementary activity approach, as proposed by Mancel [116] for the scheduling of research activities for the Mars Netlander project. In her approach, each experiment consists of a series of basic tasks or activities that are well defined (known duration, resource requirement, etc.). This elementary task approach allows us to use standard scheduling methods to schedule the activities of the research laboratory. Because of the short scheduling horizon of our problem, this is the approach we decide to use at the LECA-STAR. Another pragmatic reason for discarding uncertain approaches is that the required data to describe the activities uncertainty is not available.

1.5 Proposed improvement approach

The assumptions frequently made in the scientific literature when dealing with applications of combinatorial optimisation are that all the necessary information is available in time and in level of detail. However, as we saw before, this is not always true in real life. To be able to exploit combinatorial optimisation techniques at the LECA-STAR, we must ensure that the models we develop will have the information needed at the right time and detail level. For this, one can design an Integrated Scheduling Support System (ISSS) as the one presented in Figure 1.3.

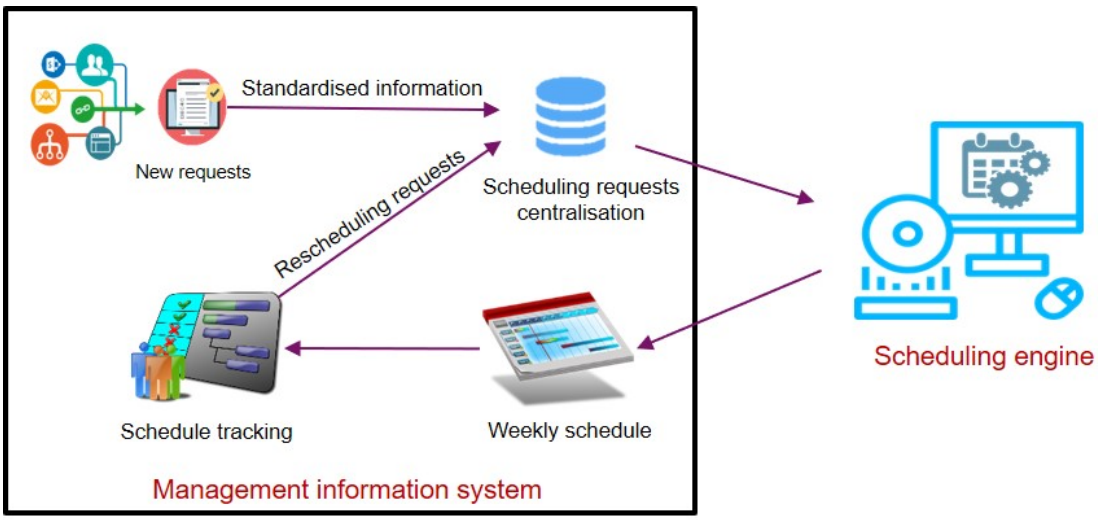

Figure 1.3: Integrated scheduling support system

This ISSS should be composed of a management information system that will centralise and standardise the information required to generate the schedule. A scheduling

engine will then exploit this information, using combinatorial optimisation techniques, in order to propose a valid schedule. The guideline for the development of the ISSS (see Figure 1.4 [1]) was developed in parallel of the management information support system and the scheduling engine, and integrate them later. What means that each part will be able to work without the other, but the integration of them will allow a better performance. However, as explained in Chapter 7, the information management system is not totally operational at the end of this research project but its exploitation should begin in a near future.

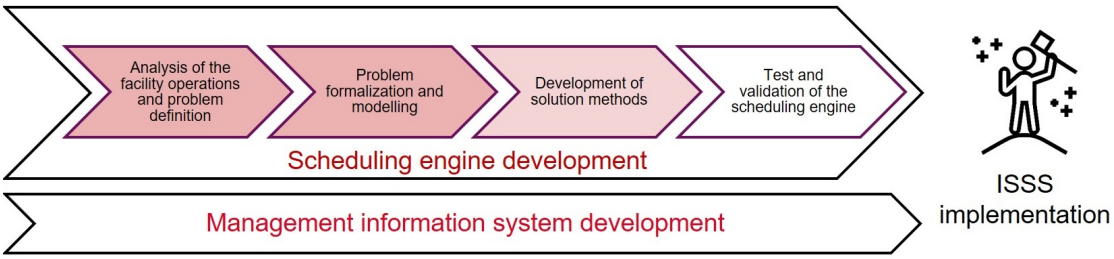

Figure 1.4: Project guideline

In the following of this section, we briefly describe the development process of the research project that allowed the development of the scheduling models and algorithms used by the scheduling engine, and presented in Part II of this manuscript. Finally, we describe some of the expected characteristics of the management information system.

1.5.1 Scheduling engine

The scheduling engine is the main topic of this dissertation. After analysing the characteristics of the scheduling process (Section 1.3.1), and contrast them against the classical scheduling problems of scientific literature (Section 2.1.4), we conclude that the scheduling problem at hand belongs to the class of *Resource-Constrained Project Scheduling Problems* (Chapter 3). To be more specific, it belongs to the class of *Multi-Skilled resource-constrained Project Scheduling Problem* or MSPSP (Section 3.2), where the resources are multi-skilled (also known as multi-purpose), and activities require a certain amount of staff member mastering a series of specific skills, which is one of the main characteristics of the activities carried out at the LECA-STAR.

We must adapt the classical MSPSP, to have a better representation of the LECA-STAR scheduling process. The main change is the fact of allowing the preemption of some activities since in the classical MSPSP, preemption is not allowed. However, for operational reasons, we must try to reduce the number of times an activity is preempted. So a first approach is to propose an MSPSP with penalty for preemption (Chapter 4), where a penalty is applied to the objective every time an activity is preempted. However,

[1] Of course in practice, the process for developing an application is not so linear and is very iterative

as we explain in Chapter 5, this first variant does not fulfil correctly all the requirements of the nuclear facility, what takes us to propose a most accurate variant: the MSPSP with partial preemption. This new variant allows the preemption of activities, but ensure that some of the resources remain allocated to the activity even when it is stopped. This way of handling preemption had not been studied in the literature before to the best of our knowledge.

The scheduling problem can be solved using a commercial solver over the Mixed-Integer Linear Programming (Section 2.2.1) or the Constraint Programming (Section 2.2.2) models we propose in Chapter 5, for medium-size instances. However, for larger industrial instances, this approach becomes less appealing as the MILP solver runs out of memory when trying to solve industrial-size problems, or MILP and CP may just take too much time to find good solutions. In order to answer the industrial need of having solutions quickly, we also develop some heuristic methods for the problem proposed in Chapter 6.

1.5.2 Management information system

The main function of this module is the centralisation and standardisation of all the scheduling requests. These requests can come from new demands, that must be transmitted using a predefined electronic form, or from deferred activities indicated at the schedule monitoring unit. Thanks to the use of an electronic form, we can ensure that all the information needed to run the optimisation models is given.

This information system should also allow sharing with all employees the last valid version of the schedule in real time. This ensures that in case of changes, everybody will be informed about avoiding mistakes. This will be a significant improvement, since that today only an offline version of the schedule is transmitted, and changes are difficult to communicate.

For the scheduling process, it is important to know in real time the status of activities and resources. This ensures that the scheduling engine has complete and up-to-date information when scheduling new activities, and will be able to react quickly to new events (breakdowns or other hazards). That is why the online schedule tracking should also be implemented. This should allow knowing the real-time execution status of the activities. Indirectly, one can know in real-time the status of the resources. In general, the online schedule tracking should lead to a better reactivity to unforeseen events, since these will be notified in a shorter time.

Having identified a time ago the need to monitor the status of execution of activities, a tool has been developed for this purpose. This tool will allow sharing the planning in real time, as well as to do a live follow-up of the progress of the activities and the operational status of the resources. The idea is then to focus the efforts in making operational this tool, and adding it the data centralisation function. The development of this tool is ensured by the engineer, and we work collaboratively to maintain coherence between the information system and the scheduling engine. We actively participated in

the tests and presentation phase of the new tool as a side project.

1.6 Concluding remarks

Research facility as the LECA-STAR plays an essential role in nuclear research and industry development. In order to guarantee that the laboratory responds to the needs of French researchers, it has to optimise its operation. An initial stage of this optimisation process is the improvement of the scheduling process. For the LECA-STAR, in the same way as for every hot laboratory, a good schedule not only allows better management of resources but also helps to improve the safety of the facility.

We focus our improvement project on scheduling the weekly activities carried out at the facility. The short scheduling horizon allows us to ignore the inherent uncertainty of the experimental projects, being able to apply standard scheduling techniques to the problem at hand. The characteristics of the scheduling process at the research facility lead us to conclude that a variant of the Multi-Skilled Project Scheduling Problem could represent the real situation of the laboratory.

Finally, we have briefly described how an Integrated Scheduling Support System would help to improve the scheduling process at the LECA-STAR. This system should be composed of two parts: A management information system in charge of the centralisation of the data required for scheduling, and the real-time monitoring of activities and resources; A scheduling engine comes to exploit this information and allows us to have a more robust and reliable schedule.

The scheduling engine is the main topic of this thesis and is presented in Part II, after a bibliographic review presented in Part I. Some aspects of the industrial implementation of the Integrated Scheduling Support System are presented in Part III.

Part I

State of the Art

Chapter 2

Combinatorial Optimisation Problems

Contents

2.1 Some classical combinatorial optimisation models 20

2.1.1 Knapsack problems 20

2.1.2 Network and graph problems 22

2.1.3 Location and routing problems 23

2.1.4 Scheduling problems 25

2.2 Classical solution methods 27

2.2.1 Mixed-Integer Linear Programming 27

2.2.2 Constraint Programming 30

2.2.3 (Meta)Heuristic methods 31

Let us start this chapter by defining a key concept. An *Optimisation Problem* involves maximising or minimising a function, by systematically choosing input values, taken from an allowed set (defined by a set of constraints), and calculating the value of an objective function to determine the "best" solution. More formally, we can define an optimisation problem as follows:

Definition 2.1. *Given a function $f\colon A \to \mathbb{R}$, $A \subset \mathbb{R}^n$, we must find an element $x_0 \in A$ such as: $f(x_0) \leq f(x)\ \forall x \in A$ for the minimisation problem or $f(x_0) \geq f(x)\ \forall x \in A$ for the maximisation problem.*

Usually, optimisation problems are modelled using a set of decision variables with a well-defined domain. According to the domain of these variables, we can classify the optimisation problems into three categories: (1) those using only discrete variables (that is, the domain of each variable consists of a finite set of values), (2) those that are modelled exclusively by continuous variables, and (3) those using both types of variables (mixed optimisation problems). Optimisation problems belonging to the first category, called *Combinatorial Optimisation Problems (COP)*, lie at the heart of this chapter. We will also include in the category of COP the problems of the third category such that fixing all the discrete variables yields a linear program. Indeed a linear program can be

seen itself as a polynomially-solvable COP where the search space is limited to the finite discrete set of the polyhedron vertices.

Combinatorial optimisation, also known as discrete optimisation, is a branch of optimisation in applied mathematics, strongly related to operations research and computer science. This discipline works with problems whose decision variables have a discrete and finite domain. As a result, a combinatorial problem has a finite number of solutions, hence the term *combinatorial*, although typically exponential in the number of variables. Neumann and Witt [128] formally defined a combinatorial problem as follows:

Definition 2.2. *A combinatorial problem can be defined as a triple* (S, f, Ω)*, where* S *is a given search space (set of all possibles values the decision variables can take),* f *is the objective function, and* Ω *is the set of constraints that must be respected to obtain feasible solutions. The objective is to find a globally optimal solution (*$s*$*) with the highest objective value for the maximisation problem or with the lowest objective value for the minimisation problem.*

Modelling real problems as combinatorial problems is, most of the time, easy; however, solving efficiently combinatorial problems represents a great challenge for researchers. One might think that a simple solution for this type of problem is to enumerate all the solutions of the search space; however, most of the time this approach is not feasible due to the combinatorial explosion (search space too vast to be explored entirely) of the problem. This difficulty in solving combinatorial problems has attracted the attention of researchers, who aim at the development of new solution techniques.

Combinatorial optimisation is maybe one of the youngest and most active areas of discrete mathematics, becoming a major research subject in the fifties [101]. There were two major changes that have stimulated combinatorial optimisation research: (1) the continuous increase in computing power that has enabled the development of more efficient algorithms, and (2) a growing confidence in the practical potential of combinatorial optimisation techniques. Combinatorial optimisation problems can be easily found in almost all fields of management and engineering such as: marketing [34, 117], supply chain [66, 162], scheduling [139], integrated circuit design [83], cryptography [91] and many others. A survey of some industrial applications of combinatorial optimisation is presented in [134].

2.1 Some classical combinatorial optimisation models

In this section, we present an overview of some classical combinatorial optimisation models and theirs application to real-life problems.

2.1.1 Knapsack problems

The easiest way to explain the concept of this problem is by making an analogy to a traveller that must pack the objects to be taken for a trip. The traveller must select,

from a set of possibles choices, the objects that maximise her/his comfort during the travel while respecting the maximum capacity of the knapsack. The knapsack problem (KP) can be formally defined as follows [93]:

Definition 2.3. *Given a set of items J, consisting of n items j with profit p_j and weight w_j, we must select a subset of J such that that the total profit of the selected items is maximised, and the sum of the weights of selected items does not exceed a fixed capacity C.*

If we associate a binary decision variable x_j to every item $j \in J$, taking the value of 1 if the item is included in the knapsack (0 otherwise), we can formulate an integer linear programming model for the KP as follows:

$$\text{maximise} \quad \sum_{j=1}^{n} p_j x_j \tag{2.1}$$

$$\text{subject to} \quad \sum_{j=1}^{n} w_j x_j \leq C, \tag{2.2}$$

$$x_j \in \{0, 1\}, \qquad j = 1, ..., n \tag{2.3}$$

Even if the KP might seem easy, being one of the simplest non-trivial integer programming model with binary variables [93], adding the integrality constraint (equation 2.3) puts the KP into the class of NP-hard, i.e. "difficult to solve" problems. The KP is important because it arises as a "sub-structure" for many other combinatorial optimisation problems [85]. For example, we find the so-called "knapsack constraint" (equation 2.2) in various resource-constrained scheduling problems as a constraint to respect the capacity of the resources (in addition to other constraints).

This is one of the most studied combinatorial optimisation problems, and several variants and extensions have been proposed in the literature to address optimisation problems coming from diverse fields. The *subset sum problem (SSP)* is a simpler variant of the KP where the objective is to fill as much as possible the knapsack, that means to maximise the total weight of the selected items while respecting the knapsack capacity. It has been largely used to develop cryptosystems, where an instance of the problem must be solved to decrypt the information [91].

Other variants propose to determine the number of similar items j to be included in the knapsack, instead of deciding whether each item is included or not (still respecting the maximum knapsack capacity). If we have limited availability of each item, the problem is called *bounded knapsack problem* (BKP). However, if the availability of each item is very large or infinite, we call it an *unbounded knapsack problem* (UKP). These variants are useful for problems where we have a given number of identical copies (same weight and profit) of each item j to be packed [93].

Baseline KP assumes that there is only one knapsack. However, sometimes we may

need to fill various knapsack in parallel, each of them with a given capacity. We face then a *multi knapsack problem* (MKP). In this problem, additionally to decide which objects will be included, we must also decide in which knapsack they will be included (always respecting the knapsack capacity). This extension has numerous applications, including task allocation among autonomous agents, multiprocessor scheduling and vehicle/container loading [71].

Suppose there is someone in the supermarket with a list of products he must buy, for each product on the list he has multiple brands from which he can choose. Each brand gives him a certain satisfaction level (profit) and has a specific price (weight). At the end, he must be sure of getting all the products in the list, regardless of their brand, while maximising his satisfaction and respecting his limited budget (knapsack capacity). This problem is an example of the *multiple-choice knapsack problem*, in which there is a fixed list of articles to put in the knapsack, we can choose between different choices (with different weight and profit) for each item, and the objective is to maximise the profit while respecting the capacity constraint. This problem can be useful in finance to decide between substitute inversion plans and capital budgeting.

Until now, the profit of every item is independent of the position in the knapsack. However, some real-life applications, such as choosing the location of products on the supermarket shelf (products on the middle have more chance to be bought) or scheduling commercials on broadcast television (advertisements aired at the start and end of a commercial break have more impact on customers), require that the profit of every item changes according to its position in the knapsack. This can be modelled using a *position-dependent knapsack* as proposed in [34, 59, 62]. Other variant modifying the behaviour of the objective function is the *quadratic knapsack problem* (QKP), where the profit of each item varies according to the presence (or absence) of other items in the knapsack [140]. A most recent variant is the *product knapsack problem* (PKP) [63], where the objective is to maximise the product of the profits of the selected items. PKP gains importance in the context of Computational Social Choices.

The *collapsing knapsack problem* (CKP) modifies the behaviours of the knapsack's capacity. In the CKP the capacity is no longer a scalar but a non-increasing function of the number of included items (inversely related to the number of items within the knapsack) [54]. Some applications of the CKP are satellite communications and time-sharing computer systems.

For a detailed review of the KP, its variants, applications and solution methods, we recommend the book by Kellerer *et al.* [93].

2.1.2 Network and graph problems

A network or graph can represent a wide variety of combinatorial problems. A network is defined by a set of nodes and a set of arcs (unidirectional connections) or edges (bi-directional connections) connecting those nodes, along with some information such as cost and capacity of arcs (if we do not have this information, we call it a graph) [85].

In a *network flow problem*, we must determine an optimal strategy for "routing" a flow through the network, always respecting the limited capacity of the connecting arcs (or the node according to the modelling approach). Flow problems can be used to represent applications with physical networks where commodities (or flow) must circulate through routes with limited capacity such as the distribution of commodities (i.e. gas, petroleum, water) [113, 166], or managing communication bus on computer systems [75].

Not all network flow problems are based on physical networks. The *assignment problem*, where one must minimise the cost of assign people to jobs, can be modelled as a network flow problem. For this, we create a network having two set of nodes (bipartite graph): a set of nodes representing the people to be assigned and the other other set representing the jobs. Each arc is connecting a person to a job that this person can execute [85]. In the classical assignment problem, one must assign one person to one job. A more general assignment problem, assigning more than one person to an activity, is used in the heuristic method presented in Chapter 6, to solve the technician allocation problem within the Multi-Skill Project Scheduling Problem with partial preemption (MSPSP-PP).

There are also many graph-based combinatorial optimisation problems that do not involve a flow, and they consider different properties of the analysed graph or network. The *vertex colouring problem* [115], requires to assign a colour to each vertex, in a given graph $G = (V, E)$, ensuring that colours on adjacent vertices (i.e. connected by an edge) are different, while minimising the number of used colours. This problem is useful, for example, for minimising the number of required frequencies needed to ensure wireless communication without interference [1]. Other graph-based problems are the *maximum clique problem*, where the objective is to identify the biggest subgraph, within the original graph, for which every node is linked to the other nodes in the subgraph; and the *minimum cut problem*, the objective of which is to find the smallest number of edges that, if removed from the graph, will divide the graph in two subgraphs. These problems have many applications in the fields of social network and communications networks [85].

Recently, Martins *et al.* [117] use a mixture of the maximum clique and minimum cut problems, known as *maximum cut-clique problem*, for an application to the market basket analysis. The problem is used for finding the products that are the most important in the market basket and then adapt marketing strategies. More application of graph-based optimisation problems can be found in [69].

2.1.3 Location and routing problems

Other network-based combinatorial problems require to find an optimal route through a graph while satisfying a set of requirements. The well known *travelling salesman problem* (TSP) is one of them. In TSP, the objective is to find the shortest walk through the network such that the walk starts and ends at the same node and visits every node of the graph exactly once [13]. This problem has application in very diverse fields such as

logistics, microchips manufacturing and DNA sequencing.

The TSP can be seen as a specific case of the *vehicle routing problem* (VRP), in which we must find a collection of k circuits, each of them corresponding to a capacity limited vehicle route, that minimise the total cost of visiting all nodes [65]. This problem is of high interest for researchers, as shown by the exponential growth of the number of publications on the subject in the last years [37], playing an essential role in logistics, supply chain management and delivering. A lot of VRP variants have been studied such as the *VRP with time windows*, where each customer (node) has an associate period (time window) in which he can be visited; the *VRP with pick up and delivery*, in which a number of products can be delivered or picked up to/from a customer (what affects the way vehicle capacity is modelled); and the *VRP with multiple depots*, where the deliveries can be done from multiple departs nodes.

Braekers *et al.* [37] highlighted in their work three relative new variants that include real-life constraints and assumptions (making the VRP more realistic): the *open VRP*, where vehicles are not forced to return to the initial depot after visiting the customer (useful for modelling third-party logistic deliveries); the *dynamic VRP*, in which the input data is updated continuously, and routes are adapted dynamically; and the *time dependent VRP*, in which time travel is a deterministic function of the current time. For a deeper lecture over VRP description, variants and solutions methods, we recommend [37, 46, 65].

Facility location problems represent also common combinatorial optimisation problems. The objective is to choose the optimal subset of locations, from a set of candidates, satisfying certain requirements and restrictions [85]. Within these problems, we found the *p-median location problem*, in which the objective is to find the location P in such a way that the total demand-weighted costs (distance or travel time) are minimised [118]. Sometimes, a client (node) cannot be served (covered) if it is located farther than an acceptable distance; these problems fall in the category of covering problem. The *location set covering problem* aims to reduce the number of required facilities to ensure a fixed level of coverage. The *maximal covering problem*, on the other hand, consists in maximising the level of coverage for a fixed number of facilities. Finally, the *minimax location problem*, also known as p-centre location problem, which seeks to minimise the maximum distance between any customer node and its nearest facility. Facility location problems have big importance in supply chain management [118] and emergency humanitarian logistics [35].

Problems mentioned before focus on locating the facilities as close as possible to customers. However, some real-life applications require locating some facilities that are undesirable to the nearby population such as waste disposal, recycling facilities, or even nuclear reactors. These applications can be modelled, for example, using the *antimedian location problem*, where we seek to maximise the average distance between the nodes and the facility; the *minimum covering location problem*, which aims to minimise the covered area (nodes within a fixed distance), subject to the condition that no two facilities are

allowed to be closer than a specific distance [29]; or the *anticentre location problem*, that maximises the minimum distance between the nodes and the facility. More details about location problem model, solution methods and applications can be found in [52].

For real-life applications, it is common to find models integrating the location and routing problems. Some applications of the location/routing integration problems can be found in [60, 148, 155].

2.1.4 Scheduling problems

Scheduling problems are concerned with the optimal allocation of limited resources to activities over time [84], taking into account the temporal constraints (delays, precedence relationship,...) and the capacity/availability of resources. A schedule should indicate the periods on which an activity is executed and the resources allocation.

In the *machine scheduling problems*, we must schedule a set of jobs that must be executed on a disjunctive resource (i.e. that can handle only one activity at a time), often called *machine*, aiming to optimise an objective function (minimise the maximum or average job tardiness or earliness, minimise the average time expended by the jobs in the system, ...). The *single-machine scheduling problem* is the simplest machine scheduling model, in which we are given a set of n jobs $j = 1, ..., n$, each of them with a specific duration p_j, to be processed on a single machine [40]. The single-machine models are crucial for developing decomposition solving methods for more complex scheduling problem, where the complex problem is decomposed into many smaller single-machine scheduling problems [138]. Sometimes a setup time for configuring the machine can be needed; if this setup time depends on the sequence (which activity is executed before a specific activity), we talk about a *machine scheduling problem with sequence-dependent setup times.*

If instead of having only one machine, we have a set of m machines $M_1, ..., M_m$ on which the jobs can be processed, we face a *parallel machine scheduling problem.* The jobs duration can be the same for all machines (identical machines) or can vary according to the machine. In the *multi-purpose machine scheduling problem* [41] we are given a set of n jobs $j = 1, ..., n$, and a set M of m parallel machines $M_1, ..., M_m$. Each job j has a set of machines $Mc_j \subseteq M$ to which it can be assigned. The objective is to find a schedule such that each job j is assigned to one of the machines in Mc_j and such that the makespan is minimised [107]. This problem combines the traditional machine scheduling problem and an allocation problem. It can be found in the literature under different names such as scheduling with eligibility constraints [107], scheduling with processing set restriction [107] or scheduling with flexible resources [53].

Usually, a job must require to go through more than one machine (stage) to be done; this is the case of the *shop scheduling problems.* In shop scheduling problems the jobs are divided into several *operations*, which have to be processed on different machines. The operations of the same job cannot be processed at the same time, and machines are disjunctive [40]. According to the order (route) on which the operations of

each job can be done, we can classify the shop scheduling problem in three categories: *flow-shop scheduling problem*, if the route of all jobs are identical; *job-shop scheduling problem*, in which each job may has a specific route; and *open-shop scheduling problem*, if the operations of jobs have not a fixed route to follow [139]. A generalisation of these problems is the *flexible shop scheduling problem*, for which, instead of having single machines at each stage, we have a set of parallel machines [138].

Models presented above, assume that each job requires only one machine (resource) at a time for its execution. However, a job may require the simultaneous use of various machines; the *multiprocessor scheduling problem*, also known as *multi-resource scheduling problem* [53], represents this case. In this problem, each job is associated with a subset of machines that are occupied simultaneously during the whole job execution. Furthermore, precedence constraints may be given between certain jobs [40]. A *multi-resource shop scheduling with resource flexibility problem*, mixing the characteristics of multi-resource and multipurpose machine scheduling problem, is presented by Dauzère-Pérès *et al.* [53]. In this problem, each job may need several resources to be performed, and furthermore, a resource may be selected in a given set of candidate resources. This scheduling problem is related to the *Multi-Skill Scheduling Problem* studied in Section 3.2.

Scheduling problems can also work over cumulative resources, i.e. they can handle more than one activity simultaneously. In cumulative resource scheduling, each job requires a specific amount of a resource, which has a fixed maximum capacity, during its whole duration. We must ensure that the sum of resource requirements in every period is always lower than or equal to the maximum capacity of the resources. The cumulative nature of resources makes it even harder to solve the scheduling problem.

The *resource-constrained project scheduling problem* (RCPSP), takes into account the characteristics of the multiprocessor scheduling problem and the cumulative resources scheduling problem, besides with a more general representation of the precedence relationships between the jobs (usually represented by a precedence graph). The RCPSP is a combinatorial optimisation problem that covers a large number of "classical" scheduling situations. The problem consists in scheduling tasks or activities on renewable resources with limited capacities. These tasks are linked together by precedence relationships (task i cannot start until task l is finished). The most common objective function is to minimise the makespan of the project ($C_{\max}$) [51].

Even if the RCPSP is easy to define, it is one of the most intractable scheduling problem [14]. It has shown to be a powerful tool to model a vast amount of real-life problems, having an important number of variants aimed to better represent the reality. A more in-depth analysis of the RCPSP characteristics, variants and solution methods will be given in Chapter 3.

2.2 Classical solution methods

2.2.1 Mixed-Integer Linear Programming

A lot of combinatorial optimisation problems require the optimisation of a linear objective function, having constraints that, in most cases, can be expressed as linear combinations of decisions variables. Furthermore, some optimisation problems with non-linear objective function can be "linearised" by approximating the nonlinear function by piece-wise linear functions [73]. Thus, these problems can be easily modelled using (Mixed-)Integer Linear Programming (ILP).

In general, integer optimisation is harder than linear (continuous) optimisation, and polynomial-time algorithms are unlikely to exist unless $P = NP$ [101]. In traditional linear programming (continuous) the feasible region is convex, while in integer optimisation is either a discrete set of points (for pure ILP) or, a union of possibly disjoint polyhedra (for MILP). In a convex region, we can ensure that any locally optimal solution is a global optimum. However, in integer optimisation, we must prove that a solution dominates all the others using other arguments than the approach of convex optimisation [85].

The equations of the MILP model do not provide a useful geometric description of the feasible region, and we must obtain a better description using polyhedral theory. Weyl's theorem implies that, if rational numbers describe the original problem formulation, there is a finite system of linear inequalities describing the convex hull of the feasible region ($conv(S)$). Optimising over the convex hull $conv(S)$ is equivalent to optimise over the search space S. We can then use classical convex optimisation approaches to solve the MILP over the convex hull $conv(S)$. The main idea is to identify a set of linear inequalities describing the convex hull $conv(S)$, which is called the ideal formulation, and then optimise over this convex region. However, in practice, the number of required linear inequalities is too large to be explicitly constructed [85].

Cutting plane algorithms, initially proposed by Gomory [74], use a partial description of $conv(S)$, generating the "strongest" inequalities (called *facets*). The algorithm first solves the linear relaxation of the problem (integer constraints have been relaxed). If the relaxed problem is unbound or infeasible, then the MILP also is. If we get an integer solution, we have solved the MILP to optimality. However, if we get a fractional solution, we must find a valid inequality (*cut*) that separates the fractional point from the polyhedron $conv(S)$ (the problem of finding such cut is called *separation problem* or *facet identification problem*). This inequality is added, and the problem's relaxation is solved again. Theoretically, this algorithm could finish in two ways: either an integer solution of the relaxed problem has been found (what means the MILP has been optimally solved) or the relaxed problem is infeasible (thus the MILP is also infeasible). In practice, the algorithm can finish in two additional ways: a) the algorithm is not able to find another cut (because a full description of the convex hull is not known, or because of accumulated round-off errors) [73]; b) the algorithm must be interrupted since it takes too much time

to converge.

Even if one of the last two option happen, the results of the algorithm are still useful, since it provides better bounds for the problem that can be used later by other solution methods such as *Branch-and-Bound*. Although the cutting plane methods are in general not polynomial in time, they are largely used due to their success in practice for pruning the search tree (see below) [106].

Branch-and-Bound (B&B) methods imply the implicit enumeration of all possible solutions of the problem, by storing partial solutions in a tree structure. In each node, we generate "child" nodes, partitioning the solution space into smaller regions (*branching*), and we use some rules to prune regions that are suboptimal (*bounding*). After visiting the whole search tree, the best solution found is returned and can be considered as optimal [122].

There are three components of Branch & Bound methods that have a significant impact over the performance of the algorithm: the way of separating the problem into subproblems (*branching strategy*), the order in which the subproblems will be explored (*search strategy*), and how to limit the exploration of suboptimal solutions (*pruning rules*). Morrison *et al.* [122] present a description on how these components influence the algorithm performance.

A usual pruning rule is to solve a linear relaxation of the integer problem on the nodes to get bounds. If the obtained bound is worst than the best current solution, the branch is pruned. The bounds obtained by the LP-relaxation are often weak [73], leading the algorithm to visit too many suboptimal branches. As stated before, adding cutting planes to the problem can improve the quality of the bounds. We can then combine the cutting planes and Branch-and-Bound algorithms to get a more powerful class of algorithms known as *Branch-and-Cut*. The main idea of the Branch-and-Cut is to add cuts to the subproblems on the nodes, to improve the bounds obtained from the linear relaxation [85].

Another possible way to solve MILP models involves the decomposition of the problem into multiple sub-problems that are in theory easy-to-solve, and that must be solved iteratively. The *Lagrangian decomposition* [67], for example, consists of isolating the set of difficult constraints to obtain separate subproblems that are solved over each constraint subset. Linking variables are created to link the subsets. Most of the decomposition strategies in the literature involve decomposition based on constraints [85]. However, for problems having an exponentially growing number of variables, it is more interesting to use a decomposition method based on variables such as the *Benders decomposition* [150]. In this approach, we fix the value of the "complicating variables", and the resulting subproblem is solved iteratively. Exploiting the associated dual of the problem, the algorithm must find a cutting plane for the current solution and add it to the problem before solving it again [73]. The *Dantzig-Wolfe decomposition* aims to replace any integer solution of a set of constraints by a convex combination of the extreme points of the convex hull. The original problem is then reformulated into a master

problem and n smaller subproblems. Most of the time, Dantzig-Wolfe decomposition results in a master problem with a huge number of variables, and column generation is used to solve it [149]. In general, decomposition methods provide bounds for the MILP problem that are tighter than the bounds obtained by simple linear relaxation. They can then be used to improve the pruning rules in a B&B algorithm.

The *Branch-and-Price* is another improved method derived from the B&B, using Dantzig-Wolfe decomposition. In this approach, we start with a restricted version of the master problem, obtained by Dantzig-Wolfe decomposition, and start adding variables (column generation) with the potential of improving the value of the objective function. To select the column to be included, we solve one or various *pricing problems* over the dual of the Dantzig-Wolfe subproblems, to find the columns with negative reduced cost. The main inconvenience of this technique is that the pricing problems are usually hard to solve (NP-Hard). Additionally, branching strategies can interfere with the structure of the pricing problems; the use of alternative branching strategies are then necessary [122]. Recent research efforts have been made to mix the Branch-and-Cut and Branch-and-Price algorithms into a new method known as *Branch-and-Cut-and-Price* [57].

The time required to solve a combinatorial optimisation problem to optimality depends strongly on the way the problem is formulated. Since the same problem can be formulated in various ways, a variety of challenging problems have been solved by reformulating them, and a lot of research works study the way of better formulate classical optimisation problems [85]. Artigues [15], for example, presents a deep analysis of the strength of time-indexed formulations for the RCPSP. Indeed, different MILP formulations of the same problem can be theoretically compared in terms of their tightness, i.e. their proximity with the ideal formulation. Roughly and for pure 0–1 problems, to show that a formulation $Ax \leq b$, $x \in \{0,1\}^n$ is tighter than a formulation $By \leq c$, $y \in \{0,1\}^p$ it suffices to show that, given a linear transformation $y = Ex + f$, constraints $Bt \leq c$, $y \in [0,1]^p$ are implied by constraints $Ax \leq b$, $y = Ex + f$, $x \in [0,1]^n$, $y \in [0,1]^p$. This shows that the first formulation is at least as good as the second one. Then a single example were the LP relaxation value of the first formulation is strictly better than that of the second formulation suffices to show that the first formulation is strictly tighter. In this thesis, we use this technique to make (limited) theoretical comparisons of the proposed MILP formulations for the studied problems in Chapters 4 and 5.

A better formulation allows us to get better bounds, and thus it should allow a faster computational solution. It is important to say that in integer optimisation the number of variables and constraints may not be an indicator of the difficulty of the problem [85]. Sometimes, it may be interesting to add some variables or constraints to have a better formulation. However, this has to be mitigated by the fact that when a tighter formulation has a large number of variables and/or constraints, the solving time of its LP relaxation may be slower than that of a less tight formulation involving fewer variables and constraints. Consequently, solving to optimality a problem by branch and bound may sometimes take more time with the tighter formulation even if the number

of nodes is reduced. This phenomenon occurs frequently in modern branch and bound solvers, in which even two MILP formulations that are theoretically equivalent in terms of relaxation tightness may obtain significantly different performances due to internal preprocessing, cut generation and heuristic procedures behaving differently. We observe this phenomenon for our problem in Chapter 5.

2.2.2 Constraint Programming

Constraint programming (CP) is a more emergent, *easy-to-model*, technique to solve complex combinatorial problems using a declarative description; it comes from logic programming and artificial intelligence. The idea is to separate the constraint declaration using a rich constraint language from the solution finding process based on an active use of constraints to reduce the search space (*constraint propagation*). CP is based on *constraint satisfaction problems* (CSPs), in which one needs to solve a constraint network, assigning values to variables in such a way that all constraints are satisfied [30].

The philosophy of CP is to allow users to use a more natural language to describe the constraints of the problem and then use a general-purpose constraint solver to tackle the problem [154]. One of the most important advantages of CP is that it allows using very diverse types of variables and constraints. Unlike MILP, CP models are not limited to use numerical variables. To model scheduling problems, for example, it is common to use *interval variables* that indicate the interval time over which an activity is executed (as we do for modelling the MSPSP-PP in Section 5.3). Using this kind of variables, we can define compactly and easily the constraints of the problem that will be difficult to model on MILP, such as: not overlapping of interval variables for disjunctive resources, for which commercial solvers have already excellent propagation algorithms.

The *constraint propagation* is a critical aspect of CP, since it allows reducing the search space by the iterative use of algorithms identifying the values of the variables to be deleted from the search space (*filtering algorithm*). To remove these values, the filtering algorithm should prove that they are *inconsistent*. We say that a value v of a variable is inconsistent with respect to a constraint C if, after fixing such value for the variable, it is not possible to assign values from their domains to the remaining variables that satisfy the constraint [160]. Lots of research works have been done to develop efficient filtering algorithms, allowing a better constraint propagation. However, more complex algorithms will require more time to propagate the constraints. It implies to search for a tradeoff between the complexity of the filtering algorithm and its ability to reduce the search space.

The most common way to solve CP problems is by using a combination of constraint propagation and a search method (similar to B&B framework) [160]. Usually, we use a *backtracking search* with a depth-first search strategy: it starts at the root of the tree and proceeds by descending to its first descendant until a leaf is encountered (a complete solution of the problem); the search then moves back to the parent node of the leaf and

explores another not yet visited branch. In each node, an uninstantiated variable (x) is selected, and a branching strategy is applied to generate the new branches. The most common strategies are to generate a branch for each possible value of the variable domain or to generate two branches: one for a forbidden value in the domain ($x \neq a$) and the other allocating the value to the variable ($x = a$) [154]. Constraint propagation is then executed. Removing inconsistencies allows pruning dead-end branches and reducing the variable domains. We can say that the constraint propagation mechanism in CP plays a similar role than linear relaxation for MILP [105].

Different approaches can be used to improve the performance of the algorithm. One can, for example, add some particular constraints (called *nogood constraints*). These constraints can be added using three main techniques: to add them during the modelling phase of the problem; to automatically add them using constraint propagation algorithms; or add them when an inconsistency or dead-end is found [154]. This third option has some similarities with cutting planes techniques for the MILP, since constraints are added after infeasibilities are found. One can also modify the way the backtracking is done or change the order in which variables are considered for generating the search tree.

Although this paradigm has been conceived to allow easy and fast modelling, a deeper analysis of the factors affecting the model performance (compactness of the model, elimination of symmetries, efficiency of propagation and search algorithms, ...) is necessary to take maximum advantage of the capabilities of the method. This analysis requires a good understanding of the problem characteristics and CP techniques. In this thesis, we propose a CP formulation of the considered problem in Chapter 5.

2.2.3 (Meta)Heuristic methods

Heuristic methods are handy for hard-to-solve problems or when problem instances are too big. A *heuristic* is a problem-dependent algorithm aiming to get good solutions (not necessarily optimal), in a reasonable time, for hard optimisation problems; this is done by exploiting the characteristics of the problem. A *metaheuristic*, on the other hand, is a problem-independent framework, consisting of a set of fundamental concepts and guiding subordinate heuristics (problem-dependent) to produce good solutions for a problem [133]. Thus, metaheuristics are easily adaptable and exploitable for an important number of problems.

To make a (non-exhaustive) classification of metaheuristics, we can divide them into two categories:

- single solution based (improvement is made using one solution at the time), and
- population-based (using a set of initial solutions).

Single solution based metaheuristics can be also subdivided into two categories: *construction methods* and *neighbourhood search methods* (also called trajectory methods) [36].

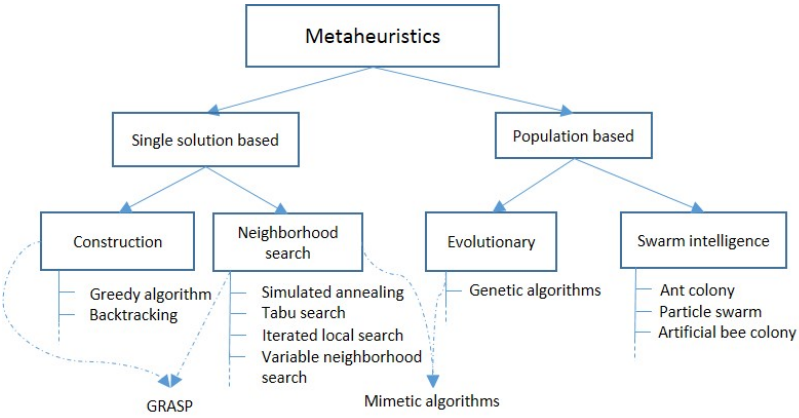

Figure 2.1: A non-exhaustive metaheuristics classification.

Construction methods. The construction methods are one of the oldest techniques in combinatorial optimisation. The main idea is to build step by step a solution. Starting from an initially empty partial solution, it seeks to extend at each step the partial solution of the previous step. The most common constructive heuristics are the *greedy algorithms*, in which we construct a solution by making the locally optimal choice at each stage, without questioning the choices made in previous steps. The principal drawback of constructive algorithms is they may accept some myopic choices leading to local optima that may be far away from a global optimum [161]. Chapter 6 proposes and compares experimentally different greedy algorithms.

Neighbourhood search methods. One way to improve the solutions obtained by constructive methods is to apply neighbourhood search. A typical neighbourhood search based method starts with a single initial solution and executes an iterative process to modify the initial solution, taking into account the objective function, to find a better neighbour of the initial solution. This process is repeated until the stop condition is met (a time limit, a maximal number of iterations). The *neighbourhood* of each solution s is defined by the subset of solutions ($N(s) \subseteq S$), which can be reached from s in the next step. Larger neighbourhoods lead to a better quality of the locally optimal solution [7]. *Very large-scale neighbourhood search* techniques, such as the *Large Neighbourhood Search*, aim to exploit this characteristic. A neighbourhood is considered to be very large-scale when it grows exponentially with the instance size of the instance or when it is too large to be searched explicitly in practice [141]. Some examples of neighbourhood search methods are *simulated annealing* (SA) and *taboo search* (TS).

Simulated annealing is an analogy of particles behaviour during metal annealing. At high temperatures, the particles can move easier, and they go to a static state as the temperature goes down. The algorithm starts with an initial solution s (generated randomly or by a construction method). At each iteration, a neighbour of s is selected

randomly. The algorithm always accepts improving solutions. Worsening solutions are accepted probabilistically according to the deterioration level and a parameter called *"temperature"*. The temperature parameter decreases over time, and thus the probability of accepting worsening solutions. The algorithm can be stopped according to different conditions: after a limit time, after a fixed number of iterations, after a fixed number of iterations without solution changes, or after reaching the final temperature. Many variants of SA have been proposed, modifying the way the temperature decreases or the acceptance conditions, for example. Franzin and Stützle [70] present a good analysis of SA variants and parameters.

SA is not able to learn from past solutions, which means that a solution could be visited several times. *Tabu search* (TS) tackles this problem by using an adaptive memory of visited solutions (*tabu list*) to guide the search process [133]. The memory helps avoiding possible cyclic search and escaping from local minima (unlike SA that uses randomisation). At each iteration, starting from an initial solution s, the algorithm generates a set of neighbours of s (avoiding the solutions in the tabu list) and keeps the best solution (s') within the set (even if s' is not better than s). The new solution is then added to the tabu list. At the end, the algorithm returns the best-found solution during all the iterations [12]. Storing a complete solution can be memory prohibitive, and inefficient as different solutions may share the same critical component that should be avoided. Usually, the tabu list is restricted to a fixed number of solutions and only some characteristics of the visited solution are stored. How the set of neighbours is generated in each iteration and the tabu list management are critical components of TS.

Greedy Randomised Adaptive Search Procedure (GRASP) is an iterative multi-start algorithm, that combines the characteristics of both constructive and neighbourhood-based algorithms. Each iteration of GRASP consists of two phases: construction and local search. In the construction phase, a feasible solution is generated using a greedy randomised algorithm. During the execution of the greedy algorithm, instead of always choosing the best local candidate, we choose a candidate randomly from a *restricted candidate list* (RCL). In the original GRASP, the probability of choosing each candidate is uniformly distributed. One can, however, use a memory component to bias the probability function and give priority to the intensification (generation of solutions with similar characteristics to the best solutions) or diversification (avoid solutions with similar characteristics) of solutions [152]. Once a solution is constructed, the local search algorithm explores its neighbourhood until a local optimum is found. The best solution found overall GRASP iterations is kept as the final result. A GRASP method is described in Chapter 6.

Population-based metaheuristics, also known as nature-inspired, can be divided into two main groups: *evolutionary algorithms* and *swarm intelligence algorithms*.

Evolutionary algorithms. The Darwinian evolution theory inspires these algorithms. They all share the idea of simulating the evolution of individuals (solutions) through selection, recombination, mutation and reproduction processes. Every evolutionary algorithm is constituted by three main components: a "population" of potential solutions (individuals); a fitness indicator of the adaptation of individuals; and an evolution mechanism allowing the creation and deletion of individuals [36]. *Genetic algorithms* (GA) are the most well-known evolutionary algorithms. GA starts with a population of initial solutions called "chromosomes" and uses a set of "genetic operators" randomly over one or two individuals (mutation and crossover) to make the population evolve. The *crossover* operator generates two children from two individuals (parents) by combining somehow its characteristics. The *mutation* operator generates a new individual by modifying the characteristics of one parent. To maintain a fixed number of individuals in the population, at the end of the iteration, only individuals (parents and children) with the best fitness indicators are kept [56].

Even if evolutionary algorithms decrease the chances of being stagnated in a local optimum, they are not able of improving a solution that could lead to a local optimum in their neighbourhood. *Memetic algorithms* (MA) try to achieve a synergy between the ability of evolutionary algorithms to avoid being blocked in a local optimum and the fine tuning ability of neighbourhood based algorithms. An MA is usually formed by a modified GA improved with a neighbourhood search algorithm. At each iteration, after the reproduction phase (crossover or mutation), the individuals are improved by executing a neighbourhood search algorithm on them [50].

Swarm intelligence algorithms These algorithms are inspired by the collective behaviour of a group of individuals or "collective intelligence". This intelligence is built up using a population of homogeneous individuals that interact with each other and with their environment [126]. These agents usually have minimal individual capability, but jointly can perform very well. *Ant colony optimisation* (ACO) is the most known in this category. ACO is inspired by the way real ants determine the optimal path from the nest to their food source. At first, ants choose randomly the path to follow and pheromones are left for guiding other ants. After some time, the optimal path should have a higher concentration of pheromones. ACO is useful for solving optimisation problems involving some sort of graph, e.g., TSP and VRP. At each iteration of an ACO, one constructs a solution step by step, at each step a partial path is randomly selected based on a probability function (pheromones). Once completed, the candidate solution is used to modify the probability function of each partial path in such a way that future sampling is biased toward high-quality solutions.

In the last years, the number of research works proposing the use of algorithms combining the characteristics of several metaheuristics has increased. These methods are known as *hybrid metaheuristics*. The main idea of hybrid metaheuristics is to exploit

the complementary features of different methods. Choosing an adequate combination of algorithms can significantly improve its performance. More general hybrid approaches look to integrate metaheuristics with other optimisation techniques such as constraint programming, mixed-integer linear programming or dynamic programming. Various surveys about hybridization can be found in [21, 135, 151]. In particular, the large neighbourhood search (LNS) framework hybridises local search with an exact method. Indeed, due to the large neighbourhood size, the search for the best neighbour is itself an optimisation sub-problem solved with dedicated algorithms or general techniques such as CP and MILP [86]. We use LNS in Chapter 6.

After this overview of the Combinatorial Optimisation methods and applications, we focus our literature review to the project scheduling problems in the following chapter.

Chapter 3

Project Scheduling Problems

Contents

3.1 Resource-Constrained Project Scheduling Problem (RCPSP) . . 38

3.1.1 Some variants and extensions 39

3.1.2 Modelling and solution approaches 41

3.2 Multi-Skill Project Scheduling Problem (MSPSP) 44

Project scheduling is an essential aspect of project management since it allows to have guidance and a pathway for the project. A project schedule must indicate a timescale and a sequence for the activities within the project; along with the allocation of people and materials at each stage of the project [18].

In the late 50s, the first techniques to schedule projects with deterministic activities duration, like the Critical Path Method (CPM) [90], were proposed [18]. Almost at the same time, the Program Evaluation and Review Technique (PERT) was developed by the United States Navy. PERT incorporates uncertainty by making it possible to schedule a project while not knowing precisely the duration of all the activities. For this, one uses different estimations of the activities duration: the optimistic time estimate, the most likely or typical time estimate, and the pessimistic time estimate. More sophisticated techniques have been developed to have a better representation of the project's uncertainty such as the Graphical Evaluation and Review Technique (GERT). Unlike the CPM and the PERT, the GERT allows the representation of probabilistic branching of the project. This is particularly useful for projects where the path to follow depends on the results of early stages (e.g. R&D or research projects). Recently, fuzzy project scheduling approaches have been largely used when the statistical information is hard to find, like in the Fuzzy GERT (FGERT) [72, 111] and, the more recent, Parallel and Reversible-Fuzzy GERT (PR-FGERT) [130].

Nevertheless, the aforementioned techniques are helpful only when resources are not constrained [157] and when uncertainty matters. Most of the time, during the project scheduling process, we must take into account some technological (machines or people availability) or financial (budget) constraints. Furthermore, as mentioned in Chapter 1, the uncertainty is reduced in our context due to the short term planning horizon. This leads us to work over a deterministic Resource-Constrained Project Scheduling Problem (RCPSP), which is the subject of this chapter.

3.1 Resource-Constrained Project Scheduling Problem (RCPSP)

The resources constraining the project execution can be classified into four categories [98]:

- *Renewable* resources are constrained on a period-by-period basis, i.e. the available capacity is renewed from period to period. Examples of this kind of resources are machines, equipment and people.
- *Nonrenewable* resources, also known as consumable resources, impose a constraint over its global utilisation for the entire duration of the project. Typical examples are the raw materials and the capital budget allocated to the project.
- *Doubly constrained* resources are constrained on a period-by-period basis and also on the global utilisation for the total project duration. We can take the example of the budget of the project again and add a constraint stating a restriction over the amount of money used at each period. Each doubly constrained resource can be represented by a pair made of a renewable resource and a nonrenewable resource.
- *Partially renewable* resources are a generalisation of both renewable and nonrenewable resources. Each partially renewable resource have a limit utilisation within a subset of periods of the planning horizon. Partially renewable resources are largely used to model labour regulations. A typical example, for a project with a scheduling horizon of a month, is the workers for which the maximal weekly working time is fixed by law.

Hartmann and Briskorn [79] present a classification of less common resource types such as *continuous resources*, *dedicated resources* or *resources with time-varying capacity*. The last kind of resources will be useful to model the scheduling problem of the nuclear facility, and they will be part of our discussion later in this chapter.

In the most current version of the RCPSP, we must schedule "non-preemptive" tasks or activities (i.e. once started, an activity must run continuously until its completeness) on renewable resources. The idea is to find a solution (a schedule) that minimise or maximise a given objective function, while respecting both the constraints of precedence relationships among the activities and the resource constraints (the sum of the resource needs of the activities in execution must not exceed the capacity of resources in each period of time) [14]. The main characteristics of the traditional RCPSP are presented in Figure 3.1.

More formally, the RCPSP is defined by a 7-tuple (V, p, E, R, B, b, H) where V is a set of activities, p is a vector of activity durations, E is a set of precedence relationships, R is a set of resources, B is a vector of resource capacities, b is a matrix of resource demands or consumption per activity, and H is the set of scheduling periods [14].

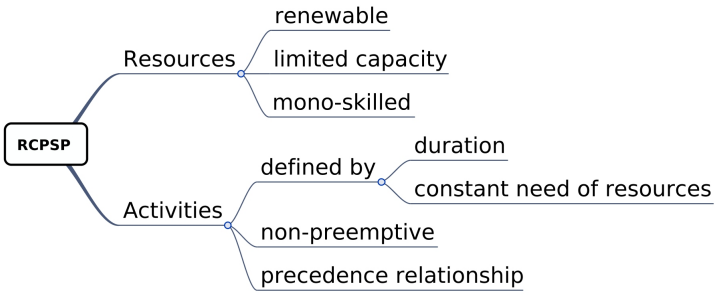

Figure 3.1: Characteristics of the classical RCPSP

Many objective functions have been used for the RCPSP. Most of them can be classified into three categories: *time-based objectives* (within which we found the most studied in the scientific literature: the minimisation of the makespan ($C_{\max}$) [76]), *cash flow or cost-based objectives* (such as Net Present Value maximisation [108]) and *robustness-based objectives* (used for scheduling under uncertainty [38]). Since this research project does not consider any cash-flow and uncertainty is not taken into account, we limit our research to time-based objective functions, especially the $C_{\max}$ minimisation (our main objective).

3.1.1 Some variants and extensions

Although the classical version of the RCPSP is very powerful, being able to model a large number of scheduling problems, it can not cover all the situations that can happen in real life problems. That is why researchers have developed more general versions of the RCPSP using the classical version as starting point [79]. For a more exhaustive lecture about the variants and extensions of the resource-constrained project scheduling problem, we recommend to read the surveys on this topic published by Harmann and Briskorn [79], Orji and Wei [131], and Habibi *et al.* [76]. Among all these variants, we distinguish a few ones that are of great interest for the modelling the problem at hand; they are described below.

Typically, time windows (i.e. the interval within an activity can be scheduled) are not taken into account in basic RCPSP. In real life project, the earliest starting time of an activity may be conditioned by an external constraint (e.g. wait for the arrival of raw material). This constraint is known as *release date* (r_i), and it states that an activity cannot start before this date. Sometimes, one must also respect some milestone of the project, i.e. activities must be ended before a specific date. This is known as a *deadline*. This constraint is called *due date* when the date fixed for the milestone of the

project can be violated. Some applications of time windows for the RCPSP, and its variants are presented in [4, 22, 79].

The classical version of the RCPSP assumes that, once started, an activity must run continuously until its completeness. However, in practice, it may happen that an activity can (or must) be preempted to be finished later. The inclusion of preemption may lead to reductions in the total duration of the project, especially when resource availability is very limited [23]. On the other hand, including the preemption of activities increases the number of possible solutions and consequently the computational complexity of the problem.

Most of the time, preemptive scheduling problems assume that activities can be interrupted and resumed later without any consequence [23]. This assumption does not represent industrial reality [24, 159], because of the presence of setup costs or times needed for resuming the activity or also because of the reduction in the production rate. To address this issue, different approaches have been proposed. Afsnar-Nadjafi [4] proposed a Preemptive RCPSP with penalty for preemption. In this problem, preemption is allowed, but a penalty is incurred every time an activity is preempted. Afsnar-Nadjafi and Mejlesi [5] also considered the inclusion of an indivisible setup-time for resuming a preempted activity. Another approach consists of limiting the number of times an activity can be preempted. Zhu *et al.* [167] presented a genetic algorithm for a preemptive RCPSP where each activity could be interrupted at most M times. Ballestín *et al.* [167] proposed a more general approach where each activity has its own preemption policy: if it can be preempted or not, the maximum number of times it can be preempted, and the minimal duration of each "part" of the activity. The work presented in [103] considers together: (1) a maximal number of preemptions, (2) a minimal duration of each part of the activity and (3) a minimal setup time for resuming a preempted activity.

A concept related with activities preemption is the *calendarisation* or *calendar constraints*. Calendar constraints make some resources unavailable during certain periods (e.g. unavailability of staff during the weekend). The RCPSP with calendarisation allows the preemption of activities only during the *calendar breaks* (i.e. when resources are not available). Some applications of this variant can be found in [45, 102]. Calendar constraints can be seen as a specific case of time-varying resource constraints. The main difference is that calendarisation assumes that the capacity of the resources for all periods, different to calendar breaks, is constant, while in time-varying resource constraints the capacity can change (not necessarily to zero) at each period. Applications of the RCPSP with time-varying resources capacities are presented by Hartmann [78] and Habibi *et al.* [77].

The basic RCPSP hypothesises that activities can only be performed using one method, which is determined by a fixed duration and fixed resource requirement. In

the *Multi-mode RCPSP*, an extension of the classical problem, an activity can be executed in several alternative modes. Each of these modes implies specific durations and resource requirements [129]. According to the survey of Habibi [76], the multi-mode characteristic is the most common in the literature. Most of the time the multi-mode characteristic is found together with other variants (the same is true for all the variants presented in this chapter). Buddhakulsomsiri and Kim [42], for example, mixed the multi-mode RCPSP with the calendarisation. Azimi and Azouji [20] propose a solution approach, using simulation, for the preemptive version of the multi-mode RCPSP. Najid and Arroub [125] added to the multi-mode RCPCP some time windows constraints. More examples and applications of the multi-mode RCPSP are presented in the survey of Noori and Taghizadeh [129].

Another variant, related to the multi-mode RCPSP, that also allows assigning resources according to different ways is the Multi-Skill Project Scheduling Problem (MSPSP) where resources can be assigned to different kinds of resource requirements of activities [28]. This variant is the basis for developing the models proposed in this thesis and is analysed later in this chapter.

3.1.2 Modelling and solution approaches

The most common technique for modelling the RCPSP and its variants is the integer (ILP) or mixed-integer (MILP) linear programming since it allows us to have a formal description of the problem's constraints easily. However, as stated in Section 2.2.1, the way of modelling the problem has a significant impact on the effort required to solve it. We can distinguish two families of formulations for the RCPSP [100]:

- *Discrete-time formulations* divide the planning horizon into uniform time intervals, and events can only occur at predefined time points. These formulations make use of time-indexed binary variables [15], so they are highly sensitive to the time horizon.

- *Continuous-time formulations* suppose that events can happen at any point of the scheduling horizon. They rely mainly in the use of precedence-based (like the flow-based formulation presented in [16]) or event-based decision variables (such as the event-based MILP formulations proposed by Koné *et al.* [99]). Some continuous-time formulations also made use of the concept of valid antichains, i.e. all feasible subsets of activities that can be simultaneously executed without violating resource or precedence constraints [119].

Looking at the characteristics of our study case, we conclude that a discrete-time formulation could be the most accurate approach to model the problem at hand (especially for modelling the time-varying resource availability and activity preemption). We can identify three well-known time-indexed formulations:

- *Pulse formulation:* uses the binary pulse variables $X_{i,t}$, $i \in V$, $t \in H$, are such that $X_{i,t} = 1$ if activity i begins at period t, zero otherwise. An activity that begins at t is to be interpreted as the fact that the activity is ongoing during the interval $[t, t+1]$ while it was not in progress at the time interval $[t-1, t]$ if $t > 0$.

- *Step formulation*: uses the binary step variables $Z_{i,t}$ such that $Z_{i,t} = 1$ if the activity i begins at time t or before. For a given activity i, the variables $Z_{i,t}$ with $t <$ starting time of the activity are all equal to 0, whereas the variables with $t \geq$ starting time are all equal to 1.

- *On/Off formulation:* uses binary variables $Y_{i,t}$, where $Y_{i,t} = 1$ if activity i is being executed at time t and $Y_{i,t} = 0$ otherwise.

Artigues [15] proved that this three time-indexed formulations are equivalent in terms of their linear relaxation strength, and that their *disaggregated* versions are stronger than their *aggregated* versions. However, in practice, the performance of formulations is not necessarily related to the LP relaxation. The time-indexed formulation proposed by Bianco and Caramia [31, 32], that introduces a new decision variable indicating the fraction of the activity that has been processed at each time period, generally outperforms other time-indexed formulations in terms of solution time. Artigues [15] proved that the theoretical linear relaxation strength of Bianco and Caramia's formulation is equivalent to the classical disaggregated time-indexed formulations.

The MILP models presented in Chapter 4 and Chapter 5 are inspired on the time-indexed formulations, more precisely on the On/Off formulation and the Step formulation.

Constraint programming has also been successfully applied to solve RCPSP like in [55], where a combination of linear programming and CP is used to calculate lower bounds for the RCPSP. The authors use constraint propagation to derive new valid inequalities (cuts, see Section 2.2.1) that are added to the LP relaxation of the problem to improve the quality of the lower bound. Liess and Michelon [110] proposed a filtering algorithm that does not use the commonly used approach called "resource timetable" (where each of the resources is associated to a timetable indicating the amount of the resource still available at each time $t \in [0, T]$). The authors substituted the resources constraint by a set of sub-constraints, each of them corresponding to a set of tasks that *cannot* be executed together without violating the resource capacity constraints. Schnell and Hartl [156] propose a more recent application of constraint programming for the multi-mode RCPSP. Their approach combines techniques of CP and *Boolean satisfiability problem (SAT)* principles. Computational experiments showed that the proposed method outperforms the state-of-the-art exact algorithms, being the first to close an important number of open instances from the literature. Kreter *et al.* [102] proposed and tested six different CP models for the RCPSP with calendar

constraint, and they showed that CP solutions are highly competitive with existing MILP formulations of the problem (average runtime required by CP models is lower). These results have motivated us to propose a CP model for our study case in Chapter 5.

RCPSP is NP-Hard [33], and the exact state-of-the-art methods can only solve instances with at most 60 activities [135]. Since a real-life project can quickly exceed this size, a large number of (meta-)heuristic methods have been proposed for the RCPSP and its variants. Surveys about this topic are presented in [3, 97, 135].

Priority-rule-based heuristics, also known as list schedule generation heuristics, seem to be the most used for the RCPSP [17]. These heuristics are made up of two components: a *Schedule Generation Scheme (SGS)* and a *priority rule* indicating the order in which activities are treated during the scheduling process. We distinguish two types of SGS: *serial* and *parallel*. Both methods generate a feasible solution by extending a partial schedule in each iteration. In the serial SGS, at each iteration, the first activity in the priority list is selected to be scheduled (inserted to the partial schedule) as soon possible (respecting precedences, resource constraints and release dates). Once a starting time is assigned to the activity, this time is permanent and cannot be changed by the subsequent iterations of the method (the same is true for parallel SGS). The parallel SGS, on the other hand, works chronologically. At each period t, the set of activities that can be scheduled at that period (*schedulable activities*), regarding the resource availability and the precedence relationships, is generated. Activities within the set are selected one by one to be scheduled at time t following the priority rule; this is executed until the capacity of the resources precludes scheduling more activities; when this happens one must repeat the process in the next t period with a nonempty set of schedulable activities. The parallel SGS seems to be one of the most popular constructive heuristics for the RCPSP [10].

According to the number of times the heuristic is executed, and hence the number of generated schedules, one can distinguish *single-* and *multi-pass* approaches. In single-pass heuristics, only one priority rule is used to select the activities to be scheduled. Multi-pass versions aim to improve the results by executing the procedure using different priority rules and keeping the best solution [39]. Priority-rule-based heuristics are very useful since they allow to have initial solutions that can be used later for starting most complex meta-heuristics. They can also be part of other meta-heuristics (like in the heuristic methods proposed in Chapter 6). Some applications of priority-based heuristics for the RCPSP can be found in [10, 39, 42].

Over the last few years, an important number of hybrid metaheuristics, combining exact methods with heuristics, for the RCPSP have been proposed [135]. Yoosefzadeh and Tareghian [164], for example, proposed a hybrid method based on Branch-and-Bound coupled with a genetic algorithm (the GA allows to calculate better upper bounds at the nodes of the tree search). Morillo-Torres *et al.* [121] also proposed a hybrid method

based on Branch-and-Bound, but this time the authors use four heuristic methods (with dominance rules) to guide the search process. A search algorithm with subproblem exact resolution, based on the large neighbourhood search, was proposed by Palpant *et al.* [132]. At each iteration of the algorithm, a subpart of a current solution is fixed while the rest defines a subproblem that is solved by an external method (e.g. CP or MILP). The solution of the subproblem is included in the main solution, and a heuristic method is used to update the current solution. The authors tested different ways to choose the subproblem, and the size of the problem is self-adaptive (to limit the time required for the exact method). Their work inspired the Large Neighbourhood Search algorithm proposed in Section 6.5 for the MSPSP-PP.

3.2 Multi-Skill Project Scheduling Problem (MSPSP)

One of the assumptions of the RCPSP is that each resource has a specific function, or in other words, the resources are mono-skilled [14]. This hypothesis can quickly become false when we are also interested in the allocation of human resources working in the project. In real life, a resource could perform several functions leading us to a *Multi-Skill Project Scheduling Problem (MSPSP)*. In our research project, for example, to model human resources as multi-skilled resources is essential. This is because, in the nuclear environment, the execution of the activities requires authorisations and training that each technician may or not have.

The MSPSP, presented by the first time in Néron [127], combines characteristics of both the classical RCPSP for the project description, and the Multi-Purpose Machine model with the addition of new resource constraints. In this variant a resource is therefore characterised by the set of skills it possesses; a task is now defined by the number of required resources with a specific skill. The MSPSP consists in determining a feasible schedule, respecting the precedence constraints between activities and the resource constraints: a resource cannot execute a skill it does not master, cannot be assigned to more than one competence requirement at a given time, and must be assigned to the corresponding activity during its whole processing time. The aim is to minimise the total duration of the project [9, 27]. The MSPSP can be seen as a Multi-mode RCPSP where each execution mode is defined for a valid resource allocation. However, most of the time, the number of resulting modes can be prohibitive for using the classical algorithms used for solving the multi-mode RCPSP [26]. Figure 3.2 summarises the characteristics of the MSPSP.

The MSPSP, initially proposed to schedule IT development projects [127], acquires great importance for scheduling activities in particular fields, such as pharmaceutical, chemical and nuclear, where the regulation requires the presence of a group of technicians having a set of well-defined skills for the execution of an activity [87]. This problem shows to be more challenging than traditional RCPSP, due to the extra decision we need to make: we need to decide not only which resources will be

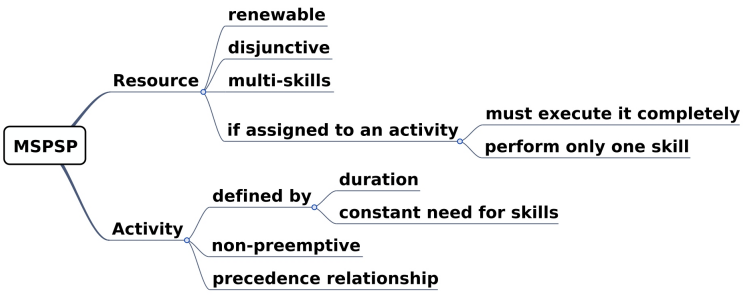

Figure 3.2: Characteristics of the MSPSP

assigned to each activity, but also the skills with which they will contribute [48]. As indicated by Bellenguez-Morineau [26] in her thesis, for each instance of the RCPSP we can match an instance of the MSPSP with resources mastering only one skill; what means the MSPSP extend the RCPSP. Since the RCPSP has been proved to be strongly NP-hard [33], we can, therefore, infer that the MSPSP is also strongly NP-hard.

In the same way as for the RCPSP, it is common to find variants of the MSPSP in the scientific literature. Javanmard *et al.* [87], for example, present a preemptive multi-skilled resource investment project scheduling where the objective is to minimise the total recruitment cost of the project. Maghsoudlou *et al.* [114] also present a preemptive MSPSP, but this time the authors include the presence of hard/soft due dates, and the objective if the minimisation of the total cost of staff allocation and the penalties for the earliness/tardiness and preemption. Li and Womer [109] worked over an MSPSP including the concept of minimal and maximal time lags (a minimal (maximal) time lag is defined as the minimal (maximal) time interval between the start or finish of an activity A_i and the start or finish of another activity A_j [163]).

Almeida [8] classifies the resources used in the MSPSP into two categories: *homogeneous* and *heterogeneous*. Problems with homogeneous resources assume that all resources can perform a given skill with the same efficiency level (like in the basic MSPSP proposed by Néron [127]). In the heterogeneous case, the efficiency level at which a skill is performed may vary across the resources. Examples using heterogeneous resources are the ones presented by Bellenguez and Néron [25] and Kia *et al.* [96]. In their problem, each activity required a given amount of each skill with a minimum level of mastering. One must assign, for each skill need, a resource who masters at least the required level of the skill during the whole processing time of the activity; this while minimising the makespan. The previous problem assumes that the mastering levels are constant over time. However, in real life, it may be interesting to include the possible evolution of mastering levels (improvement due to learning or worsening due to

forgetfulness). Problems dealing with this issue are presented by Chen *et al.* [43] and Attia *et al.* [19].

Various MILP formulations, continuous and discrete-time based, have been proposed in the literature for the MSPSP. For the continuous-time based models, most of the time authors have used precedence-based variables. Correia *et al.* [47] proposed a continuous-time based MILP for the standard version of the MSPSP, using precedence-based variables. They also propose various additional inequalities aiming to enhance the model. A precedence-based non-linear model for the MSPSP is presented Kazemipoor *et al.* [92]. The authors show later how the model can be linearized through a series of variable conversions. A MILP formulation, having some similitude with the model proposed by Correia *et al.* [47], was proposed before by Li and Womer [109] for the MSPSP with minimal and maximal time lags. Discrete-time based formulations have been proposed by Bellenguez-Morineau [26], Montoya *et al.* [120] (Correia and Saldanha-da-Gama [49] proposed minor corrections to this model) and Almeida *et al.* [11]. All of them using binary time-indexed step variables.

Almeida *et al.* [11] did a theoretical and empirical comparison of the models proposed by Correia *et al.* [47], Montoya *et al.* [120] and two new formulations proposed in the article. They proved that the continuous-time based model of Correia *et al.* [47] has theoretically the worst linear relaxation. However, for the computational experiments, this model performed the best in terms of the number of optimal solutions and the time required to prove their optimality. What proves again the fact a better linear relaxation does not necessarily mean a better performance in practice. They also proved that the disaggregated versions of the three discrete models have equal linear relaxation and that they are never worst than their aggregated version. A conclusion similar to the one done by Artigues [15] for the time-indexed formulations of the RCPSP.

MILP formulations for the preemptive MSPSP are very scant in the scientific literature. Moreover, the few ones we could find use time-indexed on/off variables. Maghsoudlou *et al.* [114] proposed a MILP formulation for the preemptive MSPSP with due dates. They used an On/Off variable $Z_{i,k,t}$, taking the value of 1 if the part k of the activity i is executed at time t, together with other on/off variable ($X_{i,m,k,t}$) indicating the periods t at which each technician m works over each part k of the activity i. An allocation binary variable ($Y_{i,m,k,s}$) is also used to indicate the skill s that every technician m performs over each part k of each activity i. Continuous variables are used to determine the earliness, tardiness and completion time of each activity. Dhib *et al.* [58] also formulated a MILP for a preemptive variant of the MSPSP where all parts of one activity, each corresponding to one skill requirement, must start simultaneously but can be preempted at different times. The authors used an on/off variable $X_{i,k,l,t}$ taking the value of 1 if resource k performs the skill l for activity i during time t. Continuous auxiliary variables are used to calculate the starting and completion time of activities and the completion time of each skill for every activity.

Young *et al.* [165] used pure Constraint Programming to solve the MSPSP. In their work, the authors proposed and tested different configurations of CP models, together with different search and propagation techniques. Using the best of their configurations, the authors were able to close an important number of open instances of the literature, showing a better performance than the Branch-and-Price algorithm proposed by Montoya *et al.* [120], thus proving the interest of using CP for solving the MSPSP. Li and Womer [109] presented a hybrid MILP/CP benders decomposition algorithm (with various configurations), where the problem is decomposed into a relaxed master problem having only assignment variables and constraints, and a feasibility subproblem only with the scheduling variables and constraints. CP is used to solve the subproblem to infer cuts that are added into the master problem for excluding infeasible assignments. The authors conclude that their decomposition algorithms are faster than their pure MILP or CP algorithms.

Solving industrial-size instances of the MSPSP is time-consuming due to the complexity of the problem. Heuristic methods are then necessary to tackle this type of instances in small times. Most of the heuristics for the MSPSP in the literature are based on the use of priority rules. In her thesis work, Bellenguez-Morineau [26] presented various greedy algorithms: one based on the Serial Generation Scheme (SGS) and two using the Parallel Generation Scheme (PGS). All of them using priority rules for determining the order in which activities are considered at each iteration of the heuristics. The technician allocation subproblem is modelled as a Minimum Cost Max Flow (MCMF) problem, where the cost of assigning a resource is related to the degree on which unscheduled activities may need the skills it masters. This way to model the allocation subproblem is used in the heuristic method proposed in Chapter 6. Based on these greedy algorithms, Bellenguez-Morineau also proposed a tabu search that gives an average optimality gap of 4% (on instances up to 90 activities). Two genetic algorithms are also presented in her thesis.

Almeida *et al.* [10] also presented a greedy algorithm using the PGS. However, this time, the authors proposed a "multi-pass" version of the algorithm, where different priority lists for activities and the weight (criticality) of resources (for its assignments) are tested and the best solution found is kept. Almeida [8] proposed a biased random-key genetic algorithm that allowed him to get an average optimality gaps of 1.1% for mean computation times lower than one minute.

Myszkowski *et al.* [124] tested different priority rules, the traditional ones and some more complex ones, with an SGS algorithm. They concluded that more complex rules not always lead to better results. Because of their ease of development and the quality of the results obtained, the authors propose using algorithms based on priority rule as an element of more elaborate metaheuristics. Later on, Myszkowski and Siemieński [123] presented a basic GRASP for the MSPSP. At each iteration of the algorithm, a

feasible priority list (sequence) for activities is randomly generated, then a randomised greedy-based algorithm, based on an SGS, is used to process the resource allocation. For the local search phase, a series of swapping moves are done over the initial sequence, and the randomised greedy-based algorithm is used again to assign the resources. The GRASP of Myszkowski and Siemieński [123] does not learn from the results of past iterations. The inclusion of learning from past iterations is indeed one of the aspects we test on the GRASP algorithm proposed in Section 6.4.

In this chapter, we presented the main characteristics and methods for the project scheduling problem. In Part II, we will introduce the methods proposed to handle the scheduling problem at the LECA-STAR.

Part II

Proposed Methods

Chapter 4

MSPSP with penalty for preemption

Contents

4.1 Problem description **52**

4.2 MILP formulations **53**

4.2.1 Model MSWP1 53

4.2.2 Model MSWP2 56

4.2.3 Model MSWP3 57

4.2.4 Model MSWP4 58

4.3 Computational experiments **60**

4.4 Limited theoretical analysis **62**

4.5 Concluding remarks **63**

After studying the scheduling process at the LECA-STAR (Chapter 1), and comparing its characteristics with the scheduling models in Chapter 3, we get a partial match between our scheduling problem and the well-known Multi-Skill Project Scheduling Problem. However, this theoretical problem assumes that activities cannot be preempted. As we saw in Chapter 1, the preemption of a subset of activities is required for modelling the real behaviour at the facility. We must then adapt the classical problem and allows the preemption of these activities. A first attempt for modelling the LECA-STAR scheduling problem is to use a Multi-Skill Project Scheduling Problem with penalty for preemption. This new variant allows the preemption of some activities. However, in order to limit the number of times each activity is preempted, one can apply a penalty every time the activity is stopped.

This chapter is structured as follows: First, we present the problem description in Section 4.1. Once defined the characteristics of our variant, we present, in Section 4.2, four MILP formulations for the MSPSP with penalty for preemption. Computational experiments on the performance of the proposed MILP models are presented in Section 4.3. Finally, Section 4.4 presents a brief theoretical analysis of the strength of the proposed models. A part of the results presented in this chapter has been published in [142].

4.1 Problem description

The characteristics of the scheduling process in the nuclear laboratory, presented in Section 1.3.1, take us to conclude that the facility scheduling problem could be modelled as an extension of the Multi-Skill Project Scheduling Problem (MSPSP). To do so, we must undergo some changes over the baseline version of the MSPSP. This will allow us to get a more accurate representation of the industrial scheduling problem.

The most significant change is related to the possibility of interrupting an activity once it has started. The non-preemption constraints (activities execution must be without interruption) is one of the main characteristics of the MSPSP [127]. In our study, it is necessary to allow the preemption of a set of activities, while still forbidding this preemption for the remainder. Nuclear regulation requires that, for some critical activities, the workspace must be put in a safe configuration whenever an activity is interrupted, which means a loss of time and a decrease of the productivity. We must then try to limit the number of times these activities are preempted. At first, a suitable approach is to use an MSPSP with penalty for preemption where we apply a penalty (M_i) every time an activity i, for which we want to limit the preemption, is preempted. This penalty is then minimised in the objective function. We can classify the activities into three sets: Non-preemptive activities ($\overline{NP}$), preemptive activities ($\overline{P}$) and preemptive activities with penalty ($\overline{PP}$).

The second set of changes is related to the characteristics of the resources we model. The MSPSP, as defined by [127], works only with disjunctive resources, which means they have a unitary capacity. In our industrial variant, we must work with both disjunctive multi-skilled resources (technicians) and cumulative (i.e. capacity higher than 1 activity at a time) mono-skilled resources (compound machines and buildings). Unlike what happens in the classical MSPSP, our technicians can execute several skills at a time per activity. However, we must ensure that a minimal number of technicians is allocated to comply with safety and operational constraints. Additionally, since some activities have a duration larger than technicians work shifts, the technicians can partially execute the activities, except for non-preemptive activities ($i \in \overline{NP}$), which duration is smaller than work shifts. Finally, to represent the absences/presences of technicians over time, we must work with time-varying resource capacity.

Traditional MSPSP does not take into account time windows for scheduling the activities. As stated in Section 1.3.1, some of the activities may be subject to time windows. These activities must be scheduled after a fixed release date (r_i) and/or before a deadline (dl_i). All other characteristics are similar to the classical MSPSP. Once defined the characteristics of the problem, we can now begin the description of its mathematical formulation.

4.2 MILP formulations

In this section, we present four discrete-time based MILP models for the MSPSP with penalty for preemption. The first three models are original and inspired by the time-indexed formulations for the (preemptive) RCPSP (see Chapter 3); the fourth one is an adaptation of the models proposed by Afshar-Nadjafi [4] for the RCPSP with penalty for preemption, and by Maghsoudlou *et al.* [114] for the preemptive MSPSP. A first version of these formulations was presented in [142]. The parameters used for the models are shown in Table 4.1.

4.2.1 Model MSWP1

4.2.1.1 Decision variables

The first model uses two types of time-indexed binary variables (Section 3.1.2), together with some auxiliary variables :

- *on/off variables* to indicate the periods of execution of each activity and the technicians allocation;
- *pulse variables* to indicate when each part of an activity starts.

Activity executions: These binary variables indicate the periods over which each activity is executed.

$$Y_{i,t} = \begin{cases} 1 & \text{if activity } i \in \overline{I} \text{ is executed during time } t \in \overline{H} \\ 0 & \text{otherwise} \end{cases}$$

Technician allocations: This second set of binary variables indicate which technicians are allocated to each activity at each time.

$$O_{j,i,t} = \begin{cases} 1 & \text{if technician } j \in \overline{J} \text{ is allocated to activity } i \in \overline{I} \text{ during time } t \in \overline{H} \\ 0 & \text{otherwise} \end{cases}$$

Auxiliary allocation variables: These auxiliary binary variables are used to ensure that all technicians allocated to a non-preemptive activity must execute it until its completeness. They exist only for non-preemptive activities.

$$S_{j,i} = \begin{cases} 1 & \text{if technician } j \in \overline{J} \text{ is assigned to execute activity } i \in \overline{NP} \\ 0 & \text{otherwise} \end{cases}$$

Notations for parameters	
$\overline{I}$	Set of activities to be scheduled
$\overline{NP} \subseteq \overline{I}$	Set of non-preemptive activities
$\overline{P} \subseteq \overline{I}$	Set of preemptive activities
$\overline{PP} \subseteq \overline{I}$	Set of preemptive activities with penalty
$\overline{E}$	Set of precedence relationships. $(i, l) \in \overline{E}$ states that activity $l \in \overline{I}$ cannot start before activity $i \in \overline{I}$ is completed
$\overline{K}$	Set of mono-skilled resources
$\overline{C}$	Set of skills
$\overline{J}$	Set of available technicians
T	Last period of the scheduling horizon
$\overline{H} = \{1, 2, ..., T\}$	Set of scheduling periods
$DR_{k,t} \in \mathbb{Z}_+$	Amount of resource $k \in \overline{K}$ available at time $t \in \overline{H}$
$Br_{i,k} \in \mathbb{Z}_+$	Amount of resource $k \in \overline{K}$ required for executing activity $i \in \overline{I}$
$DO_{j,t} \in \{0, 1\}$	Presence/absence of technician $j \in \overline{J}$ at time $t \in \overline{H}$. Equal to 1 if present, 0 otherwise
$CO_{j,c} \in \{0, 1\}$	Indicates whether a technician $j \in \overline{J}$ masters skill $c \in \overline{C}$ or not. Equal to 1 if mastered, 0 otherwise
$Bc_{i,c} \in \mathbb{Z}_+$	Amount of technicians mastering the skill $c \in \overline{C}$ required for executing activity $i \in \overline{I}$
$Nt_i \in \mathbb{Z}_+$	Minimal number of technicians required to execute activity $i \in \overline{I}$
$D_i \in \mathbb{Z}_+$	Duration of activity $i \in \overline{I}$
$r_i \in \overline{H}$	Release date for activity $i \in \overline{I}$
$dl_i \in \overline{H}$	Deadline for activity $i \in \overline{I}$
$M_i \in \mathbb{Z}_+$	Penalty for preempting activity $i \in \overline{PP}$ (expressed in time units)

Table 4.1: Parameters for the MSPSP with penalty for preemption

Start of each part of the activities: These binary variables allow to know the periods where an activity is started or resumed after preemption. Note that this variable is not necessary for preemptive activities without penalty ($i \in \overline{P}$).

$$X_{i,t} = \begin{cases} 1 & \text{if a part of activity } i \notin \overline{P} \text{ starts at time } t \in \overline{H} \\ 0 & \text{otherwise} \end{cases}$$

Project makespan: Indicates the total duration of the project.

$C_{\max} \in \mathbb{R}_+$: project makespan

4.2.1.2 Formulation

Objective function: To ensure the normal progress of research projects, we must be sure that the set of weekly activities is executed in the shortest possible time. This can be translated as the minimisation of the project makespan:

$$\min(C_{\max})$$

We must also ensure that the penalties due to the preemption (M_i) of those activities for which we want to limit the preemption ($i \in \overline{PP}$) are minimised. The value of M_i can be seen as the minimal improvement over the $C_{\max}$ expected after allowing the preemption of activity i. The minimisation of penalties can be modelled as:

$$min\left(\sum_{i\in\overline{PP}}\left(M_i * \left(\left(\sum_{t=max(1,r_i)}^{\min(dl_i,T)} X_{i,t}\right) - 1\right)\right)\right)$$

Even if an activity i is not preempted, the constraints of the problem that stipulate that an activity has to be started (see below) yield $\sum_{t\in\overline{H}} X_{i,t} = 1$. That is why we must subtract 1 before multiplying by the penalty M_i. The formulation is consequently defined as follows:

$$\min\left(C_{\max} + \sum_{i\in\overline{PP}}\left(M_i * \left(\left(\sum_{t=max(1,r_i)}^{\min(dl_i,T)} X_{i,t}\right) - 1\right)\right)\right) \tag{4.1}$$

Subject to:

$$\sum_{i\in\overline{I}}(Y_{i,t} * Br_{i,k}) \leq DR_{k,t} \qquad \forall t \in \overline{H}, \forall k \in \overline{K} \tag{4.2}$$

$$\sum_{i\in\overline{I}} O_{j,i,t} \leq DO_{j,t} \qquad \forall j \in \overline{J}, \forall t \in \overline{H} \tag{4.3}$$

$$Y_{i,t} * Bc_{i,c} \leq \sum_{j \in \overline{J}} (O_{j,i,t} * CO_{j,c}) \quad \forall i \in \overline{I}, \forall t \in \overline{H}, \forall c \in \overline{C} \tag{4.4}$$

$$\sum_{j \in \overline{J}} O_{j,i,t} \geq Y_{i,t} * Nt_i \quad \forall t \in \overline{H}, \forall i \in \overline{I} \tag{4.5}$$

$$\sum_{t=max(1,r_i)}^{\min(dl_i,T)} Y_{i,t} = D_i \quad \forall i \in \overline{I} \tag{4.6}$$

$$D_i * (1 - Y_{l,t}) \geq \sum_{t'=t}^{T} Y_{i,t'} \quad \forall (i,l) \in \overline{E}, \forall t \in \overline{H} \tag{4.7}$$

$$X_{i,t} \geq Y_{i,t} - Y_{i,t-1} \quad \forall i \notin \overline{P}, \forall t \geq 2 \tag{4.8}$$

$$X_{i,1} = Y_{i,1} \quad \forall i \notin \overline{P} \tag{4.9}$$

$$\sum_{t=1}^{T} X_{i,t} = 1 \quad \forall i \in \overline{NP} \tag{4.10}$$

$$O_{j,i,t} \geq S_{j,i} + Y_{i,t} - 1 \quad \forall j \in \overline{J}, \forall t \in \overline{H}, \forall i \in \overline{NP} \tag{4.11}$$

$$O_{j,i,t} \leq S_{j,i} \quad \forall j \in \overline{J}, \forall t \in \overline{H}, \forall i \in \overline{NP} \tag{4.12}$$

$$O_{j,i,t} \leq Y_{i,t} \quad \forall j \in \overline{J}, \forall t \in \overline{H}, \forall i \in \overline{NP} \tag{4.13}$$

$$C_{\max} \geq t * Y_{i,t} \quad \forall i \in \overline{I}, \forall t \in \overline{H} \tag{4.14}$$

Constraints (4.2) guarantee that the total demand of each resource during all periods $t \in \overline{H}$ is always lower than the resource capacity. Constraints (4.3) ensure that technicians are allocated in period t only if they are available. They also ensure that the technicians can be allocated to at most one activity during period t (since $DO_{j,t}$ is a Boolean). Skills requirements and minimal number of technicians are ensured by constraints (4.4) and (4.5) respectively. The respect of the activities duration is ensured by constraints (4.6); these constraints also limit the activity executions between their release dates (r_i) and deadlines (dl_i) when there exist. Constraints (4.7) ensure that activity l does not start before activity i is finished (precedence relationships). The start time of each part of an activity is determined by constraints (4.8) and (4.9). With inequalities (4.10) we ensure that preemption is not allowed for non-preemptive activities ($i \in \overline{NP}$). Constraints (4.11), (4.12) and (4.13) ensure that if a technician is allocated to a non-preemptive activity it must execute it until completeness. Finally, the makespan of the project is calculated using constraints (4.14).

4.2.2 Model MSWP2

4.2.2.1 Decision variables

This model is similar to Model MSWP1. The difference is that now, additionally to the variables used for Model MSWP1, we use continuous variables for determining the start and finish time of activities. These new variables will allow us to write the constraints

for precedence relationships and non-preemption differently. Moreover, binary variables indicating the start of each part of an activity ($X_{i,t}$) will only exist for preemptive activities with penalty ($i \in \overline{PP}$).

Start time of activities: This set of continuous variables gives the start time of activity $i \in \overline{I}$.

$G_i \in \mathbb{R}_+$: start time of activity $i \in \overline{I}$

Finish time of activities: This second set of continuous variables indicate the finish time of activity $i \in \overline{I}$.

$F_i \in \mathbb{R}_+$: finish time of activity $i \in \overline{I}$

4.2.2.2 Formulation

Objective function: We keep using the objective function (4.1)

Subject to: Constraints (4.2–4.6), (4.8), (4.9) and (4.11–4.14) remain unchanged. Constraints (4.7), stating the precedence relationships, are changed by:

$$F_i + 1 \leq G_l \qquad \forall (i,l) \in \overline{E} \tag{4.15}$$

The non-preemption constraints for activities within $\overline{NP}$ are now defined as follows:

$$F_i - G_i + 1 = D_i \qquad \forall i \in \overline{NP} \tag{4.16}$$

Additional constraints must be added to determine the start and finish time of the activities:

$$G_i \leq t * Y_{i,t} + (1 - Y_{i,t}) * T \qquad \forall i \in \overline{I}, \forall t \in \overline{H} \tag{4.17}$$

$$F_i \geq t * Y_{i,t} \qquad \forall i \in \overline{I}, \forall t \in \overline{H} \tag{4.18}$$

Note that the G_i can start way before the start of the first part of the activity. It is clear that it is not really a problem for the correctness of the formulation, but could the formulation be made more efficient by adding additional constraints to enforce this tightness.

4.2.3 Model MSWP3

4.2.3.1 Decision variables

This model also uses Model MSWP1 as a basis. This time, we use additional binary variables to know the finish time of activities. The new variables allow us to reformulate

the constraints related to the precedence relationship.

Finish time of activities: The binary variables $F_{i,t}$ will be equal to 1 for all periods t lower than or equal to the finish time of activity $i \in \overline{I}$, they will be 0 for the other periods. The sum on $t \in \overline{H}$ of these variables indicate the finish time of each activity $i \in \overline{I}$.

$$F_{i,t} = \begin{cases} 1 & \text{if activity } i \in \overline{I} \text{ finishes at period } t \in \overline{H} \text{ or after} \\ 0 & \text{otherwise} \end{cases}$$

4.2.3.2 Formulation

Objective function: We use the same objective function as for Model MSWP1: (4.1)

Subject to: We keep most of the constraints of Model 1. We substitute the precedence relationship constraints (4.7) with the new constraints (4.19).

$$Y_{l,t} \leq (1 - F_{i,t}) \qquad \forall (i,l) \in \overline{E}, \forall t \in \overline{H} \tag{4.19}$$

We must add constraints for getting the values of the $F_{i,t}$ variables:

$$F_{i,t} \geq Y_{i,t'} \qquad \forall t \in \overline{H}, \forall t' \geq t \tag{4.20}$$

4.2.4 Model MSWP4

4.2.4.1 Decision variables

Activities execution periods: This binary variable indicate the periods on which each unit of duration $v \in \overline{V_i}$ ($\overline{V_i} = \{1, .., D_i\}$) of activity $i \in \overline{I}$ is executed. In other words, each activity $i \in \overline{I}$ is divided into D_i parts with unitary duration.

$$Y_{i,v,t} = \begin{cases} 1 & \text{if } v\text{-th unit of duration of activity } i \in \overline{I} \text{ is executed at time } t \in \overline{H} \\ 0 & \text{otherwise} \end{cases}$$

Technician allocations: This binary variable indicates the technicians allocation over time.

$$O_{j,i,t} = \begin{cases} 1 & \text{if technician } j \in \overline{J} \text{ is allocated to activity } i \in \overline{I} \text{ during time } t \in \overline{H} \\ 0 & \text{otherwise} \end{cases}$$

Auxiliary allocation variables: This auxiliary variable has the same function than in Model 1.

$$S_{j,i} = \begin{cases} 1 & \text{if technician } j \in \overline{J} \text{ is assigned to execute activity } i \in \overline{NP} \\ 0 & \text{otherwise} \end{cases}$$

Activities preemption: These variables indicate whether an activity is preempted or not.

$$X_{i,v,t} = \begin{cases} 1 & \text{if the } v\text{-th unit of duration of activity } i \in \overline{PP} \text{ preempts at time } t \in \overline{H} \\ 0 & \text{otherwise} \end{cases}$$

Project makespan: This continuous variable gives the makespan of the project.

$C_{\max} \in \mathbb{R}_+$: project makespan

4.2.4.2 Formulation

Objective function: The minimisation of the $C_{\max}$ and the penalties for preemption can be formulated as:

$$\min \left(C_{\max} + \sum_{i \in \overline{PP}} \sum_{v \in \overline{V_i}} \sum_{t \in \overline{H}} M_i * X_{i,v,t} \right) \tag{4.21}$$

Subject to:

$$\sum_{i \in \overline{I}} O_{j,i,t} \leq DO_{j,t} \qquad \forall j \in \overline{J}, \forall t \in \overline{H} \tag{4.22}$$

$$\sum_{i \in \overline{I}} \sum_{v \in \overline{V_i}} (Y_{i,v,t} * Br_{i,k}) \leq DR_{k,t} \qquad \forall t \in \overline{H}, \forall k \in \overline{K} \tag{4.23}$$

$$Y_{i,v,t} * Bc_{i,c} \leq \sum_{j \in \overline{J}} (O_{j,i,t} * CO_{j,c}) \qquad \forall i \in \overline{I}, \forall v \in \overline{V_i}, \forall t \in \overline{H}, \forall c \in \overline{C} \tag{4.24}$$

$$\sum_{j \in \overline{J}} O_{j,i,t} \geq Y_{i,v,t} * Nt_i \qquad \forall t \in \overline{H}, \forall i \in \overline{I}, \forall v \in \overline{V_i} \tag{4.25}$$

$$\sum_{t=max(1,r_i)}^{\min(dl_i,T)} Y_{i,v,t} = 1 \qquad \forall i \in \overline{I}, \forall v \in \overline{V_i} \tag{4.26}$$

$$\sum_{t \in \overline{H}} t * Y_{i,D_i,t} + 1 \leq \sum_{t \in \overline{H}} t * Y_{l,1,t} \qquad \forall (i,l) \in \overline{E} \tag{4.27}$$

$$Y_{i,v,t} \leq Y_{i,v+1,t+1} - X_{i,v,t} \qquad \forall i \in \overline{PP}, \forall v \in \overline{V_i}, \forall t \in \overline{H} \tag{4.28}$$

$$Y_{i,v,t} \leq Y_{i,v+1,t+1} \qquad \forall i \in \overline{NP}, \forall v \in \overline{V_i}, \forall t \in \overline{H} \tag{4.29}$$

$$\sum_{t \in \overline{H}} t * Y_{i,v-1,t} + 1 \leq \sum_{t \in \overline{H}} t * Y_{i,v,t} \qquad \forall i \in \overline{I}, \forall v \in \overline{V_i} \tag{4.30}$$

$$O_{j,i,t} \geq S_{j,i} + Y_{i,v,t} - 1 \qquad \forall j \in \overline{J}, \forall i \in \overline{NP}, \forall v \in \overline{V_i}, \forall t \in \overline{H} \tag{4.31}$$

$$O_{j,i,t} \leq S_{j,i} \qquad \forall j \in \overline{J}, \forall t \in \overline{H}, \forall i \in \overline{NP} \tag{4.32}$$

$$C_{\max} \geq t * Y_{i,D_i,t} \qquad \forall i \in \overline{I}, \forall t \in \overline{H} \tag{4.33}$$

Constraints (4.22) ensure that technicians are allocated to at most one activity in period t and only if they are available. The demand of resources are guaranteed by constraints (4.23). Skills requirements and minimal number of technicians are ensured by constraints (4.24) and (4.25) respectively. Constraints (4.26) specify that only execution time is permitted for every unit of duration of an activity. These constraints also guarantee that the activity is executed between their release date and deadline. Constraints (4.27) represent the precedence relationships. Constraints (4.28) guarantee that, if two successive units of duration of an activity $i \in \overline{PP}$ (i.e., units v and $v+1$) are interrupted at time t the corresponding decision variable $X_{i,v,t}$ must be set to 1. Constraints (4.29) state that, for non-preemptive activities, the execution time of all parts of the activity must be contiguous (non-preemption constraints). Constraints (4.30) specify that the execution time for each unit of duration of an activity has to be at least one time unit later than the finish time for the previous one. Constraints (4.31) together with constraints (4.32) ensure that if a technician is allocated to a non-preemptive activity it must execute it until completeness. Finally, the makespan of the project is calculated using constraints (4.33).

4.3 Computational experiments

For computational tests, we use CPLEX 12.7 on a computer equipped with an Intel Xeon E5-2695 processor at 2.3 GHz. The solver was set in its default configuration, limiting the number of threads at 8, and computation time limited to 15 min. We generated sets of instances using a random generation algorithm that allows fixing aspects such as proportions of preemption type, percentage of activities with time windows, density of precedence relationships, skill number per technician, etc. To test the behaviour of our models regarding the proportion of each activity type (preemptive, preemptive with penalty, and non-preemptive) in an instance, we generated four sets (A, B, C and D) of 50 instances. Table 4.2 presents the specific distribution of the preemption type for each set. All instances have 30 activities with a duration between 5 to 10 time units, up to 15 skills, 8 cumulative resources, 8 technicians (multi-skilled resources), 20% of activities with time windows, the penalty for the activities within $\overline{PP}$ (M_i) have been set to 1. The remaining characteristics are randomly generated.

Table 4.3 shows the number of feasible solutions (by preemption type) that each

	Set A	Set B	Set C	Set D
Non-preemptive	10%	10%	80%	33.3%
Preemptive with penalty	10%	80%	10%	33.3%
Preemptive	80%	10%	10%	33.3%

Table 4.2: Distribution of preemption types per set of instances

model was able to found within the time limit. It also shows the number of instances for which the models proved the optimality. Model MSWP4 gets the worst results being able only to find feasible solutions for 45 out of 200 instances. Model MSWP2 seems to have a poor performance when the percentage of non-preemptive activities is high (Set C). Models MSWP1 and MSWP3 get the best results obtaining a higher number of feasible solutions for all instance types. A high density of non-preemptive activities seems to also have a negative impact on the performance of Models MSWP1 and MSWP3. In general, Models MSWP1, MSWP2 and MSWP3 perform better when the proportion of preemptive activities (without penalty) is high, since the optimality was proved only for instances of set A.

	Model MSWP1 Feasible solutions (Solved to opt.)	Model MSWP2 Feasible solutions (Solved to opt.)	Model MSWP3 Feasible solutions (Solved to opt.)	Model MSWP4 Feasible solutions (Solved to opt.)
Set A	50 (10)	50 (9)	50 (12)	1 (0)
Set B	50 (0)	50 (0)	50 (0)	29 (0)
Set C	42 (0)	9 (0)	43 (0)	0 (0)
Set D	50 (0)	45 (0)	50 (0)	15 (0)
All	192 (10)	154 (9)	193 (12)	45 (0)

Table 4.3: Feasible solutions after 15 minutes

A more in-depth analysis of results for Models MSWP1 and MSWP3 suggests that the two models have the same performance in terms of the objective function value after 15 minutes. To test this hypothesis, we use the *Wilcoxon Signed-Ranks Test* [112] (a non-parametric test designed to evaluate the difference between the mean of two correlated samples). The null hypothesis of this test asserts that the medians of the two samples are identical. For all *p-val* greater than 0.05 one can conclude that there is not enough statistical evidence to reject the null hypothesis. From the results presented in Table 4.4, we can conclude that there is not statistical difference between the performance of the two models regardless of the distribution of preemption type within the instances. A similar test was done for comparing the values of the lower bounds obtained by CPLEX after the time limit for these two models. We found enough statistical evidence to conclude that Model MSWP1 gives better lower bounds. However, the quality of the lower bound obtained by CPLEX is not good enough, generating gaps between the best-found solution and the lower bound of more than 90%.

	p-val MSWP1 = MSWP3	Conclusion
Set A	0.4417	Same performance
Set B	0.3616	
Set C	0.063	
Set D	0.856	
All	0.4093	

Table 4.4: Wilcoxon signed-ranks test

4.4 Limited theoretical analysis

One can notice that models MSWP1, MSWP2 and MSWP3 all use variables $O_{i,j,t}$ and $Y_{i,t}$ for representing the solution, and their only difference is to be found in the way we represent the precedence constraints, all of them being equivalent in $\mathbb{Z}$.

We could try to analytically prove if one of the formulations is tighter than the others in the sense that the constraints of the tighter formulation imply the ones of the other formulation in the continuous domains, while the reverse would not hold (see Section 2.2.1). From constraints (4.19) and (4.20) used in Model MSWP3 for indicating the precedence relationships, for example, one can derive the following expression:

$$Y_{i,t'} \leq (1 - Y_{l,t}) \qquad \forall (i,l) \in \overline{E}, \forall t \in \overline{H}, \forall t' \geq t \tag{4.34}$$

Doing a sum over t' in both sides of constraint (4.34) we get:

$$\begin{aligned} \sum_{t'=t}^{T} Y_{i,t'} &\leq \sum_{t'=t}^{T} (1 - Y_{l,t}) \qquad \forall t \in \overline{H} \\ &\leq (1 - Y_{l,t}) * (T - t + 1) \qquad \forall t \in \overline{H} \end{aligned} \tag{4.35}$$

One can see that the resulting constraints (4.35) have a similar structure than constraints (4.7), used in Model MSWP1 to represent the precedence relationships. However, we can not infer whether MSWP1 (Constraints 4.7) is tighter than Model MSWP3 (Constraints 4.35) or vice versa. For $t > T + 1 - D_i$ constraints 4.35 is tighter. If $t < T + 1 - D_i$ then constraints (4.7) is the tightest one.

Theorem 1. *Formulation MSWP3 is at least as tight as Formulation MSWP2.*

Proof. In the same way, from constraints (4.15),(4.17) and (4.18), used in Model MSWP2, we can infer the following expression:

$$t * Y_{i,t'} \leq t * Y_{l,t} + (1 - Y_{l,t}) * T - 1 \qquad \forall (i,l) \in \overline{E}, \forall t \in \overline{H}, \forall t' \geq t \tag{4.36}$$

One can multiply constraints (4.34), derived from Model MSWP3, by t:

$$t * Y_{i,t'} \leq t * (1 - Y_{l,t}) \qquad \forall (i,l) \in \overline{E}, \forall t \in \overline{H}, \forall t' \geq t \tag{4.37}$$

Constraints (4.36) and (4.37) have a similar structure. Now if we had for all $Y_{l,t} \in [0,1]$ that

$$t * (1 - Y_{l,t}) \leq t * Y_{l,t} + (1 - Y_{l,t}) * T - 1, \quad \forall t \in \overline{H}, \forall t' \geq t$$

we could conclude that Model MSWP3 is tighter than Model MSWP2. The latter statement is equivalent to:

$$(2t - T)Y_{l,t} + T - 1 - t \geq 0, \quad \forall t \in \overline{H}, \forall t' \geq t$$

The different cases for the left-hand side (denoted by expression A) are enumerated as follows. If $t = \frac{T}{2}$, $A = \frac{T}{2} - 1$, the expression hold if $T \geq 2$. $t > \frac{T}{2}$, then $A \geq 0$ holds for all $Y_{l,t} \in [0,1]$ if and only if it holds for $Y_{l,t} = 0$ and $T - 1 - t \geq 0$, i.e. $t \leq T - 1$. Note that the case $t = T$ need not be considered since (i,l) is a precedence constraint we must have $Y_{i,T} = 0$. If $t < \frac{T}{2}$, then $A \geq 0$ holds for all $Y_{l,t} \in [0,1]$ if and only if it holds for $Y_{l,t} = 1$, which gives $t - 1 \geq 0$. □

We tested the quality of the lower bound obtained by releasing the integrity constraints of models MSWP1, MSWP2 and MSWP3, using the same set of instances presented in Section 4.3. Computational tests showed that Model MSWP1 gives always better (or equal in the worst case) lower bounds than models MSWP2 and MSWP3. We also observed that Model MSWP3 always gave better or equal lower bounds than the ones obtained by Model MSWP2. This shows with Theorem 1 that Model MSWP3 is strictly tighter than Model MSWP2. About Model MSWP4, the experiments show that the lower bound is often dominated by our formulations. However, we conjecture about a disaggregated variant of the precedence constraints for Model MSWP4, which could give a stronger lower bound. We did not explore this direction, as this would, even more, enlarge the model size, which is already impracticable. Beyond this result, a deeper theoretical analysis should be done as future research to better understand the theoretical strength of the proposed models.

4.5 Concluding remarks

In this chapter, we described a new variant of the Multi-Skill Project Scheduling Problem. In order to better represent the reality of the laboratory, we allow the preemption of some activities. However, we can apply a penalty every time that an activity is preempted to reduce the number of time each activity is stopped.

We presented four different MILP formulations of the problem, three of them are original and inspired by the time-indexed formulations for the (preemptive) RCPSP,

and the fourth one is an adaptation of a model proposed in the literature for the preemptive MSPSP. A quick theoretical analysis of the strength of the formulations allowed identifying that our Model MSWP3 is tighter than our Model MSWP2. A more extensive analysis of the theoretical strength of the proposed formulations needs to be done as future research.

Computational experiments on a set of 200 instances showed that our proposed models could find more initial solutions (even some time prove the optimality) than the adapted version of the preemptive MSPSP found in the literature. Regarding our original formulations, models MSWP1 and MSWP3 outperform the results of Model MSWP2. A statistical test showed that the results obtained by models MSWP1 and MSWP3 are statistically equal. An analysis over the lower bounds, after a limited computation time, suggested that Model MSWP1 gave better lower bounds.

Even if the proposed variant of the MSPSP allows to model an important number of the activities carried out at the LECA-STAR, it does not fulfil all the safety constraints for a subset of activities. That is why we propose in the following chapter a more accurate problem that takes into account these additional safety constraints.

Chapter 5

MSPSP with Partial Preemption

Contents

5.1 Problem description **65**
5.2 MILP formulations **68**
5.2.1 Model MSPP1 68
5.2.2 Model MSPP2 71
5.2.3 Theoretical comparison 73
5.3 CP formulation **74**
5.4 Computational Experiments **78**
5.4.1 Testing MILP formulations 78
5.4.2 Testing CP formulation 81
5.5 Concluding remarks **82**

Typically, preemptive scheduling problems assume that all resources are released during preemption periods, and that they can be used to perform other activities. Nevertheless, at the LECA-STAR, safety constraints requires that a subset of resources remains allocated to the activity when it has been preempted. The possibility of only releasing a subset of resources during the preemption periods, what we call *partial preemption*, has not been studied yet in the scientific literature. To comply with this safety requirement, and to fill the gap in the literature, we present in this chapter a new variant of the MSPSP that uses the concept of partial preemption.

We describe in Section 5.1 the main characteristics of the new scheduling problem. Then, we formalise the problem by presenting various Mixed Integer/Linear Programming models, and their theoretical comparison in Section 5.2. Later, a Constraint Programming model is presented Section 5.3. We test the computational performance of the different MILP and CP formulations in Section 5.4. Early version of the models described in this chapter have been published in [143] and [144].

5.1 Problem description

Suppose one must execute an experimental activity that requires an inert atmosphere for its execution. In practice, one can stop (preempt) this activity and allow the technicians

and some of the equipment to be used in other activities. However, safety and operational constraints force us to preserve the inert atmosphere even when the activity is stopped (before its end). In other words, one cannot release the equipment that ensures the inert atmosphere during the preemption periods. Traditional preemptive schedule models cannot represent this behaviour since they assume that all resources are released during the preemption periods. Until now, the only way to model this activity, while respecting safety requirements, was to declare it as "non-preemptive". However, this decision can increase the project makespan, especially in our case-study where the activities may have restrictive time-windows and the availability/capacity of the resources vary over time. Aiming to overcome this inconvenience, we propose a new variant of the MSPSP that better represents the behaviour of our laboratory: the MSPSP with partial preemption (MSPSP-PP).

In the MSPSP-PP, if an activity is preempted, we release only a subset of resources while seizing the remainder (*partial preemption*). We can then classify the activities in three types according to the possibility of releasing the resources during the preemption periods: 1) Non-preemptive activities ($\overline{NP}$), if none of the resources can be released; 2) Partially preemptive activities ($\overline{PP}$), if a subset of resources can be released; and 3) Preemptive activities ($\overline{P}$), if all resources can be set free. In our study case, the partial preemption is only related to mono-skilled resources, and we made the hypothesis that technicians can always be released during preemption periods. This is because, in practice, we are not interested in allocating staff to an activity that is not in progress. All other characteristics are the same as those presented in Section 4.1 (page 52) and are summarised in Figure 5.1.

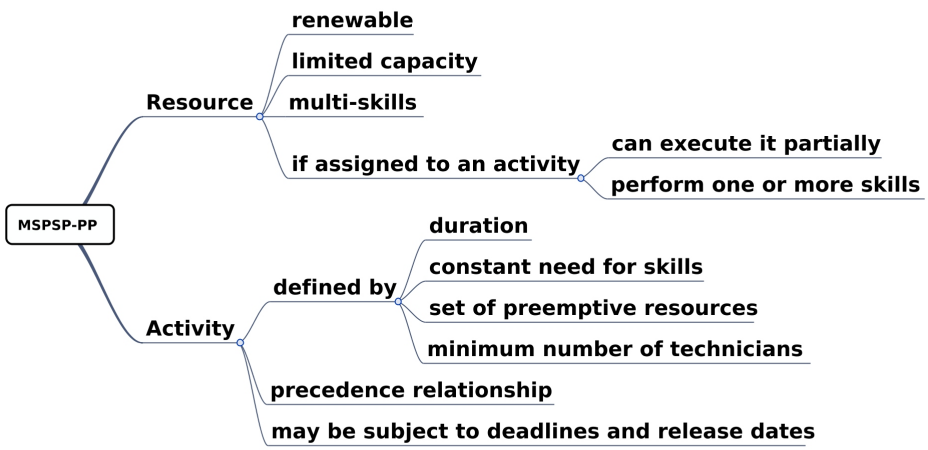

Figure 5.1: Characteristics of the Multi-Skill Project Scheduling Problem with Partial Preemption (MSPSP-PP).

The objective in the MSPSP with partial preemption is to find a feasible schedule that minimises the total duration of the project ($C_{\max}$). Finding a solution consists in determining the periods during which each activity is executed and also which resources will execute the activity in every period; all this, while respecting the resources capacity and the activities characteristics. We must schedule these activities on renewable resources with limited capacity; they can be cumulative mono-skilled resources (machines or equipment) or disjunctive multi-skilled resources (technicians) mastering Nb_j skills. Multi-skilled resources can respond to more than one skill requirement per activity and may execute it partially (except for non-preemptive activities where technicians must perform the whole activity). An activity is defined by its duration (D_i), its precedence relationships, its requirements of resources ($Br_{i,k}$), its requirements of skills ($Bc_{i,c}$), the minimum number of technicians needed to perform it (Nt_i) and the subset of preemptive resources. Activities might or not have either a deadline (dl_i) or a release date (r_i). Figure 5.2 illustrates an example of an MSPSP-PP instance and a possible solution.

	Duration	Skill needed	Machine	Dead line	Release date	Activity type
Act 1	4	(S1,1)	(M1,1)	-	-	Preemptive
Act 2	2	(S3,1),(S4,1)	(M1,1)	5	3	Non-preemptive
Act 3	4	(S2,1)	(M1,1)	-	-	Partially preemp (M1 can not be released)

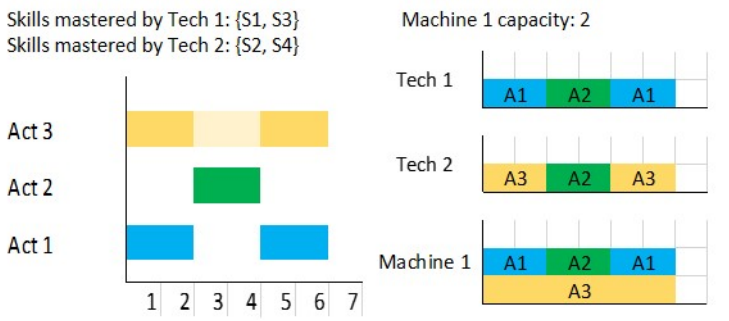

Figure 5.2: Example of an MSPSP-PP instance.

Using the concept of partial preemption, we can model not only the non-preemption constraints linked to safety but also we can use it to model resources having complex setup operations. Even if the setup times are not significant enough to be included in the model, due to process complexity and safety, it is not desirable to frequently change the configuration of these resources. Most of the time, the complexity of the resuming setup time is related only to a subset of resources, while the others can be easily preempted and resumed without significant impact. We can then declare the resources with a significant setup as non-preemptive, and those with an insignificant setup as preemptive. In this way, we can better manage the preemption over activities with significant setup, at the same time we delete the subjectivity of choosing the penalty (M_i) needed in our previous

approach (MSPSP with penalty for preemption, see Chapter 4). Indeed, choosing the right penalty (M_i) is a tricky job. If M_i is too big, there is no interest in allowing the preemption of the activity, what may increase the $C_{\max}$ of the project. However, if M_i is too small, this could cause activities to be preempted too many times disturbing the correct development of activities.

Including the concept of partial preemption in traditional scheduling problems may allow a reduction in the $C_{\max}$. In fact, for almost every scheduling problem studied in the scientific literature, if an activity requires a non-preemptive resource then this activity must be handled as non-preemptive, which produces an increase in the $C_{\max}$. Partial preemption acquires prominent importance when we schedule activities having a narrow time-window or when resource availability varies over time. Knowing that preemptive versions give better values of $C_{\max}$, we can then establish the following relation for makespan value:

$$C_{\max}(Preemptive) \leq C_{\max}(Partially\ preemptive) \leq C_{\max}(Non\text{-}preemptive)$$

The complexity of the MSPSP with partial preemption can be established using the classical RCPSP as a starting point. For each instance of the RCPSP, we can match an instance of the MSPSP with partial preemption, where all resources are mono-skilled, and none of the resources can be preempted. Thus, we can define the RCPSP as a particular case of the MSPSP with partial preemption. Since the RCPSP has been proved to be strongly NP-hard [33], we can, therefore, infer that the MSPSP with partial preemption is also strongly NP-hard.

We present below various formulations for the MSPSP-PP using Mixed-Integer/Linear Programming (MILP) and Constraint Programming (CP). Early versions of these formulations have been presented in [143, 144].

5.2 MILP formulations

For modelling the MSPSP-PP, we use time-indexed formulations again. Most of the constraints are similar to those proposed for the MSPSP with penalty for preemption (Chapter 4). The main difference resides in the way we handle the preemption. Before, we wanted to know how many times an activity was preempted. For the MSPSP-PP we must identify all periods over which the activity remains stopped. Additionally to the parameters described in Table 4.1 (page 54), we also use an additional parameter $PR_{i,k} \in \{0, 1\}$ that indicates whether the activity $i \in \overline{PP}$ allows the release of resource $k \in \overline{K}$ during the preemption periods or not. The parameter is equal to 0 if the resource can be released, 1 otherwise.

5.2.1 Model MSPP1

This model is an adaptation of Model MSWP1 of Section 4.2.1 for the MSPSP with partial preemption. We have decided to present the complete model again to facilitate

the reading of the document.

5.2.1.1 Decision variables

We reuse most of the variables presented for Model MSWP1 of the MSPSP with penalty for preemption; except for the variables related to the preemption ($X_{i,t}$) that are replaced by a set of three new binary variables.

Activity executions:

$$Y_{i,t} = \begin{cases} 1 & \text{if activity } i \in \overline{I} \text{ is executed during time } t \in \overline{H} \\ 0 & \text{otherwise} \end{cases}$$

Technician allocations:

$$O_{j,i,t} = \begin{cases} 1 & \text{if technician } j \in \overline{J} \text{ is allocated to activity } i \in \overline{I} \text{ during time } t \in \overline{H} \\ 0 & \text{otherwise} \end{cases}$$

Auxiliary allocation variables:

$$S_{j,i} = \begin{cases} 1 & \text{if technician } j \in \overline{J} \text{ is assigned to execute activity } i \in \overline{NP} \\ 0 & \text{otherwise} \end{cases}$$

Start time of the activities: These step variables will take the value of 1 for all $t \in \overline{H}$ higher than or equal to the start time of activity $i \notin \overline{P}$.

$$Z_{i,t} = \begin{cases} 1 & \text{if activity } i \notin \overline{P} \text{ begins at time } t \in \overline{H} \text{ or before.} \\ 0 & \text{otherwise} \end{cases}$$

End time of the activities: These step variables will take the value of 1 for all $t \in \overline{H}$ lower than or equal to the end time of activity $i \notin \overline{P}$. Together with variables $Z_{i,t}$, these variables allow us to know the execution interval (start and end) of each activity $i \notin \overline{P}$.

$$W_{i,t} = \begin{cases} 1 & \text{if activity } i \notin \overline{P} \text{ ends at time } t \in \overline{H} \text{ or after} \\ 0 & \text{otherwise} \end{cases}$$

Preemption periods: These on/off variables indicate the periods over which an activity is being preempted. They will allow ensuring the continuous allocation of non-preemptive resources within the execution interval (start, finish) of the activity.

These variables are only defined for partially preemptive activities ($i \in \overline{PP}$).

$$Pp_{i,t} = \begin{cases} 1 & \text{if the activity } i \in \overline{PP} \text{ is being preempted at time } t \in \overline{H} \\ 0 & \text{otherwise} \end{cases}$$

Project makespan:

$C_{\max} \in \mathbb{R}_+$: project makespan

5.2.1.2 Constraints

Objective function: Minimisation of the project makespan:

$$\min(C_{\max}) \tag{5.1}$$

Subject to:

$$\left(\sum_{i \in \overline{I}} Y_{i,t} + \sum_{i \in \overline{PP}} PR_{i,k} * Pp_{i,t}\right) * Br_{i,k} \leq DR_{k,t} \quad \forall k \in \overline{K}, \forall t \in \overline{H} \tag{5.2}$$

$$\sum_{i \in \overline{I}} O_{j,i,t} \leq DO_{j,t} \quad \forall j \in \overline{J}, \forall t \in \overline{H} \tag{5.3}$$

$$Y_{i,t} * Bc_{i,c} \leq \sum_{j \in \overline{J}} (O_{j,i,t} * CO_{j,c}) \quad \forall i \in \overline{I}, \forall c \in \overline{C}, \forall t \in \overline{H} \tag{5.4}$$

$$\sum_{j \in \overline{J}} O_{j,i,t} \geq Y_{i,t} * Nt_i \quad \forall i \in \overline{I}, \forall t \in \overline{H} \tag{5.5}$$

$$\sum_{t=\max(1,r_i)}^{\min(dl_i,T)} Y_{i,t} \geq D_i \quad \forall i \in \overline{I} \tag{5.6}$$

$$D_i * (1 - Y_{l,t}) \geq \sum_{t'=t}^{T} Y_{i,t'} \quad \forall (i,l) \in \overline{E}, \forall t \in \overline{H} \tag{5.7}$$

$$Z_{i,t} \geq Y_{i,t'} \quad \forall i \notin \overline{P}, \forall t \in \overline{H}, \forall t' \leq t \tag{5.8}$$

$$W_{i,t} \geq Y_{i,t'} \quad \forall i \notin \overline{P}, \forall t \in \overline{H}, \forall t' \geq t \tag{5.9}$$

$$Pp_{i,t} = Z_{i,t} + W_{i,t} - Y_{i,t} - 1 \quad \forall i \in \overline{PP}, \forall t \in \overline{H} \tag{5.10}$$

$$Z_{i,t} + W_{i,t} - Y_{i,t} = 1 \quad \forall i \in \overline{NP}, \forall t \in \overline{H} \tag{5.11}$$

$$O_{j,i,t} \geq S_{j,i} + Y_{i,t} - 1 \quad \forall i \in \overline{NP}, \forall j \in \overline{J}, \forall t \in \overline{H} \tag{5.12}$$

$$O_{j,i,t} \leq S_{j,i} \quad \forall i \in \overline{NP}, \forall j \in \overline{J}, \forall t \in \overline{H} \tag{5.13}$$

$$C_{\max} \geq t * Y_{i,t} \quad \forall i \in \overline{I}, \forall t \in \overline{H} \tag{5.14}$$

Constraints (5.2) ensure the satisfaction of resources needs during the execution periods ($Y_{i,t} = 1$) and also the satisfaction of needs for non-preemptive resources ($PR_{i,k} = 1$) during the preemption periods ($Pp_{i,t} = 1$). Constraints (5.3) guarantee the respect of technicians availability, and disjunctive constraint. With constraints (5.4) and (5.5) we ensure that the needs of resources, skills and minimal number of technicians are satisfied during the execution periods. Constraints (5.6) enforce each activity to be in process according to its duration and within its time window. Precedence relationships constraints are given in (5.7). Constraints (5.8) force the auxiliary binary variable $Z_{i,t}$ to be equal to 1 for all periods equal or greater than the start date of the activity. Constraints (5.9), on the other hand, force the auxiliary binary variable $W_{i,t}$ to be equal to 1 for all periods equal to or lower than the completion time of the activity. Using constraints (5.10) we determine the periods during which an activity has been preempted (or not) for partially preemptive activities. These constraints state that activities within their execution interval ($Z_{i,t} = 1$ and $W_{i,t} = 1$ simultaneously) must be either in execution ($Y_{i,t} = 1$) or preempted ($Pp_{i,t} = 1$). Constraints (5.11) ensure that there is no preemption for non-preemptive activities. Constraints (5.12) and (5.13) state that all technicians allocated to a non-preemptive activity must execute it until its completeness. Finally, constraints (5.14) express the project total duration.

One could add some redundant constraints to the model seeking faster convergence. Constraints (5.15) force variables ($Y_{i,t}$) to be equal to 0 before the release date and after the deadline.

$$\sum_{t \in [1,\ldots,r_i-1] \cup [dl_i+1,\ldots,T]} Y_{i,t} = 0 \quad \forall i \in \overline{I} \tag{5.15}$$

Given the variables $W_{i,t}$ and $Z_{i,t}$, one could replace the precedence constraints (5.7) by a disaggregated version:

$$Z_{l,t} + W_{i,t} \leq 1 \quad \forall (i,l) \in \overline{E}, \forall t \in \overline{H} \tag{5.16}$$

In the following of this document, Model MSPP1a will be the MILP model using (5.7) as precedence constraints, and MSPP1b the model where constraint (5.7) is replaced by (5.16).

5.2.2 Model MSPP2

5.2.2.1 Decision variables

This second formulation does not use anymore the binary auxiliary variables $W_{i,t}$ and $Z_{i,t}$. This time, we use two continuous variables G_i and F_i indicating the start and finish time of activity i (as for Model MSWP2 for the MSPSP with penalty for preemption).

Start time of activities: Indicate the start time of activity $i \in \overline{I}$.

$G_i \in \mathbb{R}_+$: start time of activity $i \in \overline{I}$

Finish time of activities: Indicate the finish time of activity $i \in \overline{I}$.

$F_i \in \mathbb{R}_+$: finish time of activity $i \in \overline{I}$

5.2.2.2 Constraints

Objective function: Minimise the makespan of the project (5.1).

Subject to: Most of the constraints are the same than those of Model MSPP1. Equations (5.2) to (5.6) and (5.12) to (5.14) remain unchanged. Constraints (5.7) to (5.11) are changed by:

$$F_i + 1 \leq G_l \qquad \forall (i,l) \in \overline{E} \tag{5.17}$$

$$Pp_{i,t} \leq 1 - Y_{i,t} \qquad \forall i \in \overline{PP}, \forall t \in \overline{H} \tag{5.18}$$

$$F_i - G_i + 1 \leq D_i + \sum_{t \in \overline{H}} Pp_{i,t} \qquad \forall i \in \overline{PP} \tag{5.19}$$

$$F_i - G_i + 1 \leq D_i \qquad \forall i \in \overline{NP} \tag{5.20}$$

$$F_i \geq t * Y_{i,t} \qquad \forall i \in \overline{I}, \forall t \in \overline{H} \tag{5.21}$$

$$G_i \leq t * Y_{i,t} + (1 - Y_{i,t}) * T \qquad \forall i \in \overline{I}, \forall t \in \overline{H} \tag{5.22}$$

Precedence relationships are given by constraints (5.17). With constraints (5.18) we indicate that $Pp_{i,t}$ must be zero if activity i is in execution at time t. Constraints (5.19) ensure that $Pp_{i,t}$ takes value 1 for periods where activity i has been preempted. Constraints (5.20) guarantee that activities within $\overline{NP}$ are not preempted. The finish time of each activity is calculated with constraints (5.21). Constraints (5.22) calculate the start time of each activity.

We still must ensure that variables $Pp_{i,t}$ is equal to zero for all time t outside the activity execution (t) lower than start time, and t higher than finish time. We can model these constraints in two different ways. The first one using the continuous variables F_i and G_i (this configuration will be presented as Model MSPP2a during the computational experiments):

$$F_i \geq t * Pp_{i,t} \qquad \forall i \in \overline{PP}, \forall t \in \overline{H} \tag{5.23}$$

$$G_i \leq t * Pp_{i,t} + (1 - Pp_{i,t}) * T \qquad \forall i \in \overline{PP}, \forall t \in \overline{H} \tag{5.24}$$

The second way to express theses constraints (Model MSPP2b) is using only $Y_{i,t}$

variables:

$$Pp_{i,t} \leq \sum_{t'=1}^{t} Y_{i,t'} \quad \forall i \in \overline{PP}, \forall t \in \overline{H} \tag{5.25}$$

$$Pp_{i,t} \leq \sum_{t'=t}^{T} Y_{i,t'} \quad \forall i \in \overline{PP}, \forall t \in \overline{H} \tag{5.26}$$

5.2.3 Theoretical comparison

Theorem 2. *Formulation MSPP1 is at least as tight as Formulation MSPP2.*

Proof. To show that Model MSPP1 dominates Model MSPP2, we define the following transformation.

$$G_i = T - \sum_{t=1}^{T} Z_{i,t} + 1 \text{ and } F_i = \sum_{t=1}^{T} W_{i,t}$$

(Reminder: G_i is the start time and F_i is the end time.)

We can show that the constraints of Model MSPP2 involving the F_i variables are implied by Model MSPP1 constraints and the transformation (we can restrict to activities in $\overline{PP}$ without loss of generality) taking the continuous domain $[0, 1]$ for all binary variables.

For Constraints (5.19), we have

$$\begin{aligned} F_i - G_i + 1 &= \textstyle\sum_{t=1}^{T} W_{i,t} + \sum_{t=1}^{T} Z_{i,t} - T \\ &= \textstyle\sum_{t=1}^{T} Pp_{i,t} + \sum_{t=1}^{T} Y_{i,t} \text{ from Constraints (5.10)} \\ &= \textstyle\sum_{t=1}^{T} Pp_{i,t} + D_i \quad (5.19) \end{aligned}$$

For the precedence constraints (5.17) we have $G_l = T - \sum_{t=1}^{T} Z_{l,t} + 1$ and $F_i + 1 = \sum_{t=1}^{T} W_{i,t} + 1$. To obtain (5.17) we must show that $G_l - F_i - 1 \geq 0$. We have

$$G_l - F_i - 1 \quad = T - \textstyle\sum_{t=1}^{T} Z_{l,t} - \sum_{t=1}^{T} W_{i,t}$$

(5.17) is satisfied if and only if

$$\sum_{t=1}^{T} Z_{l,t} + \sum_{t=1}^{T} W_{i,t} <= T$$

This is a consequence of precedence constraints (5.16)

$$Z_{l,t} + W_{i,t} \leq 1 \quad \forall t = 1, \ldots, T$$

By using Constraints (5.9) and $F_i = \sum_{t=1}^{T} W_{i,t}$ we can also obtain constraints (5.21). Constraints (5.22) are implied by constraints (5.9) and $G_i = T - \sum_{t=1}^{T} Z_{i,t} + 1$. □

Computational tests of the lower bound obtained by releasing the integrity constraints of models MSPP1 and MSPP2, using the same set of 200 instances presented in Section 5.4, proved that all configurations of Model MSPP1 always generated tighter lower bounds than both configurations of Model MSPP2. This, together with Theorem 2, suggest that the formulation MSPSP1 is, in fact, stronger than formulation MSPSP2.

We also tried to compare the theoretical tightness of Models MSPP1a, that uses (5.7) as precedence constraints, and MSPP1b, that uses constraints (5.16) instead. We could not analytically prove that one model is better than the other. Computational tests of the lower bound suggest that there is no total dominance by any of the models. In fact, Model MSPP1a gives better results for 45 instances, Model MSPP1b gives better results for 6 instances, and both models give the same values for the remaining 149 instances. In regard of these results, one could construct a new formulation of MSPP1 using both constraints (5.7 and 5.16) that should always give better lower bounds. We call this new formulation MSPP1c in the computational tests in Section 5.4.1.

5.3 CP formulation

As discussed in Chapter 2 and Chapter 3, constraint programming has attracted high attention among experts from many areas of computer science in the last decades due to its potential for solving hard real-life problems. The increasing interest for this technique led us to use it and evaluate its performance over the studied problem. We then propose to use CP for modelling the MSPSP with partial preemption. To model the MSPSP-PP, we use the software IBM CP Optimizer (CPO), making use of the concept of interval variables, a constrained object tailored to scheduling problems, and also other specific scheduling constraints already defined in CPO [104][1].

Variables

Activities execution intervals: The $itvs_i$ interval variables will indicate the interval between the start and the end of each activity $i \in \overline{I}$. For non-preemptive activities, the size of the interval must be equal to the duration of the activity (D_i). For preemptive and partially preemptive activities, the size of the intervals varies from D_i to T (the solver must decide the final size of the interval variable).

Intervals for each part of activities : In this model, each preemptive ($i \in \overline{P}$) and partially preemptive activity ($i \in \overline{PP}$) is divided in D_i parts of unitary duration. The $par_{i,v}$ interval variable indicates the interval during which each unit of duration

[1]We thank Dr. Philippe Laborie, who helped us to improve the CP formulation.

$v \in \overline{V_i}$ ($\overline{V_i} = \{1, .., D_i\}$) of activity $i \notin \overline{NP}$ is executed. For non-preemptive activities we generate only one part ($\overline{V_i} = \{1\}$) with size equal to the activity duration.

Technicians allocation: We made use of the optional interval variables proposed in CPO. Optional interval variables may or may not be present in the solution, so as to satisfy the constraints. The interval variable $InTech_{j,i,v}$ indicates the period when technician $j \in \overline{J}$, if present, is working in the part $v \in \overline{V_i}$ of activity $i \in \overline{I}$.

Number of technicians allocated to each part: The integer variables $nTech_{i,v}$ indicate the number of technicians that are allocated to execute the part $v \in \overline{V_i}$ of activity $i \in \overline{I}$.

Number of technicians per skill allocated to each part: The integer variables $nSk_{c,i,v}$ indicate the number of technicians mastering skill $c \in \overline{C}$ allocated to part $v \in \overline{V_i}$ of activity $i \in \overline{I}$.

Objective function

The objective in the MSPSP-PP is to minimise the project makespan. This can be expressed in CPO as follows:

$$minimise\left(\max_{\forall i \in \overline{I}}\{itvs_i.end\}\right) \tag{5.27}$$

Starting from the idea of trying to allocate always a minimum number of technicians to the activities, one can add a secondary criterion (lexicographic, using `staticLex` of CPO) that minimises the total number of technician allocations. The function `staticLex` defines a multi-criteria policy, ordering the different criteria and performing lexicographic optimisation. The first criterion is considered to be the most important, and any improvement of this criterion is worth any loss on the other criteria. The solver should be able to found better solutions faster when using the minimisation of the total number of technician allocations as secondary objective. The objective function can then be defined as:

$$minimise\left(staticLex\left(\max_{\forall i \in \overline{I}}\{itvs_i.end\}, \sum_{i \in \overline{I}} \sum_{v \in \overline{V}} nTech_{i,v}\right)\right) \tag{5.28}$$

Constraints

Span(a, {b1, .., bn}) constraint states that the interval variable a (if present) spans over all present interval variables from the set *{b1, .., bn}*. In other words, interval variable a starts together with the first present interval from *{b1, .., bn}* and ends together with

the last present interval. We use this kind of constraint to span the $par_{i,v}$ variables within the $itvs_i$ variables.

$$span(itvs_i, par_{i,v} : \forall v \in \overline{V_i})\ \forall i \in \overline{I} \tag{5.29}$$

We also need to ensure that there is no overlap for the parts of each activity. The predefined constraint `endBeforeStart(a,b)` indicates that the interval variable a must end before the interval variable b begins. These constraints are only necessary for preemptive and partially preemptive activities, and can be stated as follows:

$$endBeforeStart(par_{i,v}, par_{i,s})\ \forall i \notin \overline{NP}, \forall v \in \overline{V_i}, \forall s \in \overline{V_i} : s > p \tag{5.30}$$

Let us define $rUsage_k$ as a cumulative function indicating the usage of resource k over time, and let DR_k be a cumulative function indicating the resource capacity over time. Also let $pulse(IN, h)$ be an elementary pulse function taking the value of h over the interval IN. Preemptive resources ($PR_{i,k} = 0$) are used during the execution intervals of the parts ($par_{i,v}$). Non-preemptive resources ($PR_{i,k} = 1$), on the other hand, must be allocated during the whole execution interval of the activity ($itvs_i$). We can state the resource constraint as follows:

$$\begin{aligned} rUsage_k = & \sum_{i \in \overline{I}: PR_{i,k}=0} \sum_{p} pulse(par_{p,i}, Br_{i,k}) + \\ & \sum_{i \in \overline{I}: PR_{i,k}=1} pulse(itvs_i, Br_{i,k}) \quad \forall k \in \overline{K} \\ & rUsage_k \leq DR_k \quad \forall k \end{aligned} \tag{5.31}$$

We must guarantee that each technician is allocated to at most one activity at a time. For this, we use the predefined `noOverlap({b1, .., bn})` constraint that states that none of the interval variables within the set *{b1, .., bn}* overlaps over time. Note that we could use this expression for establishing the no overlap constraint of $Par_{i,v}$ variables (5.30). However, the way we declare it allows us to break some symmetries on the model. The disjunctive constraint over the technicians is then defined as:

$$noOverlap(InTech_{j,i,v} : \forall i \in \overline{I}, \forall v \in \overline{V_i})\ \forall j \in \overline{J} \tag{5.32}$$

Technicians cannot be assigned during their absence periods. To model these constraints, we define a step function describing the present and absent periods of each technician ($PreTech_j$). We must also use the predefined constraint `forbidExtent(a,F)`. This expression states that whenever the interval variable a is present, it cannot overlap a point t where the step function $F(t) = 0$. We ensure that absence/presence periods

are respected as follows:

$$forbidExtent(InTech_{j,i,p}, PreTech_j)\, \forall j, \forall i, \forall p \tag{5.33}$$

For ensuring the skills requirements we use the expression $alternative(a, \{b_1, .., b_n\}, c)$. The alternative constraint will enforce that if a is present, then c and only c of the interval variable within $\{b_1, .., b_n\}$ will be present, and synchronised with a. In other words, c interval variables will be selected among the set and those c selected intervals will have to start and end together with interval variable a. We can use this constraint to select the technicians that will fulfil each skill for every part of an activity. It can be defined as:

$$\begin{aligned} alternative(par_{i,v}, InTech_{j,i,v} : \forall j \in \overline{J} : CO_{j,c} = 1, nSk_{c,i,v}) \\ \forall i \in \overline{I}, \forall v \in \overline{V_i}, \forall c \in \overline{C} \end{aligned} \tag{5.34}$$

The number of selected technicians for each skill and part goes from the skill requirement of the activity ($Bc_{i,c}$) up to the maximum between the minimal number of required technicians (Nt_i) and the sum of all the skill needs of the activity. Constraints (5.34) and (5.35) ensure the respect of skill requirements.

$$Bc_{i,c} \leq nSk_{c,i,v} \leq max \left\{ Nt_i, \sum_{c' \in \overline{C}} Bc_{i,c'} \right\} \;\; \forall i \in \overline{I}, \forall v \in \overline{V_i}, \forall c \in \overline{C} \tag{5.35}$$

We use the alternative constraint again to ensure the satisfaction of the minimal number of technicians:

$$alternative(par_{i,v}, InTech_{j,i,v} : \forall j \in \overline{J}, nTech_{i,v}) \;\; \forall i \in \overline{I}, \forall v \in \overline{V_i} \tag{5.36}$$

The number of technicians allocated to each part of an activity will vary from the maximum between Nt and the highest skill requirement, up to the maximum between Nt_i and the sum of all skill requirements. Together with constraints (5.36), these constraints ensure the allocation of the minimal number of technicians.

$$\max \left\{ Nt_i, \max_{\forall c \in \overline{C}} \{Bc_{i,c}\} \right\} \leq nTech_{i,v} \leq \max \left\{ Nt_i, \sum_{c \in \overline{C}} Bc_{i,c} \right\} \;\; \forall i \in \overline{I}, \forall v \in \overline{V_i} \tag{5.37}$$

The precedence relationships can be stated as:

$$endBeforeStart(itvs_i, itvs_l)\; \forall (i, l) \in \overline{E} \tag{5.38}$$

The satisfaction of the deadlines and release dates are guaranteed by:

$$itvs_i.end \leq dl_i \;\; \forall i \in \overline{I} \tag{5.39}$$

$$r_i \leq itvs_i.start \ \ \forall i \in \overline{I} \tag{5.40}$$

Since not all the technicians are available at the same time, we can add some redundant constraints to improve the lower bounds as well as the constraint propagation. Let the parameters $avTech_t$ be the amount of available technicians during period t, and $avSk_{c,t}$ the number of available technicians at time t mastering skill c. We define two cumulative functions $TechUsage$ and $SkUsage_c$ indicating the number of technicians allocated over time, and the number of technicians mastering skill c allocated over time, respectively. We get the value for these functions as follows:

$$TechUsage = \sum_{i \in \overline{I}} \sum_{v \in \overline{V_i}} pulse\left(par_{i,v}, \max\left\{Nt_i, \max_{\forall c \in \overline{C}}\{Bc_{i,c}\}\right\}\right)$$

$$SkUsage_c = \sum_{i \in \overline{I}} \sum_{v \in \overline{V_i}} pulse(Bc_{i,c}) \ \ \forall c \in \overline{C}$$

The $alwaysIn(Cum, B, min, max)$ constraint is used to confine the values of a cumulative function Cum during an interval $[u, v)$ inside interval [min,max]. We can then limit the number of technicians allocated at each period t as follows:

$$alwaysIn(TechUsage, t, t+1, 0, avTech_t) \ \forall t \in \overline{H} \tag{5.41}$$

$$alwaysIn(SkUsage_c, t, t+1, 0, avSk_{c,t}) \ \forall t \in \overline{H}, \forall c \in \overline{C} \tag{5.42}$$

5.4 Computational Experiments

For computational tests, we use again a computer equipped with an Intel Xeon E5-2695 processor at 2.3 GHz running Ubuntu 16.04. We use CPLEX 12.7 and CP Optimizer 12.7 for solving the MILP models and the CP model, respectively (using the default configuration and limiting the number of threads used by the solvers at 8). The computation time was limited to 10 minutes. We generated four sets, each of them having 50 instances, varying the proportion of preemption type present in the instance (as shown in Table 5.1) using a random generation algorithm. All instances have 30 activities with a duration between 5 to 10 time units, up to 15 skills, 8 cumulative resources, 8 technicians (multi-skilled resources) divided into two teams, 20% of activities with time windows, the density of precedence relationships is low, and an average optimum $C_{\max}$ between 70 and 90 time units.

5.4.1 Testing MILP formulations

We test first the performance of different configurations of Model MSPP1. Since time-indexed formulations require an initial estimation of the scheduling horizon, we initially tested the two configurations using the sum of activity durations as the scheduling hori-

	Set A1	Set B1	Set C1	Set D1
Non-preemptive	10%	10%	80%	33.3%
Partially preemptive	10%	80%	10%	33.3%
Preemptive	80%	10%	10%	33.3%

Table 5.1: Distribution of preemption types for instances of the MSPSP-PP

zon. Table 5.2 presents the number of instances for which each configuration was able to found initial solutions within the time limit. Configuration MSPP1a was able to find a larger number of initial solutions. Results for models MSPP1b and MSPP1c suggest that the use of constraint (5.16) reduces the capacity of the MILP solver to find initial solutions. All MSPP1 configurations seem to perform better when the proportion of preemptive activities is high (Set A1). This performance decreases when the proportion decreases, obtaining the worst results for Set C1 (none of the configurations was able to find initial solutions for any of the instances within this set).

	Model MSPP1a			**Model MSPP1b**			**Model MSPP1c**		
	Number of instances with initial solution	Number of instances solved to optimality	Average time to optimality	Number of instances with initial solution	Number of instances solved to optimality	Average time to optimality	Number of instances with initial solution	Number of instances solved to optimality	Average time to optimality
Set A1	47	14	312.54 s	46	14	260.39 s	46	11	333.03 s
Set B1	32	0	-	16	0	-	14	0	-
Set C1	0	0	-	0	0	-	0	0	-
Set D1	7	0	-	8	1	537.09 s	10	0	-
All	86	14	312.54 s	70	15	278.84 s	70	11	333.03 s

Table 5.2: Results for all configurations of Model MSPP1 without warm start

A second test was carried out using the warm start option of CPLEX. We used an initial solution obtained by the greedy algorithm presented in Section 6.2. Table 5.3 presents the number of instances for which optimality was proved, the average time required to prove the optimality and the average optimality gap (the perceptual difference between the optimal solution or best known lower bound and the solution obtained) for each configuration of Model MSPP1. From results on Tables 5.2 and 5.3, one can conclude that the use of the constraints (5.7) and (5.16) simultaneously does not improve the practical performance of the formulation MSPP1. In fact, it may even have a negative impact since it reduces the number of instances with initial solution and the number of instances for which the optimality was proven. Configurations MSPP1a and MSPP1b have similar behaviour, and statistical tests do not allow to prove any difference in the average time to optimality or in the average optimality gap. Again, all MSPP1 configurations performs very well when the percentage of preemptive activities is high, while they gave the worst results for highly non-preemptive instances.

A similar analysis was done for testing both configurations of Model MSPP2. Table 5.4 presents the results for the two configurations without warm start. The model configuration with constraints (5.25) and (5.26), Model MSPP2b, shows a better performance, being able to find a more significant number of initial and optimal solutions.

	Model MSPP1a			**Model MSPP1b**			**Model MSPP1c**		
	Number of instances solved to optimality	Average time to optimality	Average gap	Number of instances solved to optimality	Average time to optimality	Average gap	Number of instances solved to optimality	Average time to optimality	Average gap
Set A1	44	122.64 s	0.12 %	47	110.85 s	0.01%	47	123.28 s	0.05 %
Set B1	20	240.37 s	1.59 %	19	262.99 s	1.68 %	15	204.79 s	2.16 %
Set C1	0	-	9.37 %	0	-	9.43 %	0	-	9.43%
Set D1	16	286.36 s	2.23 %	18	289.35 s	1.85 %	16	267.32 s	2.30 %
All	80	184.81 s	3.33 %	84	183.51 s	3.24 %	78	168.50 s	3.48 %

Table 5.3: Results for all configurations of Model MSPP1 with warm start

Both configurations have a bad performance when the proportion of non-preemptive activities increases, and have better results when the proportion of preemptive activities is high. When tested using warm start (see Table 5.5), we have not enough statistical evidence to conclude that one configuration outperforms the other one for all the instances. However, if we look only the instances from the set A1 (with a high proportion of preemptive activities), one can say that configuration MSPP2b is faster, and allows to get a lower average gap. For all the other sets, the performances seem to be statistically equal.

	Model MSPP2a			MSPP2b		
	Number of instances with initial solution	Number of instances solved to optimality	Average time to optimality	Number of instances with initial solution	Number of instances solved to optimality	Average time to optimality
Set A1	43	10	332.47 s	48	16	335.2 s
Set B1	3	-	-	25	-	-
Set C1	2	-	-	4	-	-
Set D1	14	-	-	29	-	-
All	62	10	332.47 s	106	16	335.2 s

Table 5.4: Results for Model MSPP2 without warm start

	Model MSPP2a			Model MSPP2b		
	Number of instances solved to optimality	Average time to optimality	Average gap	Number of instances solved to optimality	Average time to optimality	Average gap
Set A1	45	124.23 s	0.09%	46	87.39 s	0.05%
Set B1	14	119.19 s	2.79%	15	154.12 s	2.69%
Set C1	0	-	9.45%	0	-	9.45%
Set D1	19	194.22 s	2.12%	19	216.12 s	1.99%
All	78	140.37 s	3.61%	80	130.48 s	3.55%

Table 5.5: Results for Model MSPP2 using warm start

If we compare the results for the best configurations of Model MSPP1 and Model

MSPP2, we can conclude that Model MSPP2 (configuration *b*) outperforms Model MSPP1 when warm start is not used, finding a larger number of feasible solutions. When using warm start, Model MSPP1b is able to prove the optimality for a bigger number of instances, and give a lower average gap for all sets of instances. However, Model MSPSP2b seems to be faster for instances from the sets A1 (when the proportion of preemptive activities is high). Both models have difficulties for finding initial solutions or proving the optimality when the instances have a high proportion of non-preemptive activities. The computational results confirm one more time that a theoretically stronger formulation does not imply better practical performance.

5.4.2 Testing CP formulation

For the CP formulation, we wanted to know whether using a lexicographic objective could improve the performance of the model. Table 5.6 presents the results for the two configurations after 10 minutes of computation. Unlike what happened for the MILP models, both CP model configurations were able to find initial solutions for all the instances. One sees that the use of lexicographic objective function leads to an increase in the number of instances solved to optimality, and therefore, a decrease in the average gap. CP configuration with lexicographic objective function beats the single objective configuration for all sets of instances. One can then conclude that the use of lexicographic objective function improves the performance of the CP model. Both CP configurations perform better when the proportion of non-preemptive activities is high, being able to prove the optimality of a more significant number of instances in a shorter average time. We could not find enough statistical evidence to conclude about the impact of partially preemptive and preemptive activities.

	Single objective			Lexicographic objective		
	Number of instances solved to optimality	Average time to optimality	Average gap	Number of instances solved to optimality	Average time to optimality	Average gap
Set A1	0	-	4.42%	37	180.32 s	0.16%
Set B1	0	-	4.41%	33	141.92 s	0.43%
Set C1	35	133.16 s	1.10%	37	107.58 s	0.72%
Set D1	3	249.28 s	3.80%	33	161.43 s	0.44%
All	38	142.32 s	3.43%	140	147.59 s	0.44%

Table 5.6: Results for the CP formulation

When using warm start on the lexicographic model (Table 5.7), the average gap and the average time required to prove the optimality are reduced in almost a half, compared with the results without warm start. For instances within the set A1, the average gap was not reduced, but the time required to prove the optimality of the instances was reduced. For instances in set C1, on the other hand, the average time to optimality did

not change, while the average gap was reduced to a half. For sets B1 and D1, both the average time to optimality and the average gap were reduced.

	Number of instances solved to optimality	Average time to optimality	Average gap
Set A1	39	67.17 s	0.18%
Set B1	40	88.01 s	0.15%
Set C1	41	108.73 s	0.39%
Set D1	40	76.14 s	0.21%
All	160	85.27 s	0.23%

Table 5.7: Results for CP model with warm start

Analysing results from Tables 5.3, 5.5, and 5.7, one can see that all MILP models outperform CP when the percentage of preemptive activities is high (set A1), proving the optimality of a higher number of instances, and giving a lower average gap. CP, on the other hand, gives better results when this percentage is low. One could then say that the two methods are complementary. Future research should be done in order to develop a hybrid method that better exploit the characteristics of each instance.

5.5 Concluding remarks

In this chapter, we presented a new variant of the MSPSP that made use of the concept of partial preemption. This concept leads to a limited release of the resources during the preemption periods. We used two different techniques (MILP and CP) to formalise the problem. Various possible formulations have been presented for each technique. A theoretical analysis allows concluding that the MSPP1 formulation, using binary variables to express the precedence relationships, is stronger than the MSPP2 formulation, that uses continuous variables. Computational experiments allowed us to study the performance of all MILP and CP formulations. The MILP formulations showed an outstanding performance in the presence of a high proportion of preemptive or partially preemptive activities. However, they start having troubles to find initial solutions and prove the optimality when the proportion of non-preemptive activities increases. The CP formulation, on the other hand, presented an opposite behaviour; it performs better when the instances are highly non-preemptive. This behaviour leads us to think that the two modelling techniques could be complementary. As future research, we must then study better ways to combine and exploit the advantages of both techniques.

Even if CP with seems to find very good solutions (all average gaps lower than 0.72% on table 5.6) at the time limit for small instances, the MILP and CP models presented in this chapter could be not fast enough to generate good quality solutions in short time

for large instances. Having good solutions in reduced time is essential for the industrial application of this problem. That is why, in the following chapter, we present various heuristic methods aiming to answer the industrial need.

Chapter 6

Heuristic Methods for the MSPSP-PP

Contents

6.1 Flow problem for technicians allocation **86**

6.2 Greedy Algorithm: Serial Generation Scheme **88**

6.3 Binary-Tree-based Local Search Algorithm **91**

6.4 Greedy Randomised Adaptive Search Procedure **93**

6.5 Large neighbourhood search **97**

6.6 Computational experiments **100**

6.7 Concluding remarks **107**

As indicated in Chapter 1, we must be able to propose good solutions in short times to answer the industrial needs. The mathematical and logic models presented in Chapter 5 may not answer to this requirement for industrial-size instances (more than 100 activities for the LECA-STAR). In fact, when trying to solve such instances with the MILP models, the solver run out of memory (15 GB of RAM). The CP model scales better but the solver have some issues to prove optimality, so the question arise whether better feasible solutions could be obtained faster. This chapter aims precisely at developing efficient heuristic methods for finding good quality solutions in reasonably fast computational times.

First, we present how the subproblem of technicians allocation can be modelled as a flow problem; this approach is the basis for the proposed methods. A serial greedy algorithm, using priority rules, is then proposed. Aiming to improve the solutions of the greedy algorithm, we present a binary-tree-based search algorithm (published in [145]) and a greedy randomised adaptive search procedure developed in collaboration with Prof. Lars Mönch (published in [146]). Finally, we present a large neighbourhood search algorithm, a hybrid procedure combining exact and heuristic methods. All the proposed heuristics and the CP model are compared on a new set of instances.

6.1 Flow problem for technicians allocation

The MSPSP-PP can be seen as a problem consisting of two coupled subproblems: an activity scheduling problem combined with an allocation problem of the technicians performing each activity. In a heuristic approach, once the order in which the activities will be executed is defined, we still have the problem of choosing the technicians who will perform them. To achieve this allocation in the best way, we must first allocate the technicians with the least chances of being necessary to the activities not yet scheduled, that is to say, the less critical technicians.

Bellenguez-Morineau [26] proposed to model the allocation problem of technicians with the lowest criticality as a Minimum-Cost Maximum Flow (MCMF) problem [6] for the (non preemptive) MSPSP. Her model works on a graph $G_{i,t} = (X, F)$, $X = S_i \cup P_{t,i}$ (Figure 6.1), where S_i represents the set of skills required by activity i and $P_{t,i}$ is the subset of technicians available during period t and who master at least one of the skills required by activity i. F is the set of arcs connecting the nodes. The MCMF problem aims at minimising the cost required to deliver the maximum amount of flow possible in the network.

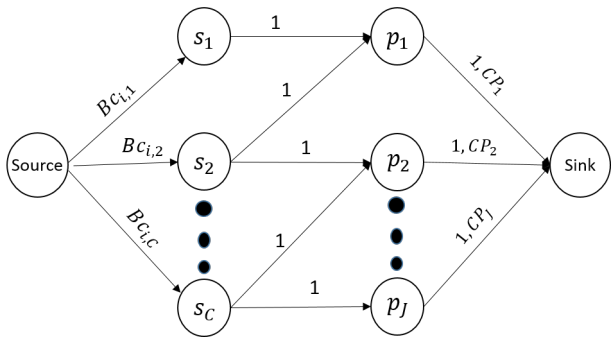

Figure 6.1: Flow graph for the MSPSP

In this graph, there is an edge between the source vertex and each vertex $s_c \in S_i$ whose maximum capacity is equal to $Bc_{i,c}$ (need of the skill c for executing the activity i). There is also an edge between a vertex s_c and a vertex $p_j \in P_{t,i}$, if and only if the technician j masters the skill c. The maximum capacity of this arc is fixed to 1 because, in the MSPSP as defined by Néron [127], a technician can only respond to one unit of need per skill. Similarly, there is an edge between each vertex p_j and the sink of the graph, with a maximum capacity equal to 1 (a technician can only answer one skill per activity). One associates a cost (CP_j) related to the criticality of the technician j to these last arcs. Let $C = |S_i|$ and $J = |P_{t,i}|$. The graph contains $C + J + 2$ nodes and at most $C + J + CJ$ arcs.

Using one of the existing polynomial algorithms (the Edmonds-Karp algorithm [64], for example), one can solve the problem of maximum flow at minimum cost for the

proposed graph. To determine the technicians to allocate, we look at the vertices $p_j \in P_{t,i}$ through which the flow passes.

If the maximum flow going through the graph is less than the sum of the skill needs, we can conclude that there is no possible assignment for this activity at time t.

The graph presented in Figure 6.1 was designed under the hypothesis that each technician can only respond to one skill requirement per activity. However, this constraint has been relaxed for the MSPSP-PP under study because it is not realistic in the industrial problem where technicians can respond to several skills per activity. We then redefine the graph to take this change into account. More precisely, the capacities of the arcs connecting all vertices p_j and the sink are now of at least the number of skills mastered by the technician j plus one $(Nb_j + 1)$. On the other hand, as indicated before, in our industrial problem we must allocate a minimal global number of technicians (Nt_i) for the activity execution independently of the required skills; in order to take this constraint into account, we add an additional vertex s_* linked to the source vertex with a capacity equal to Nt_i and connected to all vertices $p_j \in P_{t,i}$ with a capacity of 1. Concerning the unit cost of the arcs connecting technicians vertices to the sink, we use a cost function $CT_{i,j}$ (Definition 2), which varies according to the technician j and the activity i being scheduled. The new graph is shown in Figure 6.2. There are $C + J + 3$ nodes and at most $C + J + 1 + (C + 1)J$ arcs.

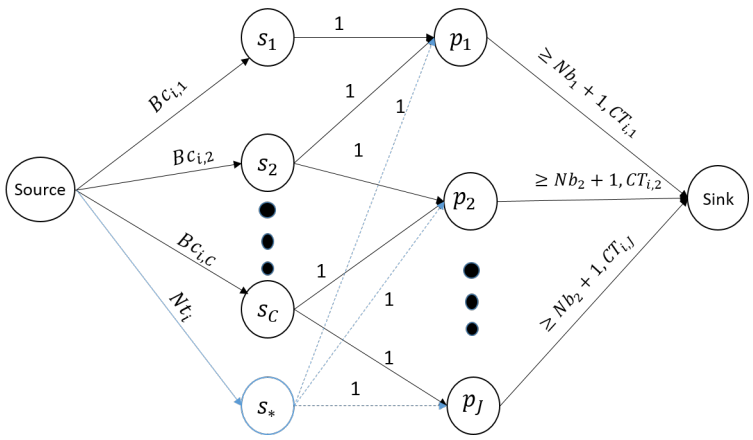

Figure 6.2: Flow graph for the MSPSP-PP

Theorem 3. *There exists a feasible solution to the technician allocation problem if and only if there exists a flow of value* $\sum_{c=1}^{C} Bc_{i,c} + Nt_i$ *in network* $G_{i,t}$.

Proof. Consider a feasible flow of value $\Phi = \sum_{c=1}^{C} Bc_{i,c} + Nt_i$ in $G_{i,t}$. Since the cut between the source and the s_k nodes has capacity Φ, the flow is maximal in the network and all these arcs are saturated. Consider the solution of the technician allocation problem that selects all technicians j who have a non-zero flow on one of the incoming arc of node P_j. For each skill $c \neq *$, due to saturation of the arc from the source to s_c,

there are $Bc_{i,c}$ selected technicians. Due to saturation of arc from the source to node s_*, there are NT_i saturated outgoing arcs from s_*, so at least Nt_i selected technicians. Furthermore since the capacity of the outgoing arc of each node P_j, the number of skills mastered by a technician is not exceeded. Hence the technician allocation solution is feasible. Consider now a feasible technician allocation. We create a flow of value NT_i from the source to s_* and of value 1 from s_* to each selected technician node P_j as well as from each such node P_j to the sink. Then we take the selected technicians in a random order and we update for each of them the flow as follows: If the technician j masters a required skill c, we add one to the flows from the source to node s_c, from s_c to P_j and from P_j to the sink only if the flow from the source to s_c is not already equal to $Bc_{i,c}$. This operation always give a feasible flow and as the number of allocated technicians for each skill is at least $Bc_{i,c}$, this lead to saturating all arcs issued from the source. □

Definition 1. *The correlation indicator $Cr_{i,j}$ expresses the correlation of the technician j and the activity i. It indicates the degree to which activity i might require technician j for its execution. Let us define ST_j as the skill set a technician j masters, and let SA_i be the skill set needed to execute activity i. The correlation indicator is calculated as follows:*

$$Cr_{i,j} = Cardinality(ST_j \cap SA_i) \tag{6.1}$$

Definition 2. *The criticality cost $CT_{i,j}$ of a technician j is an indicator of the degree to which a technician could be requested by the set of not yet scheduled activities (set L). It is directly proportional to the sum of duration (D_l) of every activity $l \in L$ multiplied by the correlation indicator between the technician j and every activity $l \in L$. This cost is inversely proportional to the correlation with the studied activity. This indicator is calculated as follows:*

$$CT_{i,j} = \frac{\sum_{\forall l \in L}(D_l * Cr_{l,j})}{Cr_{i,j}} \tag{6.2}$$

In case of an equality of such a cost for different technicians, we break the ties to ensure that the flow algorithm always minimises the number of technicians allocated to each activity.

6.2 Greedy Algorithm: Serial Generation Scheme

For this heuristic method, we propose to use a serial schedule generation scheme with priority rules. Given a set I containing the activities to be scheduled and sorted according to a priority rule, we take one by one the activities in I and perform their scheduling and technicians allocation (using the proposed method in Section 6.1) sequentially as early as possible. For every activity $i \in I$, we check each time t, beginning with $t = \underset{\sim}{r}_i$ (earliest start time, see Definition 3 below), the ability to schedule the activity during

the period t depending on the type of preemption it has. It is important to remind that, as indicated in Section 5.1, if the technicians are declared as non-preemptive, the activity automatically become non-preemptive and all its cumulative resources are then non-preemptive, what means no interruption of any resource (cumulative or technicians) is allowed. For preemptive activities, on the other hand, all cumulative resources and technicians are preemptive. For partially preemptive activities, technicians are preemptive, and a nonempty subset of cumulative resources is declared as non-preemptive (an empty subset will mean that the activity is preemptive), the remainder can be released during preemption periods. With this in mind, we can check the possibility of scheduling an activity at t as follows:

- For non-preemptive activities, we check the possibility of continuous execution from t to $t + D_i - 1$ (taking into account the availability of resources and technicians), where D_i is the duration of the activity. If the answer is positive, we schedule the whole activity, we update the remaining capacity of resources and technicians from t to $t + D_i - 1$, and move on to the next activity. If continuous execution is not possible, we check for the next t (a period where an event happens: end of an activity, the new availability of technicians, etc.) until the activity can be scheduled.

- For partially preemptive activities, we will first determine the minimum end date (starting from the analysed t period) depending on the availability of preemptive resources and technicians. We will then check the continuous availability of non-preemptive resources. If non-preemptive resources are available without interruption, we allocate them to the activity from t until the end date (we also update their remaining capacity during this periods). Preemptive resources are allocated for periods $t' \in t..end\,date$ where all preemptive resources are available (its remaining capacity is also updated). If continuity for non-preemptive resources is not verified, we go to the next t and repeat until getting an affirmative answer.

- For preemptive activities, the availability of resources and technicians during the t period is checked. If they are available, we allocate them for the period t (updating also its remaining capacity); then increase t and repeat the process until the duration of the activity is complete.

The steps of the serial generation scheme are presented in Algorithm 1.

Definition 3. *"Earliest start time" ($\underset{\sim}{r}_i$) indicates the date before which activity i cannot begin. It is calculated using the precedence constraints and is equal to the longest path from the source vertex (A_0, beginning of the project) to the activity vertex (A_i) in the precedence graph (taking into account the release date, the possible end date of the predecessors, and the availability of resources and technicians).*

Algorithm 1: Greedy Serial Generation Scheme

1. Select an activity from the list (I)
2. Find the earliest periods when this activity can be scheduled according to: its preemption type, and resources and technicians availability
3. Allocate the technicians following the method proposed in Section 6.1:
   - For non-preemptive activities the flow problem is solved only once (same technicians must execute the whole activity)
   - For preemptive and partially preemptive activities, we must solve the flow problem for each unit of duration of the activity
4. Return to Step 1 if there are still activities to be scheduled. Stop otherwise

The presence of deadlines is one of the critical constraints for generating feasible solutions using heuristic methods. In order to maximise the chance of finding feasible solutions, we propose to use a 2-step approach to generate the schedule. As a first step, activities with a deadline and its predecessor activities (set DL) are scheduled following a *slack time-based* priority list. Then, the rest of the activities (set L) are scheduled using different priority rules.

Scheduling Activities With Deadline

For this first part of the heuristic, we use a serial schedule generation scheme using a priority list based on the "slack time" of activities with a deadline (dl_i). Giving priority to activities with the smallest slack time.

Definition 4. *"Slack time" ($Slack_i$) refers to the margin that an activity i has in its planning window. It is a function of the deadline (dl_i), the earliest start time ($\underline{r}_i$), and the activity duration (D_i). We calculate it as follows:*

$$Slack_i = dl_i - \underline{r}_i - D_i \tag{6.3}$$

We define the set $Prec_i$ as the set containing all the predecessors of activity i and which is sorted according to the number of predecessors of each element in the subset. Items with the lowest number of predecessors will be at the beginning. The set DL is thus constituted as follows: $DL = \{Prec_1, Prec_2, ..., Prec_n\}$ where $Slack_1 \leq Slack_2 \leq ... \leq Slack_n$. We perform the serial scheduling of the activities contained in DL following Algorithm 1.

Scheduling Other Activities

Once planned activities with a deadline and its predecessors, we must perform the scheduling of the remaining activities (L). To choose the order in which activities will be scheduled, we propose to use the most common priority rules in the scheduling literature:

- Longest Duration (LD): prioritises the activity i with the greatest duration (D_i).
- Most Successors (MS): prioritises the activity i with the highest number of successors.
- Earliest Start Time (EST): prioritises the activity i with the lowest earliest start date ($\underset{\sim}{r}_i$).
- Earliest Finish Time (EFT): prioritises the activity i with the smallest "earliest finish time". This date is calculated by adding the duration of the activity (D_i) to the earliest start date ($\underset{\sim}{r}_i$), ie: $\underset{\sim}{r}_i + D_i$.
- Greatest Rank (GR): prioritises the activity A_i for which the sum of the duration of its successors is the largest.
- Greatest Resource Demand (GRD): prioritises the activity i with the highest resource consumption.

In order to increase the chances of finding a feasible solution from the beginning, and even improve the solution, we propose to build the set of activities to schedule L as follows: $L = \{\overline{NP}, \overline{PP}, \overline{P}\}$ where $\overline{NP}$ is the subset of non-preemptive activities, $\overline{PP}$ is the subset of partially preemptive activities and $\overline{P}$ is the subset of preemptive activities. $\overline{NP}$, $\overline{PP}$ and $\overline{P}$ are sorted according to the priority rule. With this approach, we exploit the ability of preemptive and partially preemptive activities to fill the unused spaces left after scheduling the non-preemptive activities.

The heuristic presented before is a single-pass heuristic because only one priority rule is used to select the activities to be scheduled. In order to improve the results we get, we can execute the procedure using all the activity priority rules presented before and keeping the minimum makespan, as proposed by Almeida *et al.* [10]. This process originates a so-called multi-pass heuristic.

6.3 Binary-Tree-based Local Search Algorithm

Greedy construction algorithms, like the one proposed in Section 6.2, may accept some myopic choices that lead us to local optimum, needing an additional phase were changes can be performed to ameliorate the current solution [161]. In order to improve our results, we propose to use a binary-tree-based local search algorithm partially inspired by the Limited Discrepancy Search [80].

For each sequence (priority rule) used in the greedy algorithm, there is a significant amount of possible schedules that are defined by the technician allocations we made at each step. In fact, for each period we choose to schedule an activity, there could be a large number of possible technicians allocation. The fact of choosing a specific technician may change the earliest start time of future activities since this decision modifies technician availability. Because of the combinatorial explosion, enumerate all possible solutions for the same priority rule can be prohibitive. An incomplete binary search tree maybe then interesting.

For generating this tree, we use a similar approach than in the greedy algorithm (Algorithm 1). However, now, every time we must effectuate the technician allocation (Step 3 in Algorithm 1), we generate as the left-hand branch a node representing the best allocation we get solving the MCMF with the method in Section 6.1, while as the right-hand branch we have a node representing the second best solution (this solution should not change the start time of the activity), if such solution exists. For identifying the second best solution, we solve the flow problem several times, suppressing one at a time the technicians who were selected in the initial flow solution. The solution with the lowest technician cost is kept. For non-preemptive activities, only one branching will be performed (since the flow problem must be solved only once to ensure that the same technicians execute the whole activity), while for preemptive and partially preemptive activities we must generate as many nodes as time units of duration the activity has. Figure 6.3 presents an example of a binary tree for an instance having three activities: two non-preemptive activities ($A1$ and $A3$), and a (partially) preemptive activity ($A2$) with a duration of two time units.

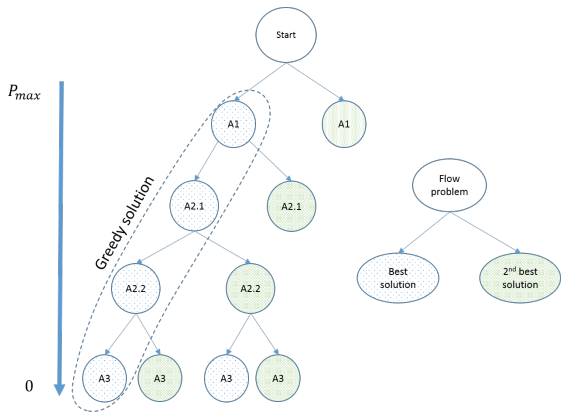

Figure 6.3: Example of a binary tree

Visiting the whole binary search tree can be still prohibitive for industrial instances (especially for instances having an important amount of preemptive and partially preemptive activities). We must limit even more the number of visited branches. From the way the solution is constructed in our greedy algorithm, we can infer that the probabil-

ity for a heuristic to make an error decreases as we add more activities to the partial schedule (going deep in the search tree); if there are fewer activities to be scheduled the criticality cost of a technician (Definition 2) is more accurate. We can then decrease the number of branches examined by giving each node a probability, decreasing according to the depth in the tree, to examine the right branch (second best answer for the MCMF). In this first version of the algorithm, we propose to use a constant gradual decrease (Δ), calculated as follows:

$$\Delta = \frac{P_{max}}{Depth_{max}} \tag{6.4}$$

In Equation 6.4, P_{max} represents the maximum probability of analysing the right branch at the top of the search tree. $Depth_{max}$ is the maximum depth of the tree. Once can reduce, even more, the number of visited branches by limiting the number of discrepancies (number of times we decide to visit the right branch) that can happen on a branch.

For exploring the search tree, we use a depth-first search approach (for avoiding memory issues), going from the left side to the right side (exploring first the answer we get using the greedy algorithm). In order to accelerate the search process, we cut all solutions (or partial solutions) that do not improve the $C_{\max}$. Every time a better $C_{\max}$ is found, the upper bound is updated. The tree-based local search procedure is presented in Algorithm 2.

Again, the proposed algorithm can be seen as a single-pass algorithm. To improve the results, we can develop its multi-pass version executing the algorithm for all the priority rules proposed in Section 6.2. To get faster results, we propose first to determinate the $C_{\max}$ for every priority list using the greedy algorithm; and after to execute the local search algorithm starting from the list with the lowest $C_{\max}$ to the one having the biggest, keeping always the best $C_{\max}$ as upper bound for cutting branches.

6.4 Greedy Randomised Adaptive Search Procedure

As stated in Chapter 2, GRASP is an iterative multi-start algorithm, in which each iteration consists of two phases: generation and local search. In the construction phase, a feasible solution is generated; then its neighbourhood is explored by the local search algorithm until a local optimum is found. The best solution found overall GRASP iterations is kept as the results. An extensive survey of GRASP and its applications are presented in [152]. In the following of this section, we describe the GRASP algorithm we propose to solve the MSPSP-PP. This heuristic was developed in collaboration with Prof. Lars Mönch.

Generation and local search phase

For the generation phase, we use a modified version of our greedy algorithm (Algorithm 1). Now instead of following a fixed priority list at each greedy iteration, we

Algorithm 2: Tree-based local search algorithm

```
Define: P(Node) is the probability of visiting the right branch at the Node;
Initialize Node to the first activity from the list and set current time to its
  release date;
while Node ≠ root do
    Identify the periods where the activity can be scheduled;
    if periods exit then
        if Left Branch Not Visited (Node) then
            Solve MCMF;
            Assign the technicians for the current activity at current time
              according to the best solution and update Current Cmax;
            Go to next node;
            // This next time unit of duration if preemptive or
               partially preemptive, next activity if non-preemptive
        else
            p ← random(0, 1);
            if (Right Branch Not Visited (Node)) & (p ≤ P(Node)) &
              (Alternative solution exists) & (Number of discrepancies <
              discrepancy limit) then
                Assign the technicians for the current activity at current time
                  according to the second best solution and update
                  Current Cmax;
                Go to next node;
            else
                Backtrack;
            end
            if Current Cmax ≥ Best Cmax then
                Backtrack;
            end
            if (Node is a leaf) & (Current Cmax < Best Cmax) then
                Update Best Cmax;
                Backtrack;
            end
        end
    else
        Backtrack;
    end
end
```

select an activity randomly from a restricted candidate list (RCL), we then identify the earliest periods when this activity can be scheduled (according to its preemption type and resources and technicians availability), and we finally allocate the technicians to the activity solving a Minimum-Cost Maximum-Flow (MCMF) problem.

Let $F(i)$ be an adaptive evaluation function, which indicates the degree of relevance of planning activity i in the current greedy iteration. Let also define M as the number of not yet scheduled activities after the current greedy iteration. The RCL is made up of the $1 + \alpha * M$, $\alpha \in [0, 1]$, activities having the best F values. Note that the RCL can only contain activities such that all their predecessors has already been scheduled.Each element within the RCL has a probability of being chosen (π_i) defined as follows:

$$\pi_i = \frac{F(i)}{\sum_{j \in RCL} F(j)} \ . \tag{6.5}$$

For choosing the value of α, we propose to use the reactive strategy proposed by Praias and Ribeiro [147], where the value of α is randomly selected from a discrete set $\Psi = \{\alpha_1, ..., \alpha_n\}$ of possible α values. The probabilities associated with the choice of each value are all initially uniformly distributed. After a few iterations, they are periodically reevaluated, taking into consideration the quality of the obtained solution for each $\alpha_k \in \Psi$. In our case, we use the $C_{\max}$ as a quality indicator.

Once a valid initial solution is found (because of the presence of time windows, not all GRASP iterations will found a valid solution), one explores its neighbourhood using the tree-based local search algorithm proposed in Section 6.3 (Algorithm 2).

Adaptive greedy evaluation function

The proposed adaptive greedy evaluation function has three components: priority rule ($L(i)$), intensification ($I(i)$) and feasibility ($G(i)$). It is defined as follows:

$$F(i) = \beta * L(i) + \delta * I(i) + \gamma * G(i) \ . \tag{6.6}$$

Priority due to a priority rule ($L(i)$): Computational experiments, presented in [145], suggest that using the greedy algorithm with priority rules "Most Successors", "Greatest Rank" and "Longest Duration" provides smaller optimality gaps. Let Sc_i be the set of successors of activity i. We can then define a priority function mixing these rules as follows:

$$L(i) = D_i + \sum_{j \in Sc_i} D_j \ . \tag{6.7}$$

Intensification component ($I(i)$): The idea is to use the characteristics of a set ε of q elite solutions to influence the construction phase. In our algorithm, the quality of the solution is highly dependent on the order (Seq_k) in which activities have been treated by the greedy algorithm to obtain the solution k. Let define $Bef(k, i, l)$ as a

binary function taking the value of 1 is activity i was treated before activity l in the Seq_k, 0 otherwise. Let NS be the set of not yet scheduled activities. The intensification component is defined as follows:

$$I(i) = \sum_{k \in \varepsilon} \sum_{l \in NS} Bef(k, i, l) \ . \tag{6.8}$$

Feasibility factor ($G(i)$): As it has been made clear from the beginning of this chapter, time windows make it difficult to find feasible solutions with the greedy algorithm. We propose then to introduce a component giving priority to activities with a short slack time, this will allow us to increase our chances of finding valid initial solutions faster. Slack time ($Slack_i$) refers to the margin that an activity i has in its planning window (Definition 4). The feasibility factor is defined as follows:

$$G(i) = \frac{1}{dl_i - \underline{r}_i - d_i} \ . \tag{6.9}$$

Note that $L(i)$, $I(i)$ and $G(i)$ must be normalised before taking the weighted sum. Moreover, we have β, δ, $\gamma \in [0, 1]$ and $\beta+\delta+\gamma = 1$. Parameters δ and γ are self-adaptive, they are equal initially, and their values are periodically updated. If after N GRASP iterations the number of infeasible solutions increases, we must increase the value of γ; on the contrary, if this number decreases, we decrease γ (to try more diverse solutions). On the other hand, the parameter δ decreases when the diversity of obtained solutions is too low and increases when the variability is high.

For measuring the variability (diversity) of the solutions, we keep using the sequences used to generate them as reference. Let us define LSe as the set of the last sequences that generated valid solutions during the last GRASP iterations, and $Nb(i, j, LSe)$ as the number of sequences $k \in LSe$ where activity i was handled before activity j by the greedy algorithm. $Seq_{k,v}$ represents the $v-$th element (activity) of the sequence k. For each sequence, we can define a similarity index ($Sim(k, LSe)$), very close to the Kendall tau distance [94], indicating the degree on which a sequence k shares the same characteristics of the other sequence $\in LSe$, as follows:

$$Sim(k, LSe) = \sum_{v=1}^{card(Seq_k)-1} \sum_{u=v}^{card(Seq_k)} Nb(Seq_{k,v}, Seq_{k,u}, LSe) \tag{6.10}$$

A decrease on the average value of $Sim(k, LSe)$ after N iterations indicates a better diversity on the solutions.

Updating the elite solutions set

A warm-up phase is necessary to be able to constitute the initial set of elite solutions (i.e. the sequences that generate these solutions). During this warming phase, all sequences,

regardless of the quality of the solutions they generate, will be included in the elite solution set until complete the number of elite solutions; the only condition to include sequences in ε is that all sequences must be different. Note that during this warming phase $F(i) = \beta * L(i) + \gamma * G(i)$ $(\beta + \gamma = 1)$. α and γ will self-adapt according to the solution quality and the number of infeasible solutions respectively.

Once the warming phase is ended, the set of elite solution is updated following the quality of solutions and their similarity indicator. If a new sequence generates a solution with a $C_{\max}$ lower than the worst $C_{\max}$ in the elite solutions, this new sequence is included in ε. The sequence with the worst $C_{\max}$ is then deleted from the elite set. In case several sequences have the worst $C_{\max}$, we delete the sequence with the highest $Sim(k, \varepsilon)$ value (to improve the diversity within the elite solution). If the new solution generates a solution with a $C_{\max}$ equal to the worst $C_{\max}$ in ε, we will include the new sequence only if it has a lower $Sim(k, \varepsilon)$ value than the sequences with the worst $C_{\max}$ in the elite set (the sequence with the highest similarity indicator value is deleted).

Algorithm 3 summarises the proposed greedy randomised adaptive search procedure for the MSPSP-PP.

6.5 Large neighbourhood search

In this section, we present a Large Neighbourhood Search (LNS) algorithm with subproblem exact resolution inspired by the work proposed by Palpant *et al.* [132] for the RCPSP. At each iteration, starting from a feasible initial solution, the method fixes a subpart of the current solution, while the other part is solved using an exact method.

In our algorithm, we start with an initial solution obtained by the greedy algorithm (Section 6.2). For generating the subproblem to be solved, we define a sliding *time window* with a fixed length, that is a function of the average duration of the activities (various length are tested for the computational experiments), and will be shifted to the right at each iteration of the method (the first time window starts at $t = 0$. For the following, it starts in the middle of the previous one, see Figure 6.4). At each iteration of the algorithm, all the activities within this time window are selected (according to their preemption type) to be rescheduled solving a MILP or CP model (models presented in Chapter 5). If, after solving the subproblem, we obtain a solution that can lead to improvement (at least the finish time of one of the activities within the time window has decreased), we keep the subproblem solution and insert it to the global problem solution. Using the greedy algorithm, we try to improve the scheduling of activities to the right of the time windows. Once these operations are performed, the time window is shifted to the right, and the process is repeated. The heuristic stops when the time window reaches the $C_{\max}$ of the current solution.

As a modification, a multi-sweep version of the algorithm can be done. After reaching the $C_{\max}$ of the current solution, the sliding time window is returned to the period t where the first change between the initial solution and the new one occurs. The time

Algorithm 3: GRASP for the MSPSP-PP

```
Gite ← 0 // Valid GRASP iteration counter parameters
MaxIterations// Maximum number of iterations
while Gite ≤ MaxIterations do
    n ← n + 1 // Iterations counter for parameters
    // Generate initial solution
    Chose randomly α value;
    while (Not all activities are scheduled)&(Schedule is still feasible) do
        Generate RCL;
        Randomly choose an activity from RCL;
        Schedule activity as early as possible;
    end
    if Feasible solution then
        Execute local search (Algorithm 2);
        Update ε;
        Gite ← Gite + 1 // Valid GRASP iterations counter
    else
        fail ← fail + 1 // Fails counter
    end
    // Update α, β, δ and γ
    if n = N then
        Update the probability of each α;
        Calculate average Sim(k, LSe);
        if average Sim(k, LSe) < previous one then
            Increase δ // We propose to use 0.1 steps
        else
            Decrease δ ;
        end
        if fail < previous one then
            Decrease γ;
        else
            Increase γ ;
        end
        Adjust β value // Ensure that β + δ + γ = 1
        fail ← 0;
        n ← 0
    end
end
```

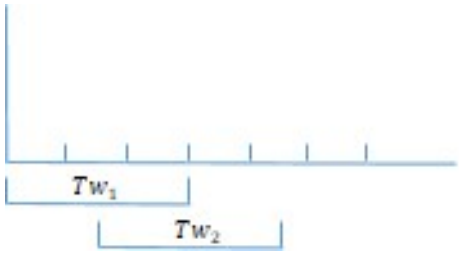

Figure 6.4: Time window right shift

window starts to slide again until it reaches the $C_{\max}$. The multi-sweep version stops after two iterations without improving the $C_{\max}$. Algorithm 4 summarises the proposed method.

Algorithm 4: LNS for the MSPSP-PP

```
Generate initial solution;
Improvement ← True;
while Improvement do
    Select activities for the subproblem;
    Construct subproblem;
    Solve subproblem // using MILP or CP
    if Subproblem solution is improved then
        Include subproblem solution in the global solution;
        Improve the schedule to the right of the current time window // using
            greedy algorithm
    end
    if Cmax is inside the current time window then
        if Current Cmax is equal than previous want then
            Improvement ← False;
        else
            Return time window to t where first change happened
        end
    else
        Shift time window to the right;
    end
end
```

Selection of activities

To select the activities to include in the subproblem, we must look at their preemption type:

- For *non-preemptive* and *partially preemptive* activities, we include all activities for which the interval [start, end] of the activity overlaps the time window.

- For *preemptive* activities, we only include the time units of the activity that are executed within the time window.

Solving the subproblem

The scheduling horizon of the subproblem goes from the earliest start time to the latest end time of selected activities. The fixed activities will be reflected in the resources and technicians availability. We use the models presented in Chapter 5. However, a modification must be done over the objective function. Now, we must try to minimise the average end time of activities (except when time windows reach the $C_{\max}$ of the current solution, in this case, the objective function is still the minimisation of $C_{\max}$). The new objective function is:

$$minimise : \sum_i end_i \tag{6.11}$$

6.6 Computational experiments

For testing the performance of the heuristic methods, we generated new sets of instances (A2, B2, C2, D2). For each instance on a set, there is an instance on the other sets having the same characteristics, except for the distribution of preemption type for the activities (see Table 6.1). Each of the four sets has a total of 50 instances, each of them with 50 activities to be scheduled, and an expected $C_{\max}$ going from 130 to 170 time units. The average duration of the activities goes from 5 up to 15 time units; they may require up to 15 skills and up to 8 cumulative resources. 20% of the activities are subject to time windows. A total of 8 technicians (multi-skilled resources) are available in 2 teams (out of 4 each) doing work-shifts of 12 hours. All other characteristics were generated randomly. We decide to use instances with only 50 activities, instead of 100 activities that is in average the number of activities scheduled at the LECA-STAR every week, to have better lower bounds for evaluating the optimality gap of the proposed heuristics. Indeed the solvers (MILP and CP) may have some issues for finding good lower bounds for larger instances.

	Set A2	Set B2	Set C2	Set D2
Non-preemptive	10%	10%	80%	33.3%
Partially preemptive	10%	80%	10%	33.3%
Preemptive	80%	10%	10%	33.3%

Table 6.1: Distribution of preemption types for instances of the MSPSP-PP

The proposed heuristics has been coded in C++. To solve the flow problems, we used the adapted C++ version of the Edmonds-Karp algorithm proposed by Ababei [2]. To obtain the lower bound or, in some cases, the optimal solutions, we use the MILP and CP models proposed in Chapter 5, which were solved using CPLEX 12.7.1 and CP Optimizer 12.7.1. All computational tests have been carried out using a machine under Ubuntu 16.04.6 operating system, equipped with an Intel Xeon E5-2695 processor running at 2.3 GHz.

Greedy algorithm

Table 6.2 shows the average gap values (percentage difference between the obtained solution an the best known lower bound) for the greedy algorithm using the priority rules presented in Section 6.2. It also shows the average gap for the CP model after 5 minutes of computation using CP Optimizer with only one thread. Results obtained with the MILP model are not given since the solver run out of memory before giving any initial solution. We observe that the heuristic using Greatest Resource Demand (GRD) as priority rule seems to give the lower average gap, followed by the priority rules Longest Duration (LD) and Most Successors (MS). The worst results are obtained using the Earliest Start Time (EST) and Earliest Finish Time (EFT).

	Gap for the greedy algorithm				
	Set A2	Set B2	Set C2	Set D2	All
LD	7.33%	7.78%	15.65%	9.28%	10.01%
MS	8.44%	8.26%	16.85%	9.27%	10.70%
EST	9.61%	9.99%	18.98%	10.00%	12.14%
EFT	10.68%	10.79%	22.72%	10.48%	13.67%
GR	8.69%	8.49%	16.66%	9.28%	10.78%
GRD	7.33%	7.88%	15.90%	8.02%	9.78%
Multi-pass	5.51%	6.14%	12.79%	6.51%	7.74%
CP (after 5 min)	6.01%	6.65%	7.65%	5.56%	6.47%

Table 6.2: Average gap for the greedy algorithm per priority rule

The results of the Wilcoxon signed-rank test [112] for the equality of gap by priority list are presented in Table 6.3. Any *p-val* higher or equal than 0.05 indicates that the average gaps are statistically equal. We observe that LD and GRD priority rules have a statistically equal average gap, and are statistically better than all the other priority rules. MS and GR priority rules also have statistically equal average gap and outperform EST and EFT priority rules. Statistical tests prove that EFT is the worst priority rule in these numerical tests.

If we compare the results of the multi-pass version of the greedy algorithm against the results obtained after 5 minutes of computing of the CP model, we see that the average gap obtained by CP is slightly lower than the one obtained by the greedy algorithm.

	p-values for gap equality test				
	MS	**EST**	**EFT**	**GR**	**GRD**
LD	0.015	0.000	0.000	0.012	0.263
MS	-	0.002	0.000	0.257	0.001
EST	-	-	0.000	0.003	0.000
EFT	-	-	-	0.000	0.000
GR	-	-	-	-	0.001

Table 6.3: Wilcoxon signed-rank test for gap equality by priority list

However, if we analyse the results for each set of instances, we observe that the greedy algorithm gets a lower average gap for instances with a low proportion of non-preemptive activities (Sets A2 and B2). Statistical tests (Table 6.4) indicate that the CP model (after 5 min) outperforms the greedy algorithm only when the proportion of non-preemptive activity is high (Set C2). Note that the computation time for obtaining the greedy solution is lower than one second, which proves the interest of the greedy algorithm.

	p-values for gap equality test				
	Set A2	Set B2	Set C2	Set D2	All
p-val	0.255	0.421	0.000	0.08	0.000

Table 6.4: Wilcoxon test for equality between greedy algorithm and CP

Tree-based local search

The probability of visiting the right-hand branch (P_{max}) is the main parameter of the proposed algorithm since it plays a role over the number of visited branches, and thus over the execution time (time required to visit the generated tree) and the solution quality. We tested the tree-based local search algorithm (multi-pass version) with different values of P_{max} ($5\%, 10\%$ and 15%) to study the behaviour of the execution time and the quality solution (measured by the average gap). We arbitrarily set the maximum number of discrepancies by branches to 5. Results are presented in Figure 6.5 for all instances, and in Figure 6.6 for each set of instances.

The gap and time evolution charts (Figure 6.5 and Figure 6.6) show an exponential increase of the average execution time when the value of P_{max} increases. The average gap, on the other hand, seems to decrease following a linear evolution. If we take the results for $P_{max} = 10\%$ (the best compromise between solving time and average gap) as a reference, we observe that the local search algorithm beats CP results for sets A2, B2 and D2. The worst average gap is still for the set C2. We observe, however, that average execution time required for instances of set C2 is significantly low. This is normal since few nodes (and thus few branches) are generated for instances with a high proportion of non-preemptive activities. One could improve the results for set C2 by giving a higher

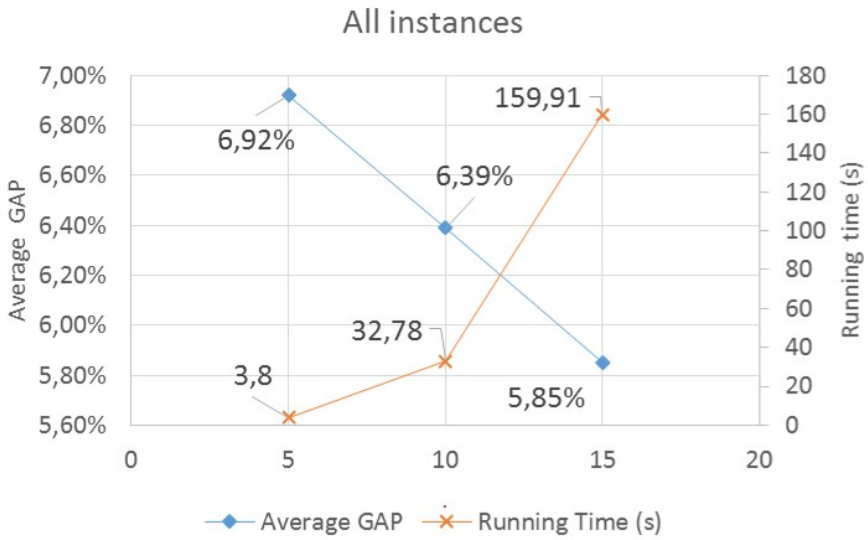

Figure 6.5: Gap and time evolution in function of P_{max} for all instances

value of P_{max}. That is why we propose to adapt the value of this parameter using a function that depends on the number of non-preemptive activities:

$$P_{max} = prob_{low} * \exp\left(\frac{\ln(prob_{high}) - \ln(prob_{low})}{cardinality(\overline{I})} * cardinality(\overline{NP})\right) \tag{6.12}$$

In equation 6.12, $prob_{low}$ and $prob_{high}$ indicates the minimum and the maximum value that the initial probability can get. We tested this self-adaptive version set the values of $prob_{low}$ and $prob_{high}$ to 10% and 80% respectively. Table 6.5 presents the results. We observe that this configuration allows obtaining less variable average execution times for all sets of instances. The results for sets A2, B2 and D2, outperform those obtained by the CP solver after five minutes. Increasing the value of P_{max} for the instances of set C2 improve the average gap; however, this improvement is not enough to outperform CP.

	Average gap	Average execution time
Set A2	4.03%	89.89 s
Set B2	4.78%	160.35 s
Set C2	8.77%	115.10 s
Set D2	4.30%	193.88 s
All	5.46%	139.80 s

Table 6.5: Results for local search algorithm with self-adaptive P_{max}

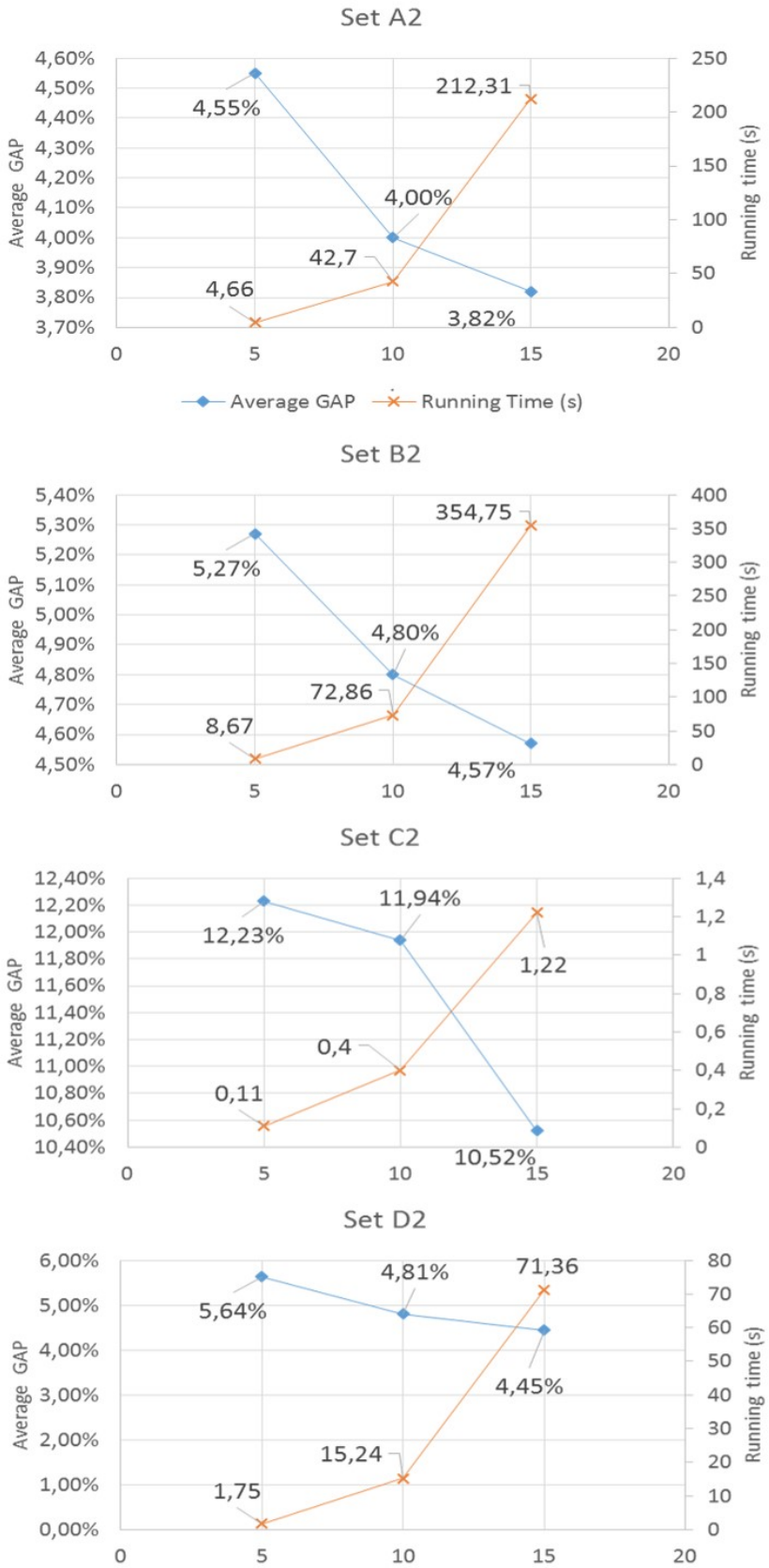

Figure 6.6: Gap and time evolution in function of P_{max} by preemption type

GRASP

The grasp algorithm was tested over the set of 200 instances. We fixed the size of the elite solution set to 20 solutions, and the maximum number of valid GRASP iteration to 550. To analyse the impact of the intensification component, we tested a basic GRASP version where this component is not taken into account, and let it run also for a maximum of 550 valid GRASP iteration. For the local search phase, we used the self-adaptive version presented in the previous section, fixing $prob_{low}$ and $prob_{high}$ to 5% and 60%, respectively. The number of discrepancies by branch was limited to 1. Results are presented in Table 6.6.

	Complete GRASP		Basic GRASP	
	Average gap	Average execution time	Average gap	Average execution time
Set A2	2.21%	77.76 s	2.32%	73.31 s
Set B2	2.51%	88.75 s	2.77%	91.34 s
Set C2	8.26%	37.53 s	8.63%	35.94 s
Set D2	2.98%	71.22 s	3.30%	68.02 s
All	3.99%	68.81 s	4.26%	67.15 s

Table 6.6: Results for GRASP algorithm

We observe that the GRASP with intensification component (complete GRASP) gives a lower average gap for all sets of instances. Statistical tests (Table 6.7) show that there is enough evidence to say that the intensification component improved the results obtained for sets B2 and D2. For sets A2 and C2, the GRASP with intensification is in the worst case equal to its version without intensification. We can conclude then that the intensification component helps to improve the quality of final solutions. Both configurations outperform the CP results for sets A2, B2 and D2. CP remains better when the proportion of non-preemptive activities is high (set C2).

	Means equality test	
	p-val	Conclusion
Set A2	0.246	Equal
Set B2	0.026	Different
Set C2	0.1129	Equal
Set D2	0.017	Different
All	0.001	Different

Table 6.7: Statistical test for improvement due to the intensification component

LNS

We tested our algorithm using different fixed lengths for the sliding time window, all of them in function of the average duration ($\overline{D}$) of activities within the instances: $0.5 * \overline{D}$, $\overline{D}$ and $1.5 * \overline{D}$. The first issue we want to study is the interest of using CP or MILP to solve the optimisation subproblem; we then tested the single-pass version of the heuristic using both techniques. Note that we limited the number of threads used by CPLEX and CP Optimizer to 1, and the maximum time expended on each optimisation subproblem is 30 s. The maximum computation time allowed to the whole heuristic is 5 min. We use as an initial solution the best solution obtained by the multi-pass version of the greedy algorithm. Results are presented in Table 6.8.

	MILP						CP					
	$0.5 * \overline{D}$		$\overline{D}$		$1.5 * \overline{D}$		$0.5 * \overline{D}$		$\overline{D}$		$1.5 * \overline{D}$	
	Average gap	Average time	Average gap	Average time	Average gap	Average time	Average gap	Average time	Average gap	Average time	Average gap	Average time
Set A2	2.34%	23.36 s	2.18%	49.69 s	1.97%	164.9 s	3.47%	234.08 s	3.39%	311.09 s	3.45%	310.46 s
Set B2	2.62%	231.77 s	3.05%	286.28 s	3.56%	312.54 s	4.69%	310.95 s	4.55%	308.33 s	4.60%	305.13 s
Set C2	7.93%	56.95 s	7.85%	80.92 s	7.53%	173.17 s	8.20%	88.42 s	8.17%	80.48 s	7.95%	109.59 s
Set D2	2.96%	102.12 s	2.87%	179.40 s	3.28%	281.73 s	4.32%	311.03 s	3.95%	312.93 s	3.94%	311.65 s
All	3.97%	103.56 s	3.99 %	149.07 s	4.09%	233.14 s	5.18%	236.12 s	5.02%	253.21 s	4.99%	259.21 s

Table 6.8: Results for single sweep LNS using MILP and CP

We observe that the heuristic using MILP always gives a smaller average gap. It is also, most of the time, faster than the heuristic using CP. The analysis of results of all instances does not allow to determine whether the length of the sliding time windows has an impact over the quality of the final solution. However, if we analyse the results for each set of instances, we observe that for sets A2 and C2 (i.e. instances with a low proportion of partially preemptive activities) a bigger length of the sliding time window reduces the final average gap for the MILP heuristic. On the other hand, if the proportion of partially preemptive activities is high (set B2), the MILP heuristic gives better results when the length of the time window is smaller.

Table 6.9 presents the results for the multi-sweep version of the heuristic. As expected, there is a small reduction of the average gap for all sets of instances and all time windows length. For set A2, we observe a similar behaviour to the one observed during the single-sweep version: larger time windows generate lower gaps. Instances from set B2 have lower average gaps when time windows are small.

A multi-start version of the algorithm was also tested. This time, we process the LNS (using the single-sweep MILP heuristic) over six initial solutions generated by using the greedy algorithm and priority rules proposed in Section 6.2. The best solution over all LNS iterations is kept. We used the fixed length of the time windows equal to $0.5 * \overline{D}$ (since it gives the faster results), and limited the maximum time expanded to solve the optimisation subproblem to 15 s. Each iteration of the LNS is limited to 1 minute. The results presented in Table 6.10 show that the proposed multi-start algorithm is not better than the complete execution over the best greedy solution (see Table 6.8), since

	MILP					
	$0.5 * \overline{D}$		$\overline{D}$		$1.5 * \overline{D}$	
	Average gap	Average time	Average gap	Average time	Average gap	Average time
Set A2	2.13%	47.59 s	1.83%	151.43 s	1.78%	296.94 s
Set B2	2.44%	287.51 s	3.01%	312.89 s	3.56%	312.93 s
Set C2	7.34%	99.26 s	7.67%	159.04 s	7.16 %	279.21 s
Set D2	2.39%	191.59 s	2.64%	270.46 s	3.24%	310.87 s
All	3.58%	156.65 s	3.79%	223.46 s	3.94%	299.99 s

Table 6.9: Results for multi-sweep LNS using MILP

the average gaps are statistically equal, and the multi-start version requires a higher average execution time.

	Average gap	Average time
Set A2	2.23%	129.09 s
Set B2	3.51%	395.24 s
Set C2	7.41%	233.01 s
Set D2	3.12%	345.83 s
All	4.07%	275.79 s

Table 6.10: Results for multi-start LNS

From the results of the multi-start version, one can see that it is better to use a complete LNS over a solution of good quality (rather than executing limited multi-start). We can then improve the results obtained by the GRASP algorithm by executing an LNS iteration over the best solution. We execute first the complete version of the GRASP algorithm using the configuration described before. The LNS algorithm (its MILP version) then improves the obtained solution on a single sweep. We use a fixed time window length of $0.5 * \overline{D}$. Results are presented in Table 6.11. We observe that combining GRASP and LNS improves the quality of the obtained solutions significantly. This combination outperforms the solutions obtained by the CP solver within a limited time (5 min). Similar to the other proposed heuristics, this algorithm performs better when the proportion of non-preemptive activities is low.

6.7 Concluding remarks

In this chapter, we presented various heuristic methods for solving the MSPSP-PP. Initially, a greedy algorithm is proposed. At each iteration, the greedy algorithm decomposes the MSPSP-PP into two subproblems: scheduling and technician allocation. For the scheduling part, a priority rule is used. The allocation subproblem is solved

	Average gap	Average time
Set A2	1.56%	87.29 s
Set B2	1.79%	279.56 s
Set C2	6.68%	82.65 s
Set D2	2.06%	147.97 s
All	3.02%	149.37 s

Table 6.11: Results for LNS after GRASP

using a Minimum-Cost Maximum Flow (MCMF) problem. Computational tests show that the greedy algorithm allows obtaining solutions (in time lower than 1 second) that are near to the ones obtained by the CP solver after 5 min of computing.

A tree-based local search algorithm, partially inspired by the Limited Discrepancy Search, was also proposed to improve the solutions obtained by the greedy algorithm. A self-adaptive version of the local search algorithm allows obtaining solutions with an average gap of 5.46%.

Then, a greedy randomised adaptive search procedure (GRASP), combining the greedy and local search algorithms, has been introduced. The proposed GRASP includes some components looking for an improvement of the solution feasibility, and the use of characteristics of best-found solutions to bias the generation of new solutions (intensification). The GRASP got good results during the computational test, being able to obtain solutions with an average gap of only 3.99%. The use of intensification in the GRASP showed an improvement of the quality of solutions when the proportion of partially preemptive activities is high. The GRASP method is the one giving the best tradeoff between execution time and solution quality.

Finally, a Large Neighbourhood Search algorithm was also proposed. The LNS algorithm mixes the greedy algorithm with the exact resolution of an optimisation subproblem. Computational tests showed that the use of MILP for solving the optimisation sub-problem allows obtaining better solutions faster (compared with the use of CP for the subproblem exact resolution). The LNS method allows getting solutions with an average gap of 3.58%. Executing LNS after GRASP allows to get the better results (average gap of 3.02%), largely beating the results of the CP solver (with time limited to 5 min).

All proposed methods are negatively affected by the presence of a high proportion of non-preemptive activities (obtained the higher average gaps). Further research needs to be done in order to improve the quality of solutions when this proportion is high. Special attention must be given to the study of a better local search algorithm since it has an important role in all the proposed methods. The local search algorithm proposed in this chapter has an exponential time complexity. Additionally, we can not guarantee that it always find the local optimum of the neighbour.

We now have the models and solution methods to process the scheduling of activities within the LECA-STAR laboratory. We will discuss in Chapter 7 about their implantation in the facility.

Part III

Industrial Experience

Chapter 7

Industrial application

Contents

7.1 Scheduling engine – GUI 113

7.1.1 Data entry 114

7.1.2 Schedule generation 116

7.1.3 Schedule display 117

7.1.4 GUI tests 117

7.2 Implementation of the management information system 119

7.3 Concluding remarks 120

After proposing the models, developing the algorithms and validating them on representative but limited data sets of the case study, the objective is now to develop a prototype of a decision support system in line with the LECA-STAR real operations activities. In this chapter, we present the first tests of the scheduling engine and the management information system. In Section 7.1, we describe a prototype Graphic User Interface (GUI) for the scheduling engine, and we also briefly discuss the results after testing the engine with a subset of activities at the LECA-STAR. The analysis of the feedback from the first test of the management information system is presented in Section 7.2. We also present in this section some ways of improvement to facilitate the implementation of the information system in the near future.

7.1 Scheduling engine – GUI

A Graphic User Interface (GUI) was developed in order to confirm the interest and validity of the models presented in Part II for scheduling the real activities at the LECA-STAR. The idea was to develop a standalone version of the scheduling engine. This engine should allow the entry of scheduling requests, translate this information into the format used by the scheduling algorithms, and finally present the obtained schedules on Gantt charts. In other words, this GUI should allow the planning team, who are not familiar with combinatorial optimisation methods, to communicate easily with the scheduling algorithms of Part II.

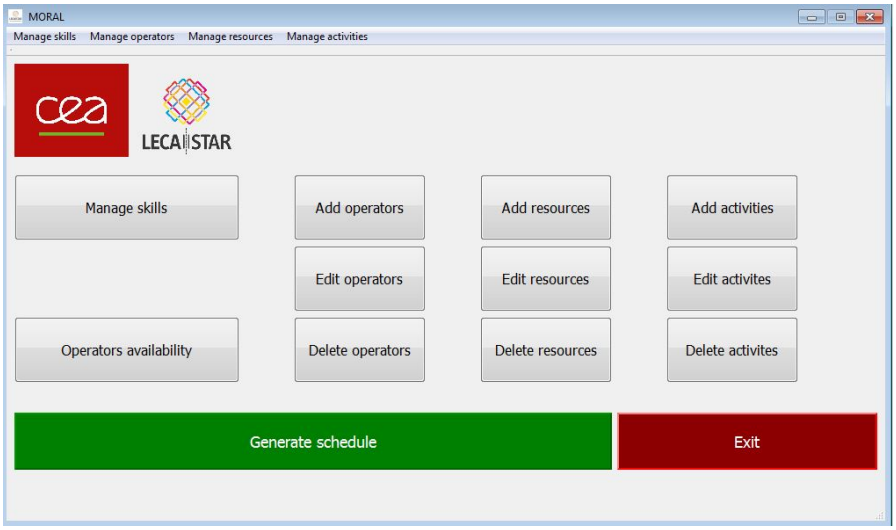

Figure 7.1: Scheduling engine GUI - main window

7.1.1 Data entry

To work, the scheduling models and algorithms proposed in this dissertation require at least the information presented in Sections 4.2 and 5.2. The GUI allows the user to interactively enter all the required information: technicians availability and skills, resources capacity, and activities to be scheduled. For each technician (Figure 7.2), one can indicate the set of skills it masters and also select its availability periods, hour by hour. In the same way, for each resource, one can indicate the evolution of the available capacity over time (Figure 7.3). Note that for uncountable resources, such as electricity and general ventilation system, the user must indicate a capacity huge enough to handle all the activities at the same time during the presence of these resources, and zero during their absence.

To enter the activity scheduling requests, the user must indicate in the activities edition window (Figure 7.4) all the characteristics of the activity: name, duration (in hours), resources and skills needs, set of preemptive resources, precedence relationships, release times and deadlines. Since all the methods presented in this dissertation work with discrete time, we decide to use the hour as the discrete time unit.

Note that, additionally to the release time and deadline, the GUI allows the user to indicate some forbidden/allowed scheduling periods for the activities (e.g. an activity that can only be scheduled in the morning, see Figure 7.5). The models proposed in Chapter 5 and Chapter 6 do not take into account directly this kind of constraints. To answer this industrial need, every time an activity requires to be scheduled at particular periods, the GUI automatically generates a fictitious resource having an availability profile corresponding to the scheduling periods (i.e. the resources will be available only during the indicated scheduling periods). The fictitious resource requirement is also

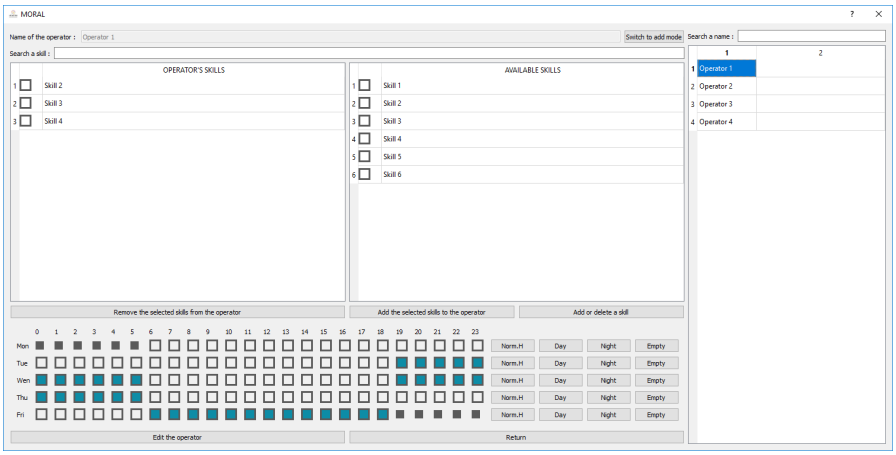

Figure 7.2: Technicians data management

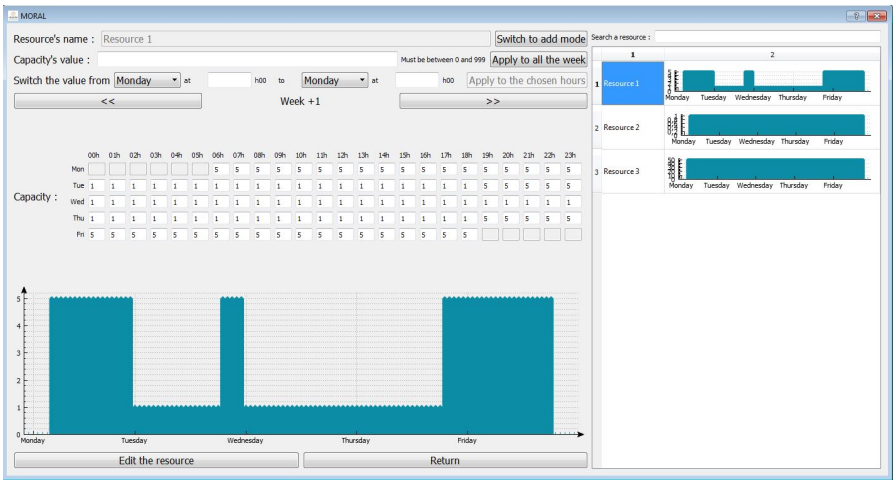

Figure 7.3: Resources data management

automatically generated. The GUI handles all these operations without showing the fictitious resources to the user. A verification phase is processed to ensure that a fictitious resource with the same availability profile is not generated twice. We decide to use this approach since it did not require modifications over the already existing scheduling algorithms. However, another way to handle this constraint is by allowing the GUI to automatically add the constraints to the models, by fixing the value for the forbidden periods.

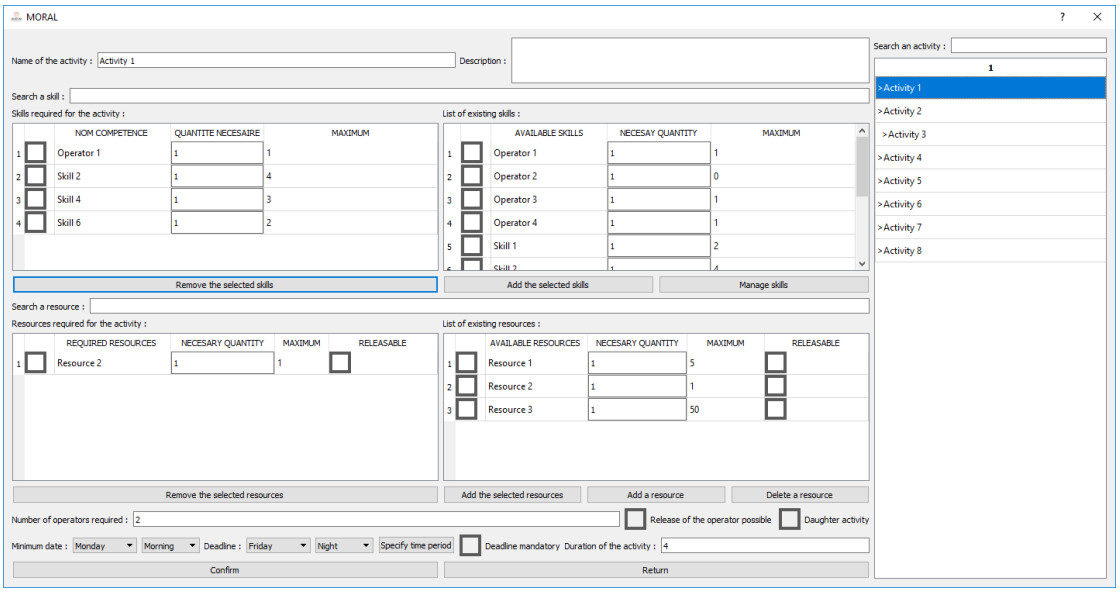

Figure 7.4: Activities requests

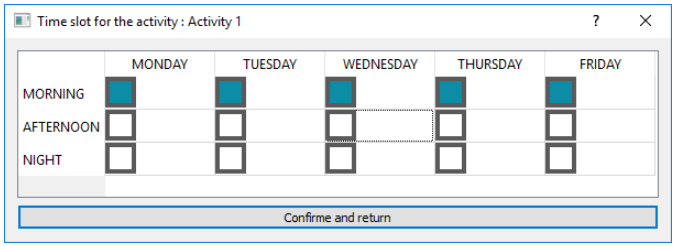

Figure 7.5: Indicating forbidden/allowed scheduling periods

7.1.2 Schedule generation

For generating the schedule, we use the models and methods proposed in Chapter 5 and Chapter 6 for the MSPSP-PP. Thus, after indicating all the required information, the user can choose the algorithm he wants to use for generating the schedule. To simplify

the choice, we limit the number of options. The user must first indicate if he wants to use an exact or a heuristic method.

If the user decided to use an exact method, he must then choose between using MILP or CP for generating the schedule. For the MILP model, we decided to use Model MSPP2b from Chapter 5.1 since it seemed to be a little faster to prove the optimality during computational tests in Section 5.4. Regardless of the selected technique (MILP or CP), the user must also indicate a maximum computation time. After validating his selection, the GUI will first generate an initial solution using the multi-pass greedy algorithm presented in Section 6.2. This initial solution will be used as a warm start for the MILP or CP solver.

On the other hand, if the user wants to use a heuristic method, he must select between two options: "*basic heuristic*" or "*heuristic + refinement*". The first option will launch the GRASP algorithm (Section 6.4). The second choice will execute a sweep of the LNS algorithm (Section 6.5) over a solution generated by the GRASP algorithm.

Even if the objective to schedule all the activities during the week (the equivalent of 108 working hours at the LECA-STAR), this is not always possible. That is why we give an additional slack to the scheduling horizon, allowing the scheduling algorithms to schedule activities even after the end of the week. If an activity is scheduled or partially scheduled after the end of the week, the GUI informs this situation to the user and then shows the schedule.

7.1.3 Schedule display

Once the scheduled is generated, the GUI shows the corresponding Gantt chart of the obtained solution (Figure 7.6). This chart indicates the periods of execution of each activity, as well as the name of the technicians that must execute each part of the activity.

Additionally, to have a better view of the work of each technician, the GUI also gives the user the option of generating a Gantt chart showing the schedule of each of them. Finally, in order to facilitate the exploitation of the resulting schedule, the GUI allows its exportation to the Microsoft Excel format used currently at the research facility.

7.1.4 GUI tests

For testing the scheduling engine, we decided to use the historical information of the schedules corresponding to five weeks of a subset of activities at the LECA-STAR (activities carried out by one of the five research teams in the facility, the LEPC). The LEPC team is the larger team at the LECA-STAR (in the number of technicians and activities carried out), and it conducts activities such as spent fuel processing and reconditioning, fuel rod re-manufacturing, fuel element puncturing, and other non-destructive examination activities for irradiated fuels and materials.

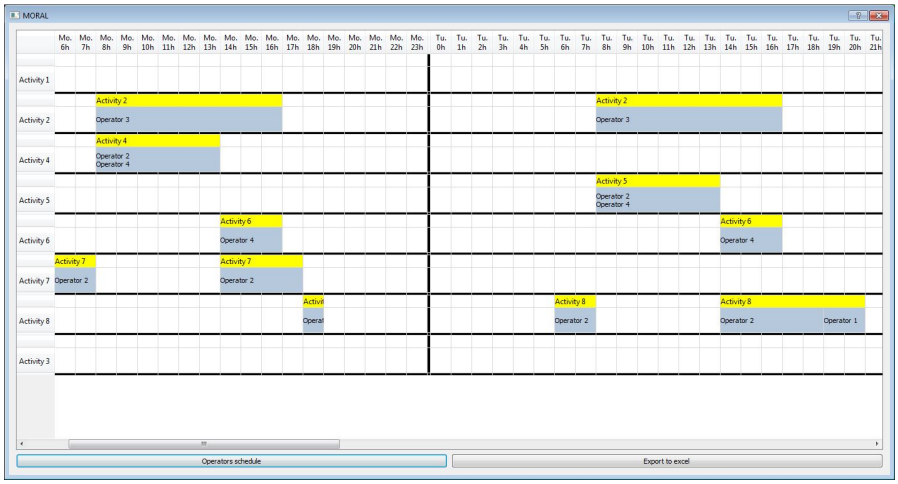

Figure 7.6: Gantt chart for a generated schedule

The LEPC has a total of 16 technicians, who are divided into four teams of 4 technicians each. Only one team is available at a time, and they do work shifts of twelve hours, what allows to work continuously from 06:00 a.m. on Monday until 18:00 on Friday (a total of 108 working hours). The LEPC executes its activities on seven hot cells. Taking into account the hot cells, we could identify a total of 12 resources required for executing the LEPC activities. Note that the list of resources is not exhaustive since we only listed the resources required during the five weeks we analysed. For the skills requirements, we have identified a total of 13 skills. Again, this is not an exhaustive list, and it is limited to the skills required by the activities executed during the studied five weeks. Table 7.1 presents the number of activities to be scheduled, the number of activities according to the preemption type, the sum of the duration of all the activities, and the improvement obtained by the scheduling engine.

Week	Number of activities	Activities declared as non-preemptive	Activities declared as partially preemptive	Activities declared as preemptive	Sum of activities duration	Makespan improvement
1	35	11	19	5	414 h	4 h
2	26	12	11	3	370 h	1 h
3	28	12	13	3	330 h	2 h
4	27	12	10	5	382 h	1 h
5	30	10	16	4	378 h	0 h

Table 7.1: Characteristics of instances for GUI test

The interactions and restrictions caused by the other activities carried out in the installation are expressed by modifying the availability of resources and technicians. For example, during week 5, some maintenance activities (not carried by the LEPC) required to stop the general ventilation, which is needed for executing activities in the hot cells. So, during the periods where the facility scheduled these maintenances, the capacity of the resource “general ventilation” was set to zero in the scheduling engine. If an LEPC

technician was required to execute an activity outside the LEPC, we just indicate his absence during the periods where such activity was scheduled.

The resulting schedules were presented to the planning team, which validated the consistency of the results. When we compared the total duration of the schedules proposed by the GUI and those generated with the current approach, we observed only a minimal reduction of a few hours (two hours on average). However, what the planning team appreciated most was the possibility of having a new valid schedule after some changes in a short time.

The most criticised issue regarding the solutions generated by the scheduling engine is that sometimes, for preemptive activities, the engine proposed to allocate technicians during intervals that the planning team considered too short, in relation to the total duration of the activity. An evolution of the scheduling models and algorithms should include additional constraints limiting the smaller size of technicians allocation intervals.

7.2 Implementation of the management information system

Once proved the accuracy of our models in a subset of activities, the next step is to test them with all the activities at the LECA-STAR. To achieve this test, it is necessary to have available and operational the management information system for collecting and centralising all the required information. However, at the time of writing this dissertation, the management information system was not totally implemented and operational.

An initial test phase of the new management information system was carried out with a selected group of laboratory employees. This group was asked to transfer their activity scheduling request via a digital form (see Figure 7.7). For this initial test, we use “open” form, i.e., all fields were free. It was decided to initially use this form to imitate as close as possible to the process performed by e-mail. The electronic form required that all fields were filled before sending the request. This is to ensure that all the required information was completed.

Even if the form was not very strict, during the first feedbacks some of the requesters indicated that they were having troubles to complete all the information, since before the planning team always completed the missing information. They also said that filling the new form required more time than completing an e-mail. This is because most of the time, they could reuse old e-mails to transfer their requests.

To overcome this ergonomic problem, as future evolution, one can do an exhaustive characterisation of the most common activities at the facility, and then construct a database with all this information. One can use then this database to auto-complete the request form. This should significantly decrease the time required to fulfil the form. Additionally, this database will allow us to have a better and more accurate description of the activities. It can also improve safety since it ensures that all constraints are always added to everyday activities; what eliminates the risk of forgetting essential constraints

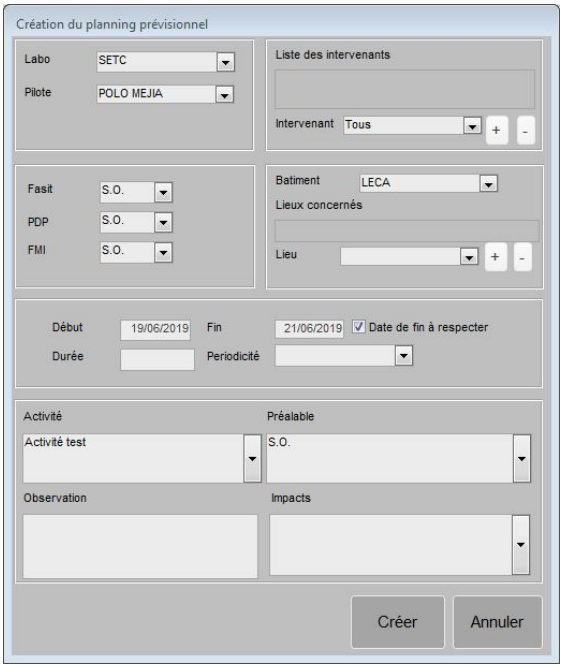

Figure 7.7: Form used for testing the management information system

during the form fulfilling.

Additionally, one can adopt some communication strategies to ensure the adhesion of the employees to the new management system. One must be sure of clearly communicating the objectives and benefits of the new system to all the staff. For this, conferences can be organised in which the advances and results of the new tool are presented. In these conferences, one can resolve doubts and listen to suggestions from employees. An active listening attitude is necessary to guarantee the adherence of all the staff. When the end user participates actively in the development of the tool, the possibility of acceptance increases significantly. One could also organise working groups in which employees make improvement proposals or changes to the management system. All these strategies should facilitate the implementation of the management information system in the near future.

7.3 Concluding remarks

In this chapter, we presented a GUI test for the scheduling engine. This GUI allows the use of the models and algorithms presented in Part II for scheduling the activities of the nuclear research facility. After presenting the obtained schedules for a subset of activities to the planning team, one concludes that the proposed schedules are, in general, consistent and allow a small reduction in the total duration of the schedule.

The possibility of having a new schedule after some changes was the most interesting aspect for the planning team. However, some adjustment need still to be done to ensure more acceptable schedules.

After a first test of the management information system, we identified that some ergonomic problems are the main cause for the delay of new system implementation. To solve the ergonomic issues, one can construct a database with the information of most relevant activities, and use this information to auto-complete the form used for transferring the schedule requests. Conferences and working groups could be used as communication tools for sharing the interest and benefices of the new information system, what could help facilitate the acceptance and implementation of the management information system in the near future.

Conclusions and perspectives

The French Alternative Energies and Atomic Energy Commission or CEA (French: Commissariat à l'Énergie Atomique et aux Énergies alternatives), plays an essential role in the development of the French nuclear industry and research. Its department of nuclear fuel studies (DEC) has as main objective to develop knowledge on different aspects of the nuclear fuels pre- and post-irradiation. The LECA-STAR, a hot laboratory, is the research facility in charge of carrying most of the post-irradiation experiences over nuclear fuel. Because of the strategic importance of this facility for the development of nuclear fuels, one must ensure its optimal operation. The complexity and diversity of the activities carried out at the LECA-STAR makes the scheduling process be a critical optimisation axis.

In this PhD research project, we focused our effort on the development of scheduling models and algorithms, from the combinatorial optimisation theory, allowing the optimisation of the scheduling process at the LECA-STAR. Aware of the importance of having the correct information for the use of our methods, we proposed the development and implementation of a management information system as a side project. This system, in addition to centralising and standardising the scheduling requests, must allow monitoring in real time the state of progress of the activities and utilisation of the resources. These functions should allow being more reactive in front of the unexpected.

After doing a bibliographic review of some combinatorial methods and scheduling problems (Part I of this dissertation) and analysing the characteristics of the facility, we concluded that the problem at hand could be modelled as a preemptive variant of the classical Multi-Skill Project Scheduling Problem (MSPSP). Initially, for limiting the number of times an activity is preempted, we proposed to include a penalty over objective function whenever every activity is stopped, leading to the MSPSP with penalty for preemption presented in Chapter 4. We presented different Mixed Integer/Linear Programming (MILP) formulations for this new variant and achieved some computational experiments to test the performance of each formulation, identifying the two models that gave the best results during the numerical tests. In general, the models we proposed outperformed the results of an adapted version of an already existing MILP formulation for the preemptive MSPSP. A quick theoretical analysis was done to study the strength of our formulations, and we concluded that one of our model (Model 3) has a tighter formulation than another of the propositions (Model 2). However, a more in-depth theoretical analysis should be done as future research to understand better the theoretical strength of our models.

Even if the MSPSP with penalty for preemption allows modelling an important number of the activities carried out at the LECA-STAR, it does not fulfil all the safety requirement for a subset of critical activities. For some activities, we must ensure that a subset of resources remain allocated to the activity when it is preempted. Tradition-

ally, preemptive scheduling models assume that all resources are released during the preemption periods, and this approach of only releasing a subset of resources (partial preemption) had not been considered before. We then proposed a most accurate issue including the concept of partial preemption in Chapter 5: the MSPSP with partial preemption (MSPSP-PP). After describing the characteristics of the new problem, we presented two MILP formulations, along with their theoretical analysis, followed by a Constraint Programming (CP) formulation based on the IBM CP Optimizer interval variables. Computational tests showed that the MILP models perform better when the percentage of preemptive activities in the instances is high. However, they have some troubles when the proportion of non-preemptive activities increases. CP model gave interesting results since it allowed to found initial solutions for all instances (what was not the case for the MILP without warm start). CP model performs better when the instances are highly non-preemptive; and, most of the time, it can prove the optimality faster than MILP models. Nonetheless, if an initial solution is given, the MILP models can prove faster the optimality of instances with a high proportion of preemptive activities. This two techniques can then been considered as complementary, and additional research should be done to try to combine and exploit the characteristics of both.

To comply with the industrial need of having good scheduling solutions in reduced time, we proposed in Chapter 6 various heuristic methods for the MSPSP-PP. A greedy algorithm, based on a serial generation scheme and a flow problem for technician allocation, allows obtaining solutions (in time lower than 1 second) that are near to the ones obtained by the CP solver after 5 min of computing, having an average optimality gap of 7.74%. These results are later improved by a local search algorithm, inspired by the Limited Discrepancy Search, to an average gap of 5.46%. A Greedy Randomised Adaptive Search Procedure (GRASP), that uses the structure of the best solutions to bias the random generation of solutions, allowed to get an average gap of 3.99%. A Large Neighbourhood Search (LNS) with exact subproblems solution was also presented, and allowed to obtain an optimality gap of 3.58%. Finally, executing LNS after GRASP allows getting better results (average gap of 3.02%), largely beating the results of the CP solver (with time limited to 5 min). All proposed methods are negatively affected by the presence of a high proportion of non-preemptive activities (they obtained the higher average gaps). Further research needs to be done in order to improve the quality of solutions when this proportion is high. Special attention must be given to the study of a better local search algorithm since it has an important role in all the proposed methods.

Once developed the models and algorithms for the MSPSP-PP, we presented, in Chapter 7, a GUI test for the scheduling engine, that exploits our previous works for scheduling the activities of the nuclear research facility. The feedback of the planning team is that the proposed schedules are consistent and allow a reduction in the total duration of the schedule. However, some adjustments still need to be done over the scheduling models to better answer the industrial requirements. In the same chapter, we also analyse the main reasons why the implementation process of the management

information system was delayed, identifying some ergonomic problems as the primary cause. To face the ergonomic issues, we proposed the construction of a database with the information of most relevant activities, and use this information to auto-complete the form proposed for transferring the schedule requests. Some communications tools could be also useful for facilitating the acceptance and future implementation of the information system.

The methods and models presented in this dissertation could be extended to other close academic problems such as (partially) preemptive RCPSP [68] and preemptive multi-mode RCPSP [158]. In particular, the alternative representations of the precedence constraints in the MILP formulations could be adapted and tested for these problems as well as for the preemptive version never studied, to the best of our knowledge, of the RCPSP/max (the RCPSP with generalised precedence constraints) [102].

The concept of partial preemption could be useful in other industrial problems where the release of some resources poses operational issues such as in the pharmaceutical and chemical industries. It could also be useful for modelling preemptive scheduling problems where the setup time of activities is only related to a small part of the required resources. Finally, the work presented in this dissertation could be extended to other research laboratories, in particular for CEA facilities but more generally for all projects requiring the scheduling of R&D activities (software development, design activities, etc.).

Bibliography

[1] Karen I. Aardal, Stan P. M. van Hoesel, Arie M. C. A. Koster, Carlo Mannino, and Antonio Sassano. Models and solution techniques for frequency assignment problems. *Annals of Operations Research*, 153(1):79–129, Sep 2007. (Cited in page 23.)

[2] Cristinel Ababei. C++ adapted version of the Edmonds-Karp relabelling MCMF algorithm. https://github.com/eigenpi/MCMF4, December 2009. Online; accessed 01 September 2018. (Cited in page 101.)

[3] Mohammad Abdolshah. A review of resource-constrained project scheduling problems (RCPSP) approaches and solutions. *International Transaction Journal of Engineering, Management, & Applied Sciences & Technologies*, 5(4):253–286, 2014. (Cited in page 43.)

[4] Behrouz Afshar-Nadjafi. Resource constrained project scheduling subject to due dates: preemption permitted with penalty. *Advances in Operations Research*, 2014, 2014. (Cited in pages 40 and 53.)

[5] Behrouz Afshar-Nadjafi and Mahyar Majlesi. Resource constrained project scheduling problem with setup times after preemptive processes. *Computers & Chemical Engineering*, 69:16–25, 2014. (Cited in page 40.)

[6] Ravindra K. Ahuja. *Network Flows: Theory, Algorithms, and Applications*. Pearson Education, 1st edition, 2017. (Cited in page 86.)

[7] Ravindra K. Ahuja, Özlem Ergun, James B Orlin, and Abraham P Punnen. A survey of very large-scale neighborhood search techniques. *Discrete Applied Mathematics*, 123(1-3):75–102, 2002. (Cited in page 32.)

[8] Bernardo F. Almeida. *Multi-skill resource-constrained project scheduling problems: models and algorithms*. PhD thesis, Universidade de Lisboa, Faculdade de Ciências, Lisbon, 2018. (Cited in pages 45 and 47.)

[9] Bernardo F. Almeida, Isabel Correia, and Francisco Saldanha-da Gama. An instance generator for the multi-skill resource-constrained project scheduling problem. *Faculdade de Ciências da Universidade de Lisboa-Centro de Matemática, Aplicações Fundamentais e Investigação Operacional*, 2015. (Cited in page 44.)

[10] Bernardo F. Almeida, Isabel Correia, and Francisco Saldanha-da Gama. Priority-based heuristics for the multi-skill resource constrained project scheduling problem. *Expert Systems with Applications*, 57:91–103, 2016. (Cited in pages 43, 47, and 91.)

[11] Bernardo F. Almeida, Isabel Correia, and Francisco Saldanha-da Gama. Modeling frameworks for the multi-skill resource-constrained project scheduling problem: a theoretical and empirical comparison. *International Transactions in Operational Research*, 26(3):946–967, 2019. (Cited in page 46.)

[12] Arul Amuthan and K. Deepa Thilak. Survey on tabu search meta-heuristic optimization. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*, pages 1539–1543. IEEE, 2016. (Cited in page 33.)

[13] David L. Applegate, Robert E. Bixby, Vasek Chvatal, and William J. Cook. *The traveling salesman problem: a computational study.* Princeton university press, 2006. (Cited in page 23.)

[14] Christian Artigues. The resource-constrained project scheduling problem. In *Resource-constrained project scheduling: models, algorithms, extensions and applications*, pages 21–36. John Wiley & Sons, 2008. (Cited in pages 26, 38, and 44.)

[15] Christian Artigues. On the strength of time-indexed formulations for the resource-constrained project scheduling problem. *Operations Research Letters*, 45(2):154–159, 2017. (Cited in pages 29, 41, 42, and 46.)

[16] Christian Artigues, Philippe Michelon, and Stéphane Reusser. Insertion techniques for static and dynamic resource-constrained project scheduling. *European Journal of Operational Research*, 149(2):249–267, 2003. (Cited in page 41.)

[17] Christian Artigues and David Rivreau. Heuristics. In *Resource-constrained project scheduling: models, algorithms, extensions and applications*, pages 87–105. John Wiley & Sons, 2008. (Cited in page 43.)

[18] Asmaa Atef, Mohamed Abdel-Baset, and Ibrahim El-henawy. Project scheduling: Survey and research potentials. *International Journal of Computer Applications Technology and Research*, 4(4), 2015. (Cited in page 37.)

[19] El-Awady Attia, Philippe Duquenne, and Jean-Marc Le-Lann. Considering skills evolutions in multi-skilled workforce allocation with flexible working hours. *International Journal of Production Research*, 52(15):4548–4573, 2014. (Cited in page 46.)

[20] Parham Azimi and Naeim Azouji. An optimization via simulation approach for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problems. *International Journal of Industrial Engineering & Production Research*, 28(4):429–439, 2017. (Cited in page 41.)

[21] Michael O. Ball. Heuristics based on mathematical programming. *Surveys in Operations Research and Management Science*, 16(1):21–38, 2011. (Cited in page 35.)

[22] Francisco Ballestin, Vicente Valls, and Sacramento Quintanilla. Due dates and RCPSP. In *Perspectives in modern project scheduling*, pages 79–104. Springer, 2006. (Cited in page 40.)

[23] Francisco Ballestín, Vicente Valls, and Sacramento Quintanilla. Pre-emption in resource-constrained project scheduling. *European Journal of Operational Research*, 189(3):1136–1152, 2008. (Cited in page 40.)

[24] Francisco Ballestín, Vicente Valls, and Sacramento Quintanilla. Scheduling projects with limited number of preemptions. *Computers & Operations Research*, 36(11):2913–2925, November 2009. (Cited in page 40.)

[25] Odile Bellenguez and Emmanuel Néron. Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In *International Conference on the Practice and Theory of Automated Timetabling*, pages 229–243. Springer, 2004. (Cited in page 45.)

[26] Odile Bellenguez-Morineau. *Méthodes de résolution pour un problème de gestion de projet multi-compétence*. PhD thesis, Université François Rabelais, Tours, 2006. (Cited in pages 44, 45, 46, 47, and 86.)

[27] Odile Bellenguez-Morineau. Methods to solve multi-skill project scheduling problem. *4OR*, 6(1):85–88, Mar 2008. (Cited in page 44.)

[28] Odile Bellenguez-Morineau and Emmanuel Néron. Multi-mode and multi-skill project scheduling problem. In *Resource-constrained project scheduling: models, algorithms, extensions and applications*, pages 149–160. John Wiley & Sons, 2008. (Cited in page 41.)

[29] Oded Berman and Rongbing Huang. The minimum weighted covering location problem with distance constraints. *Computers & Operations Research*, 35(2):356–372, 2008. (Cited in page 25.)

[30] Christian Bessière. Constraint propagation. In Frank van Harmelen and Toby Walsh, editors, *Handbook of Constraint Programming*, volume 2 of *Foundations of Artificial Intelligence*, pages 29–83. Elsevier, 2006. (Cited in page 30.)

[31] Lucio Bianco and Massimiliano Caramia. A new formulation for the project scheduling problem under limited resources. *Flexible Services and Manufacturing Journal*, 25(1-2):6–24, 2013. (Cited in page 42.)

[32] Lucio Bianco and Massimiliano Caramia. The resource constrained project scheduling problem: A theoretical comparison between a recent formulation and the main time indexed linear programming based approaches. *RAIRO-Operations Research*, 51(3):519–532, 2017. (Cited in page 42.)

[33] Jacek Błazewicz, Jan Karel Lenstra, and Alexander H.G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete applied mathematics*, 5(1):11–24, 1983. (Cited in pages 43, 45, and 68.)

[34] Srinivas Bollapragada and Marc Garbiras. Scheduling commercials on broadcast television. *Operations Research*, 52(3):337–345, June 2004. (Cited in pages 20 and 22.)

[35] Chawis Boonmee, Mikiharu Arimura, and Takumi Asada. Facility location optimization model for emergency humanitarian logistics. *International Journal of Disaster Risk Reduction*, 24:485–498, 2017. (Cited in page 24.)

[36] Ilhem Boussaïd, Julien Lepagnot, and Patrick Siarry. A survey on optimization metaheuristics. *Information sciences*, 237:82–117, 2013. (Cited in pages 31 and 34.)

[37] Kris Braekers, Katrien Ramaekers, and Inneke Van Nieuwenhuyse. The vehicle routing problem: State of the art classification and review. *Computers & Industrial Engineering*, 99:300–313, 2016. (Cited in page 24.)

[38] Mario Brčić, Damir Kalpić, and Krešimir Fertalj. Resource constrained project scheduling under uncertainty: a survey. In *23rd Central European Conference on Information and Intelligent Systems*, 2012. (Cited in page 39.)

[39] Tyson R Browning and Ali A Yassine. Resource-constrained multi-project scheduling: Priority rule performance revisited. *International Journal of Production Economics*, 126(2):212–228, 2010. (Cited in page 43.)

[40] Peter Brucker and Sigrid Knust. *Complex scheduling*. Springer-Verlag Berlin Heidelberg, 2nd edition, 2012. (Cited in pages 25 and 26.)

[41] Peter Brucker and Rainer Schlie. Job-shop scheduling with multi-purpose machines. *Computing*, 45(4):369–375, 1990. (Cited in page 25.)

[42] Jirachai Buddhakulsomsiri and David S Kim. Priority rule-based heuristic for multi-mode resource-constrained project scheduling problems with resource vacations and activity splitting. *European Journal of Operational Research*, 178(2):374–390, 2007. (Cited in pages 41 and 43.)

[43] Rong Chen, Changyong Liang, Dongxiao Gu, and Joseph Y.T. Leung. A multi-objective model for multi-project scheduling and multi-skilled staff assignment for it product development considering competency evolution. *International Journal of Production Research*, 55(21):6207–6234, 2017. (Cited in page 46.)

[44] Shang-Kuan Chen, Yen-Wu Ti, and Kuo-Yu Tsai. Nuclear power plant construction scheduling problem with time restrictions: a particle swarm optimization

approach. *Science and Technology of Nuclear Installations*, 2016, 2016. (Cited in page 11.)

[45] Junzilan Cheng, John Fowler, Karl Kempf, and Scott Mason. Multi-mode resource-constrained project scheduling problems with non-preemptive activity splitting. *Computers & Operations Research*, 53:275–287, 2015. (Cited in page 40.)

[46] Jean-François Cordeau, Gilbert Laporte, Martin W.P. Savelsbergh, and Daniele Vigo. Vehicle routing. *Handbooks in operations research and management science*, 14:367–428, 2007. (Cited in page 24.)

[47] Isabel Correia, Lídia Lampreia Lourenço, and Francisco Saldanha-da Gama. Project scheduling with flexible resources: formulation and inequalities. *OR spectrum*, 34(3):635–663, 2012. (Cited in page 46.)

[48] Isabel Correia and Francisco Saldanha-da Gama. A modeling framework for project staffing and scheduling problems. In *Handbook on Project Management and Scheduling Vol. 1*, pages 547–564. Springer, 2015. (Cited in page 45.)

[49] Isabel Correia and Francisco Saldanha-da Gama. A note on "branch-and-price approach for the multi-skill project scheduling problem". *Optimization Letters*, 9(6):1255–1258, 2015. (Cited in page 46.)

[50] Carlos Cotta, Luke Mathieson, and Pablo Moscato. Memetic algorithms. In Rafael Martí, Panos M. Pardalos, and Mauricio G. C. Resende, editors, *Handbook of Heuristics*, pages 607–638. Springer, 2018. (Cited in page 34.)

[51] Jean Damay, Alain Quilliot, and Eric Sanlaville. Linear programming based algorithms for preemptive and non-preemptive RCPSP. *European Journal of Operational Research*, 182(3):1012–1022, 2007. (Cited in page 26.)

[52] Mark S. Daskin and Kayse Lee Maass. Location analysis and network design. In *Operations, Logistics and Supply Chain Management*, pages 379–398. Springer, 2019. (Cited in page 25.)

[53] Stéphane Dauzère-Pérès, William Roux, and Jean-Bernard Lasserre. Multi-resource shop scheduling with resource flexibility. *European Journal of Operational Research*, 107(2):289–305, 1998. (Cited in pages 25 and 26.)

[54] Federico Della Croce, Fabio Salassa, and Rosario Scatamacchia. A new exact approach for the 0–1 Collapsing Knapsack Problem. *European Journal of Operational Research*, 260(1):56–69, July 2017. (Cited in page 22.)

[55] Sophie Demassey, Christian Artigues, and Philippe Michelon. Constraint-propagation-based cutting planes: An application to the resource-constrained

project scheduling problem. *INFORMS Journal on computing*, 17(1):52–65, 2005. (Cited in page 42.)

[56] Sachin Desale, Akhtar Rasool, Sushil Andhale, and Priti Rane. Heuristic and meta-heuristic algorithms and their relevance to the real world: a survey. *International journal of computer engineering in research trends*, 2(5):296–304, 2015. (Cited in page 34.)

[57] Jacques Desrosiers and Marco E. Lübbecke. Branch-price-and-cut algorithms. *Wiley encyclopedia of operations research and management science*, 2010. (Cited in page 29.)

[58] Cheikh Dhib, Ameur Soukhal, and Emmanuel Néron. Mixed-integer linear programming formulation and priority-rule methods for a preemptive project staffing and scheduling problem. In *Handbook on Project Management and Scheduling Vol. 1*, pages 603–617. Springer, 2015. (Cited in page 46.)

[59] Fabián Díaz-Núñez, Nir Halman, and Óscar C. Vásquez. The TV advertisements scheduling problem. *Optimization Letters*, 13(1):81–94, Feb 2019. (Cited in page 22.)

[60] Michael Drexl and Michael Schneider. A survey of variants and extensions of the location-routing problem. *European Journal of Operational Research*, 241(2):283–308, 2015. (Cited in page 25.)

[61] Nicolas Dupin and El-Ghazali Talbi. Dual heuristics and new lower bounds for the challenge euro/roadef 2010. *Matheuristics 2016*, page 60, 2016. (Cited in page 11.)

[62] Fabián Díaz-Núñez, Franco Quezada, and Óscar C. Vásquez. The knapsack problem with scheduled items. *Electronic Notes in Discrete Mathematics*, 69:293–300, August 2018. (Cited in page 22.)

[63] Claudia D'Ambrosio, Fabio Furini, Michele Monaci, and Emiliano Traversi. On the product knapsack problem. *Optimization Letters*, 12(4):691–712, June 2018. (Cited in page 22.)

[64] Jack Edmonds and Richard M. Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, April 1972. (Cited in page 86.)

[65] Burak Eksioglu, Arif Volkan Vural, and Arnold Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009. (Cited in page 24.)

[66] Majid Eskandarpour, Pierre Dejax, Joe Miemczyk, and Olivier Péton. Sustainable supply chain network design: An optimization-oriented review. *Omega*, 54:11–32, July 2015. (Cited in page 20.)

[67] Marshall L Fisher. The lagrangian relaxation method for solving integer programming problems. *Management science*, 50(12_supplement):1861–1871, 2004. (Cited in page 28.)

[68] Pierre Fouilhoux, A. Ridha Mahjoub, Alain Quilliot, and Hélène Toussaint. Branch-and-cut-and-price algorithms for the preemptive RCPSP. *RAIRO-Operations Research*, 52(2):513–528, 2018. (Cited in page 125.)

[69] Leslie R Foulds. *Graph theory applications*. Springer Science & Business Media, 2012. (Cited in page 23.)

[70] Alberto Franzin and Thomas Stützle. Revisiting simulated annealing: A component-based analysis. *Computers & Operations Research*, 104:191–206, 2019. (Cited in page 33.)

[71] Alex S. Fukunaga. A branch-and-bound algorithm for hard multiple knapsack problems. *Annals of Operations Research*, 184(1):97–119, Apr 2011. (Cited in page 22.)

[72] M.H. Karimi Gavareshki. New fuzzy gert method for research projects scheduling. In *2004 IEEE International Engineering Management Conference (IEEE Cat. No. 04CH37574)*, volume 2, pages 820–824. IEEE, 2004. (Cited in pages 11 and 37.)

[73] Krasimira Genova and Vassil Guliashki. Linear integer programming methods and approaches–a survey. *Journal of Cybernetics and Information Technologies*, 11(1), 2011. (Cited in pages 27 and 28.)

[74] Ralph E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical society*, 64(5):275–278, 1958. (Cited in page 27.)

[75] Sathish Gopalakrishnan, Lui Sha, and Marco Caccamo. Hard real-time communication in bus-based networks. In *25th IEEE International Real-Time Systems Symposium*, pages 405–414, Dec 2004. (Cited in page 23.)

[76] Farhad Habibi, Farnaz Barzinpour, and Seyed Sadjadi. Resource-constrained project scheduling problem: review of past and recent developments. *Journal of Project Management*, 3(2):55–88, 2018. (Cited in pages 39 and 41.)

[77] Farhad Habibi, Farnaz Barzinpour, and Seyed Jafar Sadjadi. A multi-objective optimization model for project scheduling with time-varying resource requirements and capacities. *Journal of Industrial and Systems Engineering*, 10:92–118, 2017. (Cited in page 40.)

[78] Sönke Hartmann. Project scheduling with resource capacities and requests varying with time: a case study. *Flexible services and manufacturing journal*, 25(1-2):74–93, 2013. (Cited in page 40.)

[79] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1–14, 2010. (Cited in pages 38, 39, and 40.)

[80] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *IJCAI (1)*, pages 607–615, 1995. (Cited in page 91.)

[81] Farhad Hassanzadeh, Mohammad Modarres, Hamid R. Nemati, and Kwasi Amoako-Gyampah. A robust R&D project portfolio optimization model for pharmaceutical contract research organizations. *International Journal of Production Economics*, 158:18–27, 2014. (Cited in page 11.)

[82] Farhad Hassanzadeh, Hamid Nemati, and Minghe Sun. Robust optimization for interactive multiobjective programming with imprecise information applied to r&d project portfolio selection. *European Journal of Operational Research*, 238(1):41–53, 2014. (Cited in page 11.)

[83] Stephan Held, Bernhard Korte, Dieter Rautenbach, and Jens Vygen. Combinatorial optimization in VLSI design. *Combinatorial Optimization-Methods and Applications*, 31:33–96, 2011. (Cited in page 20.)

[84] Willy Herroelen. Project scheduling – Theory and practice. *Production and Operations Management*, 14(4):413–432, 2005. (Cited in page 25.)

[85] Karla L. Hoffman and Ted K. Ralphs. Integer and combinatorial optimization. In Saul I. Gass and Michael C. Fu, editors, *Encyclopedia of Operations Research and Management Science*, pages 771–783. Springer US, Boston, MA, 2013. (Cited in pages 21, 22, 23, 24, 27, 28, and 29.)

[86] W Jaśkowski, Marcin Szubert, and Piotr Gawron. A hybrid mip-based large neighborhood search heuristic for solving the machine reassignment problem. *Annals of Operations Research*, 242(1):33–62, 2016. (Cited in page 35.)

[87] Shima Javanmard, Behrouz Afshar-Nadjafi, and Seyed Taghi Akhavan Niaki. Preemptive multi-skilled resource investment project scheduling problem: Mathematical modelling and solution approaches. *Computers & Chemical Engineering*, 96:55–68, 2017. (Cited in pages 44 and 45.)

[88] Benjamin Johnson, Alexandra Newman, and Jeffrey King. Optimizing high-level nuclear waste disposal within a deep geologic repository. *Annals of Operations Research*, 253(2):733–755, 2017. (Cited in page 11.)

[89] Vincent Jost and David Savourey. A 0–1 integer linear programming approach to schedule outages of nuclear power plants. *Journal of Scheduling*, 16(6):551–566, 2013. (Cited in page 11.)

[90] Zeki Karaca and Turgay Onargan. The application of critical path method (CPM) in workflow schema of marble processing plants. *Materials and Manufacturing Processes*, 22(1):37–44, 2007. (Cited in page 37.)

[91] Aniket Kate and Ian Goldberg. Generalizing cryptosystems based on the subset sum problem. *International Journal of Information Security*, 10(3):189–199, 2011. (Cited in pages 20 and 21.)

[92] Hamed Kazemipoor, Reza Tavakkoli-Moghaddam, and P Shahnazari-Shahrezaei. Solving a novel multi-skilled project scheduling model by scatter search. *South African Journal of Industrial Engineering*, 24(1):121–131, 2013. (Cited in page 46.)

[93] Hans Kellerer, Ulrich Pferschy, and David Pisinger. *Knapsack problems*. Springer-Verlag Berlin Heidelberg, Berlin, 1 edition, 2004. (Cited in pages 21 and 22.)

[94] Maurice G. Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938. (Cited in page 96.)

[95] Mohamed Ali Khemakhem and Hédi Chtourou. Efficient robustness measures for the resource-constrained project scheduling problem. *International Journal of Industrial and Systems Engineering*, 14(2):245–267, 2013. (Cited in page 11.)

[96] Reza Kia, Parisa Shahnazari-Shahrezaei, and Sina Zabihi. Solving a multi-objective mathematical model for a multi-skilled project scheduling problem by cplex solver. In *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 1220–1224. IEEE, 2016. (Cited in page 45.)

[97] Rainer Kolisch and Sönke Hartmann. Experimental investigation of heuristics for resource-constrained project scheduling: An update. *European Journal of Operational Research*, 174(1):23–37, 2006. (Cited in page 43.)

[98] Rainer Kolisch and Rema Padman. An integrated survey of deterministic project scheduling. *Omega*, 29(3):249–272, 2001. (Cited in page 38.)

[99] Oumar Koné, Christian Artigues, Pierre Lopez, and Marcel Mongeau. Event-based MILP models for resource-constrained project scheduling problems. *Computers & Operations Research*, 38(1):3–13, 2011. (Cited in page 41.)

[100] Georgios M. Kopanos, Thomas S. Kyriakidis, and Michael C. Georgiadis. New continuous-time and discrete-time mathematical formulations for resource-constrained project scheduling problems. *Computers & Chemical Engineering*, 68:96 – 106, 2014. (Cited in page 41.)

[101] Bernhard Korte and Jens Vygen. *Combinatorial optimization*, volume 21 of *Algorithms and Combinatorics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. DOI: 10.1007/978-3-642-24488-9. (Cited in pages 20 and 27.)

[102] Stefan Kreter, Andreas Schutt, and Peter J. Stuckey. Using constraint programming for solving RCPSP/max-cal. *Constraints*, 22(3):432–462, 2017. (Cited in pages 40, 42, and 125.)

[103] Philippe Laborie. IBM ILOG CP Optimizer for detailed scheduling illustrated on three problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, Lecture Notes in Computer Science, pages 148–162. Springer, Berlin, Heidelberg, May 2009. (Cited in page 40.)

[104] Philippe Laborie, Jérôme Rogerie, Paul Shaw, and Petr Vilím. IBM ILOG CP optimizer for scheduling. *Constraints*, 23(2):210–250, 2018. (Cited in page 74.)

[105] Claude Le Pape. Programmation par contraintes. In Vangelis Th. Paschos, editor, *Optimisation combinatoire 1 : concepts fondamentaux*, pages 331–345. Hermès Sci. Lavoisier Paris, 2005. (Cited in page 31.)

[106] Yin Tat Lee, Aaron Sidford, and Sam Chiu-Wai Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*, pages 1049–1065, Oct 2015. (Cited in page 28.)

[107] Joseph Y.T. Leung and Chung-Lun Li. Scheduling with processing set restrictions: A survey. *International Journal of Production Economics*, 116(2):251–262, 2008. (Cited in page 25.)

[108] Pieter Leyman and Mario Vanhoucke. Capital- and resource-constrained project scheduling with net present value optimization. *European Journal of Operational Research*, 256(3):757 – 776, 2017. (Cited in page 39.)

[109] Haitao Li and Keith Womer. Scheduling projects with multi-skilled personnel by a hybrid MILP/CP benders decomposition algorithm. *Journal of Scheduling*, 12(3):281, 2009. (Cited in pages 45, 46, and 47.)

[110] Olivier Liess and Philippe Michelon. A constraint programming approach for the resource-constrained project scheduling problem. *Annals of Operations Research*, 157(1):25–36, 2008. (Cited in page 42.)

[111] Kuo-Ping Lin, Wu Wen, Chang-Chien Chou, Chih-Hung Jen, and Kuo-Chen Hung. Applying fuzzy gert with approximate fuzzy arithmetic based on the weakest t-norm operations to evaluate repairable reliability. *Applied Mathematical Modelling*, 35(11):5314–5325, 2011. (Cited in page 37.)

[112] Richard Lowry. Wilcoxon signed-rank test. `http://vassarstats.net/textbook/ch12a.html`, 2019. Accessed: 20-05-2019. (Cited in pages 61 and 101.)

[113] Catia M.S. Machado, Sergio F. Mayerle, and Vilmar Trevisan. A linear model for compound multicommodity network flow problems. *Computers & Operations Research*, 37(6):1075–1086, 2010. (Cited in page 23.)

[114] Hamidreza Maghsoudlou, Behrouz Afshar-Nadjafi, and Seyed Taghi Akhavan Niaki. Preemptive multi-skilled resource constrained project scheduling problem with hard/soft interval due dates. *RAIRO-Operations Research*, 2018. (Cited in pages 45, 46, and 53.)

[115] Enrico Malaguti and Paolo Toth. A survey on vertex coloring problems. *International Transactions in Operational Research*, 17(1):1–34, 1 2010. (Cited in page 23.)

[116] Catherine Mancel. *modélisation et résolution de problèmes d'optimisation combinatoire issus d'applications spatiales.* PhD thesis, INSA de Toulouse, 2004. (Cited in page 12.)

[117] Pedro Martins, Antonio Ladrón, and Helena Ramalhinho. Maximum cut-clique problem: ILS heuristics and a data analysis application. *International Transactions in Operational Research*, 22(5):775–809, September 2015. (Cited in pages 20 and 23.)

[118] Teresa Melo, Stefan Nickel, and Francisco Saldanha-Da-Gama. Facility location and supply chain management–a review. *European Journal of Operational Research*, 196(2):401–412, 2009. (Cited in page 24.)

[119] Aristide Mingozzi, Vittorio Maniezzo, Salvatore Ricciardelli, and Lucio Bianco. An exact algorithm for the resource-constrained project scheduling problem based on a new mathematical formulation. *Management science*, 44(5):714–729, 1998. (Cited in page 41.)

[120] Carlos Montoya, Odile Bellenguez-Morineau, Eric Pinson, and David Rivreau. Branch-and-price approach for the multi-skill project scheduling problem. *Optimization Letters*, 8(5):1721–1734, 2014. (Cited in pages 46 and 47.)

[121] Daniel Morillo-Torres, Luis Fernando Moreno-Velásquez, and Francisco Javier Díaz-Serna. A branch and bound hybrid algorithm with four deterministic heuristics for the resource constrained project scheduling problem (rcpsp). *Dyna*, 82(190):198–207, 2015. (Cited in page 43.)

[122] David R. Morrison, Sheldon H. Jacobson, Jason J. Sauppe, and Edward C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016. (Cited in pages 28 and 29.)

[123] Paweł B. Myszkowski and Jędrzej J. Siemieński. GRASP applied to multi-skill resource-constrained project scheduling problem. In *International Conference on Computational Collective Intelligence*, pages 402–411. Springer, 2016. (Cited in pages 47 and 48.)

[124] Paweł B. Myszkowski, Marek E. Skowroński, and Łukasz Podlodowski. Novel heuristic solutions for multi-skill resource-constrained project scheduling problem. In *2013 Federated Conference on Computer Science and Information Systems*, pages 159–166. IEEE, 2013. (Cited in page 47.)

[125] Najib M. Najid and Marouane Arroub. An efficient algorithm for the multi-mode resource constrained project scheduling problem with resource flexibility. *International Journal of Mathematics in Operational Research*, 2(6):748–761, 2010. (Cited in page 41.)

[126] Satyasai Jagannath Nanda and Ganapati Panda. A survey on nature inspired metaheuristic algorithms for partitional clustering. *Swarm and Evolutionary computation*, 16:1–18, 2014. (Cited in page 34.)

[127] Emmanuel Néron. Lower bounds for the multi-skill project scheduling problem. In *Proceeding of the Eighth International Workshop on Project Management and Scheduling*, pages 274–277, 2002. (Cited in pages 44, 45, 52, and 86.)

[128] Frank Neumann and Carsten Witt. *Bioinspired Computation in Combinatorial Optimization: Algorithms and Their Computational Complexity*, chapter 2, pages 9–19. Springer Berlin Heidelberg, 2010. (Cited in page 20.)

[129] Siamak Noori and Kaveh Taghizadeh. Multi-mode resource constrained project scheduling problem: A survey of variants, extensions, and methods. *International Journal of Industrial Engineering & Production Research*, 29(3):293–320, 2018. (Cited in page 41.)

[130] Gholamreza Norouzi, Mehdi Heydari, Siamak Noori, and Morteza Bagherpour. Developing a mathematical model for scheduling and determining success probability of research projects considering complex-fuzzy networks. *Journal of Applied Mathematics*, 2015(Article ID 809216), June 2015. 15 pages. (Cited in pages 11 and 37.)

[131] Ifeyinwa M. J. Orji and Sun Wei. Project scheduling under resource constraints: A recent survey. *International Journal of Engineering Research and Technology*, 2(2), February 2013. (Cited in page 39.)

[132] Mireille Palpant, Christian Artigues, and Philippe Michelon. LSSPER: Solving the resource-constrained project scheduling problem with large neighbourhood search. *Annals of Operations Research*, 131(1-4):237–257, 2004. (Cited in pages 44 and 97.)

[133] José Antonio Parejo, Antonio Ruiz-Cortés, Sebastián Lozano, and Pablo Fernandez. Metaheuristic optimization frameworks: a survey and benchmarking. *Soft Computing*, 16(3):527–561, 2012. (Cited in pages 31 and 33.)

[134] Vangelis Th. Paschos. *Applications of Combinatorial Optimization.* Wiley-ISTE, 2nd edition, 2014. (Cited in page 20.)

[135] Robert Pellerin, Nathalie Perrier, and François Berthaut. A survey of hybrid metaheuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, 2019. (Cited in pages 35 and 43.)

[136] Fátima Pérez, Trinidad Gómez, Rafael Caballero, and Vicente Liern. Project portfolio selection and planning with fuzzy constraints. *Technological Forecasting and Social Change*, 131:117–129, 2018. (Cited in page 11.)

[137] Gordon Matthew Petersen. *Algorithms and Methods for Optimizing the Spent Nuclear Fuel Allocation Strategy.* PhD thesis, University of Tennessee, Knoxville, USA, 2016. (Cited in page 11.)

[138] Michael L. Pinedo. *Planning and scheduling in manufacturing and services.* Springer, 2005. (Cited in pages 25 and 26.)

[139] Michael L. Pinedo. *Scheduling - Theory, Algorithms, and Systems.* Springer International Publishing, 5 edition, 2016. (Cited in pages 20 and 26.)

[140] David Pisinger. The quadratic knapsack problem—a survey. *Discrete Applied Mathematics*, 155(5):623–648, March 2007. (Cited in page 22.)

[141] David Pisinger and Stefan Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010. (Cited in page 32.)

[142] Oliver Polo-Mejía, Marie-Christine Anselmet, Christian Artigues, and Pierre Lopez. A new RCPSP variant for scheduling research activities in a nuclear laboratory. In *47th International Conference on Computers & Industrial Engineering (CIE47)*, pages 463–470, Lisbon, Portugal, October 2017. (Cited in pages 51 and 53.)

[143] Oliver Polo-Mejía, Marie-Christine Anselmet, Christian Artigues, and Pierre Lopez. Mixed-integer and constraint programming formulations for a multi-skill project scheduling problem with partial preemption. In *12th International Conference on Modelling, Optimization and Simulation (MOSIM 2018)*, pages 367–374, Toulouse, France, June 2018. (Cited in pages 65 and 68.)

[144] Oliver Polo-Mejía, Marie-Christine Anselmet, Christian Artigues, and Pierre Lopez. Multi-skill project scheduling in a nuclear research facility. In *Proceedings of the 16th International Conference on Project Management and Scheduling*, pages 367–374, April 2018. (Cited in pages 65 and 68.)

[145] Oliver Polo-Mejía, Christian Artigues, and Pierre Lopez. A heuristic method for the multi-skill project scheduling problem with partial preemption. In *8th International Conference on Operations Research and Enterprise Systems*, pages 111–120, Prague, Czech Republic, February 2019. (Cited in pages 85 and 95.)

[146] Oliver Polo-Mejía, Christian Artigues, Pierre Lopez, and Lars Mönch. Memory and feasibility indicators in GRASP for multi-skill project. In *Metaheuristics International Conference*, Cartagena, Colombia, July 2019. (Cited in page 85.)

[147] Marcelo Prais and Celso C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12(3):164–176, 2000. (Cited in page 95.)

[148] Caroline Prodhon and Christian Prins. A survey of recent research on location-routing problems. *European Journal of Operational Research*, 238(1):1–17, 2014. (Cited in page 25.)

[149] Jakob Puchinger, Peter J. Stuckey, Mark G. Wallace, and Sebastian Brand. Dantzig-wolfe decomposition and branch-and-price solving in G12. *Constraints*, 16(1):77–99, 2011. (Cited in page 29.)

[150] Ragheb Rahmaniani, Teodor Gabriel Crainic, Michel Gendreau, and Walter Rei. The Benders decomposition algorithm: A literature review. *European Journal of Operational Research*, 259(3):801–817, 2017. (Cited in page 28.)

[151] Günther R. Raidl, Jakob Puchinger, and Christian Blum. Metaheuristic hybrids. In Michel Gendreau and Jean-Yves Potvin, editors, *Handbook of Metaheuristics*, pages 385–417. Springer International Publishing, Cham, 2019. (Cited in page 35.)

[152] Mauricio G. C. Resende and Celso C. Ribeiro. Greedy randomized adaptive search procedures: Advances and extensions. In *Handbook of Metaheuristics*, pages 169–220. Springer, 2019. (Cited in pages 33 and 93.)

[153] Bert De Reyck and Roel Leus. R&D project scheduling when activities may fail. *IIE Transactions*, 40(4):367–384, 2008. (Cited in page 11.)

[154] Francesca Rossi, Peter van Beek, and Toby Walsh. Constraint programming. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*, volume 3 of *Foundations of Artificial Intelligence*, pages 181–211. Elsevier, January 2008. (Cited in pages 30 and 31.)

[155] Michael Schneider and Michael Drexl. A survey of the standard location-routing problem. *Annals of Operations Research*, 259(1-2):389–414, 2017. (Cited in page 25.)

[156] Alexander Schnell and Richard F Hartl. On the generalization of constraint programming and boolean satisfiability solving techniques to schedule a resource-constrained project consisting of multi-mode jobs. *Operations Research Perspectives*, 4:1–11, 2017. (Cited in page 42.)

[157] Vacharee Tantisuvanichkul and Moray Kidd. Project scheduling a review through literature. In *RICS Construction and Property Conference*, page 1678, 2011. (Cited in page 37.)

[158] Vincent Van Peteghem and Mario Vanhoucke. A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research*, 201(2):409–418, 2010. (Cited in page 125.)

[159] Mario Vanhoucke. Setup times and fast tracking in resource-constrained project scheduling. *Computers & Industrial Engineering*, 54(4):1062–1070, May 2008. (Cited in page 40.)

[160] Petr Vilím. *Global constraints in scheduling.* PhD thesis, Univerzita Karlova, Matematicko-fyzikální fakulta, 2007. (Cited in page 30.)

[161] Stefan Voß, Andreas Fink, and Cees Duin. Looking ahead with the pilot method. *Annals of Operations Research*, 136(1):285–302, 2005. (Cited in pages 32 and 91.)

[162] Yan-Ling Wang. Combinatorial optimization design and operating management mechanisms for logistics supply chain management. In *2010 IEEE International Conference on Management of Innovation & Technology*, pages 192–196, Singapore, Singapore, 2010. IEEE. (Cited in page 20.)

[163] Jan Węglarz, Joanna Józefowska, Marek Mika, and Grzegorz Waligóra. Project scheduling with finite or infinite number of activity processing modes–a survey. *European Journal of Operational Research*, 208(3):177–205, 2011. (Cited in page 45.)

[164] Hamidreza Yoosefzadeh and Hamed Reza Tareghian. Hybrid solution method for resource-constrained project scheduling problem using a new schedule generator. *The International Journal of Advanced Manufacturing Technology*, 66(5-8):1171–1180, 2013. (Cited in page 43.)

[165] Kenneth D. Young, Thibaut Feydy, and Andreas Schutt. Constraint programming applied to the Multi-Skill Project Scheduling Problem. In *Principles and Practice of Constraint Programming*, Lecture Notes in Computer Science, pages 308–317. Springer, Cham, August 2017. (Cited in page 47.)

[166] Jun Zhou, Jinghong Peng, Guangchuan Liang, and Tao Deng. Layout optimization of tree-tree gas pipeline network. *Journal of Petroleum Science and Engineering*, 173:666–680, 2019. (Cited in page 23.)

[167] Jie Zhu, Xiaoping Li, and Weiming Shen. Effective genetic algorithm for resource-constrained project scheduling with limited preemptions. *International Journal of Machine Learning and Cybernetics*, 2(2):55–65, 2011. (Cited in page 40.)