Mixed-Integer Linear Programming Based Approaches for the Resource Constrained Project Scheduling Problem

Janniele Aparecida Soares Araujo
Universidade Federal de Ouro Preto

Orientador: Haroldo Gambini Santos

UNIVERSIDADE FEDERAL DE OURO PRETO

Mixed-Integer Linear Programming Based Approaches for the Resource Constrained Project Scheduling Problem

Janniele Aparecida Soares Araujo
Universidade Federal de Ouro Preto

Orientador: Haroldo Gambini Santos

Tese submetida ao Instituto de Ciências Exatas e Biológicas da Universidade Federal de Ouro Preto para obtenção do título de Doutora em Ciência da Computação.

Ouro Preto, Dezembro de 2019

A111m Araujo, Janniele Aparecida Soares .
Mixed-integer linear programming based approaches for the resource constrained project scheduling problem [manuscrito] / Janniele Aparecida Soares Araujo. - 2019.
96f.: il.: color; grafs; tabs.

Orientador: Prof. Dr. Haroldo Gambini Santos.

Tese (Doutorado) - Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Departamento de Computação. Programa de Pós-Graduação em Ciência da Computação.
Área de Concentração: Ciência da Computação.

1. Financiamento de projetos. 2. Orçamento-programa. 3. Programação linear . I. Santos, Haroldo Gambini . II. Universidade Federal de Ouro Preto. III. Titulo.

CDU: 004.421

Catalogação: www.sisbin.ufop.br

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE OURO PRETO
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UFOP

ATA DE DEFESA DE DOUTORADO

Aos 06 dias do mês de dezembro do ano de 2019, às 13:30 horas, nas dependências do Departamento de Computação (Decom), foi instalada a sessão pública para a defesa de tese da doutoranda Janniele Aparecida Soares Araujo, sendo a banca examinadora composta pelo Prof. Dr. Haroldo Gambini Santos (Presidente - UFOP), pelo Eduardo Uchoa Barboza (Membro - Externo), pelo Prof. Dr. Marcone Jamilson Freitas Souza (Membro - UFOP), pelo Prof. Dr. Sanjay Dominik Jena (Membro - Externo), pelo Prof. Dr. Tulio Angelo Machado Toffolo (Membro - UFOP). Dando início aos trabalhos, o presidente, com base no regulamento do curso e nas normas que regem as sessões de defesa de tese, concedeu à doutoranda 60 minutos para apresentação do seu trabalho intitulado "Mixed-Integer Linear Programming Based Approaches for the Resource Constrained Project Scheduling Problem". Terminada a exposição, o presidente da banca examinadora concedeu, a cada membro, um tempo máximo de 30 minutos para perguntas e respostas à candidata sobre o conteúdo da tese, na seguinte ordem: Primeiro Prof. Dr. Sanjay Dominik Jena; segundo Eduardo Uchoa Barboza; terceiro Prof. Dr. Tulio Angelo Machado Toffolo; quarto Prof. Dr. Marcone Jamilson Freitas Souza; quinto Prof. Dr. Haroldo Gambini Santos. Dando continuidade, ainda de acordo com as normas que regem a sessão, o presidente solicitou aos presentes que se retirassem do recinto para que a banca examinadora procedesse à análise e decisão, anunciando, a seguir, publicamente, que a doutoranda foi aprovada por unanimidade, sob a condição de que a versão definitiva da tese deva incorporar todas as exigências da banca, devendo o exemplar final ser entregue no prazo máximo de 60 (sessenta) dias à Coordenação do Programa. Para constar, foi lavrada a presente ata que, após aprovada, vai assinada pelos membros da banca examinadora e pela doutoranda. Ouro Preto, 06 de dezembro de 2019.

Prof. Dr. Haroldo Gambini Santos
Presidente

Eduardo Uchoa Barboza

Prof. Dr. Marcone Jamilson Freitas Souza

xxxxxxxxxxxxxxxxxxxxxxxxxx
Prof. Dr. Sanjay Dominik Jena
(Participação por Videoconferência)

Prof. Dr. Tulio Angelo Machado Toffolo

Janniele Aparecida Soares Araujo
Doutoranda

Certifico que a defesa realizou-se com a participação a distância do(s) membros(s) Prof. Dr. Sanjay Dominik Jena e que, depois das arguições e deliberações realizadas, cada participante a distância afirmou estar de acordo com o conteúdo do parecer da banca examinadora, redigido nesta ata.

Prof. Dr. Haroldo Gambini Santos
Presidente

Dedico este trabalho a Deus*, aos meus pais* José Geraldo *e* Rosângela*, ao meu marido* Ítalo *e ao meu orientador* Haroldo*, pessoas de suma importância em minha vida.*

Mixed-Integer Linear Programming Based Approaches for the Resource Constrained Project Scheduling Problem

Abstract

Resource Constrained Project Scheduling Problems (RCPSPs) without preemption are well-known $\mathcal{NP}$-hard combinatorial optimization problems. A feasible RCPSP solution consists of a time-ordered schedule of jobs with corresponding execution modes, respecting precedence and resources constraints. First, in this thesis, we provide improved upper bounds for many hard instances from the literature by using methods based on Stochastic Local Search (SLS). As the most contribution part of this work, we propose a cutting plane algorithm to separate five different cut families, as well as a new preprocessing routine to strengthen resource-related constraints. New lifted versions of the well-known precedence and cover inequalities are employed. At each iteration, a dense conflict graph is built considering feasibility and optimality conditions to separate cliques, odd-holes and strengthened Chvátal-Gomory cuts. The proposed strategies considerably improve the linear relaxation bounds, allowing a state-of-the-art mixed-integer linear programming solver to find provably optimal solutions for 754 previously open instances of different variants of the RCPSPs, which was not possible using the original linear programming formulations.

Keywords: resource constrained project secheduling, mixed-integer linear programming, preprocessing, cutting planes, stochastic local search, neigborhood composition.

Declaração

Esta tese é resultado de meu próprio trabalho, exceto onde referência explícita é feita ao trabalho de outros, e não foi submetida para outra defesa nesta nem em outra universidade. Parte deste trabalho já foi publicado e este texto é uma composição adaptada de artigos publicados pela autora. Abaixo listo a relação de cada trabalho publicado até a data desta defesa e o conteúdo relacionado a este documento:

- Araujo, J. A. S., Santos, H. G., Gendron, B., Jena, S. D, Brito, S. S. e Souza D. S. *Strong Bounds for Resource Constrained Project Scheduling: Preprocessing and Cutting Planes*. Computers & Operations Research. 113 (2020) 104782. Qualis A1. Conteúdo apresentado em todos os capítulos.

- Araujo, J. A. S. e Santos, H. G. *Separation Strategies to Chvátal-Gomory Cuts for the Resource Constrained Project Scheduling Problems: a Computational Study*. The 17th International Conference on Computational Science and Its Applications - ICCSA2017. (2017) 452-466. Qualis B1. Conteúdo apresentado no capítulo 5.

- Araujo, J. A. S., Santos, H. G., Baltar, D. D., Toffolo, T. A. M. e Wauters, T. *Neighborhood Composition Strategies in Stochastic Local Search*. 10th International Workshop on Hybrid Metaheuristics - HM2016. (2016) 118-130. Qualis B3. Conteúdo apresentado no capítulo 4.

- Soares, J. A., Santos, H. G., Baltar, D. D. e Toffolo, T. A. M. *LAHC applied to The Multi-Mode Resource -Constrained Multi-Project Scheduling Problem*, in 7th Multidisciplinary International Conference on Scheduling : Theory and Applications - MISTA 2015. (2015) 905-908. Conteúdo apresentado no capítulo 4.

Janniele Aparecida Soares Araujo

Agradecimentos

Primeiramente, eu agradeço a Deus pelas vitórias que obtive, por renovar minhas forças e me dar proteção sempre. Agradeço ao Divino Espírito Santo por iluminar a minha inteligência em todos os momentos.

Agradeço principalmente ao meu orientador Prof. Dr. Haroldo Gambini Santos pela impecável orientação, pelas oportunidades concedidas e por sempre ter acreditado em mim e em meu potencial. Agradeço ainda pelas valiosas contribuições, pela atenção, por todo o tempo dedicado e pela transmissão de conhecimentos, tudo foi fundamental para a minha formação acadêmica.

Je remercie les professeurs Bernard Gendron et Sanjay Dominik Jena pour leurs conseils alors que j'étais au Centre Interuniversitaire de Recherche sur les Reseaux d'Entreprise, la Logistique et le Transport (CIRRELT), Université de Montréal, Canada.

Agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pela bolsa concedida e por promover a internacionalização da minha pesquisa. Agredeço a Universidade Federal de Ouro Preto (UFOP), por prover, enducação pública, gratuita e de qualidade e especialmente por me conceder a oportunidade de obter o título de Doutora em Ciência da Computação.

Não poderia deixar de agradecer aos meus pais, José Geraldo Soares e Rosângela Soares Pessoa, e ao meu marido Ítalo Campos Araújo, que me deram amor, força e coragem para seguir em frente durante essa caminhada.

Por fim, agradeço a todos que me ajudaram direta ou indiretamente neste trabalho.

Prefácio

Que os vossos esforços desafiem as impossibilidades lembrai-vos de que as grandes coisas do homem foram conquistadas do que parecia impossível.

Charles Chaplin

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

$\mathcal{CG}$	Conflict Graph
B&B	Branch and Bound
B&C	Branch and Cut
C1M	Change One Mode
C2M	Change Two Modes
C3M	Change Three Modes
C4M	Change Four Modes
CG	Chvátal-Gomory Cut
CL	Clique Cuts
CPD	Critical Path Duration
CPM	Critical Path Method
CPP	Compact Project on Percentage
CP	Constraint Programming
CSP	Compact Subsequent Projects
CV	RCPSP Knapsack Cover Cuts
FBI	Forward-Backward Improvement
GRC	Greedy Randomized Constructive
ICAPS	International Conference on Automated Planning and Scheduling

ILS Iterated Local Search

IP Integer Programming

ISJ Invert Sequence of Jobs

JSSP Job-Shop Scheduling Problems

LAHC Late Acceptance Hill-Climbing

LCSP Labor Constrained Scheduling Problem

LCV Lifted RCPSP Knapsack Cover Cuts

LPR Lifted Precedence Based Cuts

LP Linear Programming

MILP Mixed-Integer Linear Pro-gramming

MISTA Multidisciplinary International Scheduling Conference: Theory & Applications

MMLIB Multi-Mode Library

MMRCMPSP Multi-Mode Resource Constrained Multi-Project Scheduling Problem

MMRCPSP Multi-Mode Resource Constrained Project Scheduling Problem

NLP Nonlinear Programming

OH Odd-Holes Cuts

OJ Offset Job

OODDT On/Off Disaggregated Discrete-Time

OODT On/Off Discrete Time

OOE On/Off Event-based

OP Offset Project

PDDT Pulse Disaggregated Discrete-Time

PDT Pulse Discrete Time

PR Precedence Based Cuts

PSO Particle Swarm Optimization

PSPLIB Project Scheduling Problem Library

RCPSP Resource Constrained Project Scheduling Problem

SAT Satisfiability Solving

SA Simulated Annealing

SCGC Strengthened Chvátal-Gomory Cut

SCIP Solving Constraint Integer Programs

SCP Swap and Compact Two Projects

SDDT Step Disaggregated Discrete-Time

SDT Step Discrete Time

SGS Schedule Generation Scheme

SIJW Successive Insertions of a Job in a Window

SLS Stochastic Local Search

SMRCPSP Single-Mode Resource Constrained Project Scheduling Problem

SPE Squeeze Project on Extreme

SSJW Successive Swap of a Job in a Window

STJ Swap Two Jobs

TMS Total Makespan

TPD Total Project Delay

URCSP Uniform Resources Constrained Scheduling Problem

VNS Variable Neighborhood Search

Chapter 1

Introduction

Combinatorial optimization problems have received continuous interest from researchers in computer science, mathematics and operational research. Such problems often occur in academia, industry and engineering. The non-preemptive Resource Constrained Project Scheduling Problem (RCPSP) represents an interesting and challenging class of combinatorial optimization problems, classified as $\mathcal{NP}$-hard (Blazewicz et al., 1983; Garey and Johnson, 1979) from a theoretical point of view and very hard to solve in practice. In this problem, a project is composed of a set of jobs, where each job has precedence relations and requires time and resources to be executed. Each job can have one or more execution modes with different durations and resource consumption in each mode; also, jobs can have dependency between them. Some variants of RCPSP also consider issues that go beyond the dependency between jobs and multiple modes of execution, such as the use of renewable and non-renewable resources and the sharing of some resources between different projects.

These problems cover a wide range of applications, such as particle therapy for cancer treatment in healthcare (Riedler et al., 2017), civil engineering (Liu and Wang, 2008), manufacturing and assembly of large products (Liu et al., 2014), as well as development and launching of complex systems (Demeulemeester and Herroelen, 2002). In academia, the interest in this class of problems can be observed considering the amount of books (Artigues et al., 2013; Burke and Pinedo, 2017; Demeulemeester and Herroelen, 2002; Schwindt and Zimmermann, 2015), the scientific journal Journal of Scheduling[1], regular

[1] `https://www.springer.com/journal/10951`

conferences such as MISTA[2] and ICAPS[3] and public benchmark datasets devoted to this class of problems such as the Project Scheduling Problem Library (PSPLIB) (Kolisch and Sprecher, 1996), the MISTA Challenge (Wauters et al., 2016) datasets and the MMLIB (Van Peteghem and Vanhoucke, 2014) datasets. We present in Chapter 2 the problem of study and its definitions, as well as the benchmark datasets characteristics.

Comprehensive reviews on RCPSPs can be found, for example, in Artigues et al. (2008), Demeulemeester and Herroelen (2002), Schwindt and Zimmermann (2015), and Habibi et al. (2018). A background and related works for understanding better this thesis are presented in Chapter 3. In the literature, incomplete and complete algorithms were proposed to find solutions for the RCPSPs. Heuristics are incomplete algorithms: they produce valid feasible solutions and the corresponding upper bounds, but it is not possible to be sure that the generated solution is the optimal one. Several heuristic methods, such as Tabu Search (Kochetov and Stolyar, 2003; Nonobe and Ibaraki, 2002; Skowroński et al., 2013), Monte Carlo (Asta et al., 2014; Chen and Zhang, 2012; Walędzik and Mańdziuk, 2017), Genetic Algorithms (Alcaraz et al., 2004; Hartmann, 2002), Path-relinking (Berthaut et al., 2018; Muritiba et al., 2018) and others, have been proposed for the RCPSPs. Unlike heuristics, exact methods are complete algorithms that produce upper and lower bounds to find provably optimal solutions. Mixed-Integer Linear Programming (MILP) methods (Zhu et al., 2006), as well as Constraint Programming (CP) (Demassey et al., 2005), Satisfiability Solving (SAT) (Coelho and Vanhoucke, 2011; Vanhoucke and Coelho, 2016) and hybrid versions (Schnell and Hartl, 2017), are complete algorithms and have also been proposed for the RCPSPs.

The generation of good upper bounds is important to improve the performance of exact methods. Our approach, presented in Chapter 4, produces upper bounds using Stochastic Local Search (SLS) (Hoos and Stützle, 2005). We propose and computationally evaluate neighborhood composition strategies to obtain tight upper bounds. New best-known solutions were obtained.

As the most significant contribution of this thesis, in Chapter 5, we propose automatic MILP reformulation strategies for the RCPSPs and its variants. We propose a new preprocessing technique to improve resource-related constraints by strengthening their coefficients using known information about different renewable, non-renewable resources and precedence constraints. We also propose a cutting plane algorithm employing five different cut separation algorithms. Conflict-based cuts such as cliques and odd-holes

[2] `http://www.schedulingconference.org/`
[3] `http://icaps-conference.org/`

are generated considering an implicit dense conflict graph, which is updated at each iteration considering optimality and feasibility conditions. This conflict graph is also used in a MILP separation of the Chvátal-Gomory cuts to produce stronger inequalities. Furthermore, new lifted versions of the well-known disaggregated precedence cuts and cover cuts are separated. The contribution to the linear relaxation bound provided by each cut family is examined on experiments on a large set of benchmark instances of the three problem variants considered in this thesis. Overall, stronger bounds were obtained, allowing a state-of-the-art MILP solver to find provably optimal solutions for 754 previously open instances, which was not possible using the original linear programming formulations. The computational results obtained from the experiments executed, as well as, our conclusions and future works are presented respectively in Chapters 6 and 7.

The importance of guaranteeing the quality of the solutions and, simultaneously, reducing the processing time for the production of such solutions allowed us to outline the objective of this doctoral thesis, which consists of developing improved methods to solve the aforementioned problems, to improve the quality of the best-known solutions and the lower bounds available. In this chapter, we provide a brief introduction to the addressed problem, followed by a discussion about our motivations and our contributions.

1.1 Motivation and Contributions

Considering the digital libraries of problems (PSPLIB, MISTA and MMLIB), several instances are still open. Of the 5688 instances considered in this work 63% are still open. The present thesis is motivated by the fact that to find the optimal solutions for these instances constitutes a challenge for the state-of-the-art methods and the vast applicability of these models. Our work also includes a very comprehensive experimental analysis of the methods developed.

The main result of this thesis is the design and development of new and improved state-of-the-art algorithms for the resolution of the RCPSPs. It was possible to reduce to 49% the number of open instances in the literature. These algorithms, proposals and experimental results were documented and disseminated to the scientific community through publications: firstly with conference articles and in sequence with a scientific paper in an indexed journal with recognized relevance in the area.

The algorithms are implemented in the context of systematic MILP search methods. MILP algorithms, have the advantage of providing a certifiably optimal solution in finite time.

Chapter 2

Resource Constrained Project Scheduling Problems

Resource Constrained Project Scheduling Problems (RCPSPs) are of significant academic and practical importance. From the theoretical point of view, RCPSPs are $\mathcal{NP}$-hard (Blazewicz et al., 1983; Garey and Johnson, 1979). Some practical applications were presented in the introduction of this thesis.

Briefly, the RCPSP seeks to carry out scheduling or programming to a project, in which each job must be allocated at a particular time period over a horizon and assigned in a specific mode (way of carrying out a job), respecting the constraints imposed for the problem.

2.1 Formal Definition

One of the first formulations for the problem was proposed by Pritsker et al. (1969). Therefore, the conceptual formulation of the RCPSP is given. Consider the description of the input data for the model shown in this section:

$\mathcal{J}$ set of all jobs;

$\mathcal{R}$ set of renewable resource;

$\mathcal{T}$ set of time periods in the planning horizon;

$\boldsymbol{\mathcal{T}_j}$ set of time periods in the planning horizon for job $j \in \mathcal{J}$ after preprocessing;

$\boldsymbol{\mathcal{S}}$ set of direct precedence relationships between two jobs $(j,l) \in \mathcal{J} \times \mathcal{J}$;

$\boldsymbol{q_{rj}}$ required amount of resource r to execute job $j \in \mathcal{J}$;

$\boldsymbol{\breve{q}_r}$ available amount of renewable resource $r \in \mathcal{R}$;

$\boldsymbol{a_p}$ artificial job which represents the end of the project.

This basic formulation contains only one type of binary decision variable, x_{jt}, indexed by job and time, this formulation is known as discrete-time (DT). Decision variables are defined so that x_{jt} is 1 if job j starts at time t and 0 otherwise.

Minimize:

$$\sum_{t \in \mathcal{T}_{a_p}} t \cdot x_{a_p t} \tag{2.1}$$

Subject to:

$$\sum_{t \in \mathcal{T}_j} x_{jt} = 1 \qquad \forall j \in \mathcal{J} \tag{2.2}$$

$$\sum_{t \in \mathcal{T}_j} t \cdot x_{jt} \leq \sum_{t \in \mathcal{T}_j} (t \cdot x_{lt} - d_j) \qquad \forall (j,l) \in \mathcal{S} \tag{2.3}$$

$$\sum_{j \in J} \sum_{h=max(t,\mathcal{T}_j)}^{min(t+d_j-1,\mathcal{T}_j)} q_{rj} \cdot x_{jh} \leq \breve{q}_r \qquad \forall r \in \mathcal{R}, \forall t \in \mathcal{T} \tag{2.4}$$

$$x_{jt} \in \{0,1\} \qquad \forall j \in \mathcal{J}, \forall t \in \mathcal{T}_j \tag{2.5}$$

The objective function (2.1) in the mathematical formulation minimizes the time of artificial job completion. Constraints (2.2) ensure that each job executes only once. Constraints (2.3) ensure the precedence relationships between jobs. Constraints (2.4) guarantee the feasibility w.r.t. renewable resources consumption. Finally, constraints (2.5) specify the decision variables as binary.

Kolisch and Sprecher (1996) proposed an approach based on the idea of Pritsker et al. (1969), with a single type of binary decision variable, controlling the finishing time instead of the starting time; an additional index has been added to control how to execute each job. Consider here the set $\mathcal{M}_j$ of modes available for job $j \in \mathcal{J}$. By definition, x_{jmt} is equal to 1 if job j runs in m mode and ends in time t and x_{jmt} is

equal to 0 otherwise. These formulation models constraints renewable resources and non-renewable resources.

2.2 Problem Variants

In this thesis, we consider the following problem variants of the RCPSP, from the most specific one to the most generalized version:

SMRCPSP: single-mode resource constrained project scheduling problem;

MMRCPSP: multi-mode resource constrained project scheduling problem;

MMRCMPSP: multi-mode resource constrained multi-project scheduling problem.

In the literature other variants of the problem considering time lags, time dependent capacities, flexible resource constrained and uncertain job durations are presented: Resource Constrained Project Scheduling Problem with Minimal and Maximal Time Lags (RCPSP/max)[1], Multi-Mode Resource Constrained Project Scheduling Problem with Minimal and Maximal Time Lags (MRCPSP/max)[2], Resource Investment Problem with Minimal and Maximal Time Lags (RIP/max)[3], Resource Constrained Project Scheduling Problem With Time-Dependent Resource Capacities (RCPSP/TDRC)[4], Flexible Resource Constrained Scheduling Problem (FRCPSP)[5]. Other variations with more specific modifications can also be found.

The SMRCPSP is the simplest variant and involves only one processing mode for each job. A feasible solution consists of the assignment of jobs at specific time periods over a planning horizon, respecting precedence and resource usage constraints. The resources in the SMRCPSP are renewable at each time period.

In the MMRCPSP, it is possible to choose between different job processing modes, each of them having different durations and consuming different amounts of resources.

[1] http://www.wiwi.tu-clausthal.de/en/abteilungen/produktion/forschung/schwerpunkte/project-generator/rcpspmax/

[2] http://www.wiwi.tu-clausthal.de/en/abteilungen/produktion/forschung/schwerpunkte/project-generator/mrcpspmax/

[3] http://www.wiwi.tu-clausthal.de/de/abteilungen/produktion/forschung/schwerpunkte/project-generator/ripmax/

[4] http://www.om-db.wi.tum.de/psplib/newinstances.html

[5] http://www.om-db.wi.tum.de/psplib/getdatafrcpsp.cgi

Two types of resources are available: the renewable resources at each time period and the non-renewable ones available for the entire project execution. While the use of renewable resources only impacts the delay/speedup of the projects, the use of non-renewable resources can produce infeasible solutions.

A generalization of the previous problem variants to handle multiple projects and global renewable resources is the MMRCMPSP. While the previous two problem variants, in their objective (minimization) functions, consider the makespan, i.e., the total length of the schedule to finish all jobs, an important modeling feature of this generalization is an additional objective: the project delays from the Critical Path Duration (CPD). The CPD is a theoretical lower bound on the earliest finishing time ($\check{e}_p^f$) of project p, computed by the Critical Path Method (CPM) (Kelley Jr and Walker, 1959) and disregarding any resource constraints.

Figure 2.1 illustrates the example instance $j102\_4.mm$[6], a small instance for the MMRCPSP variant.

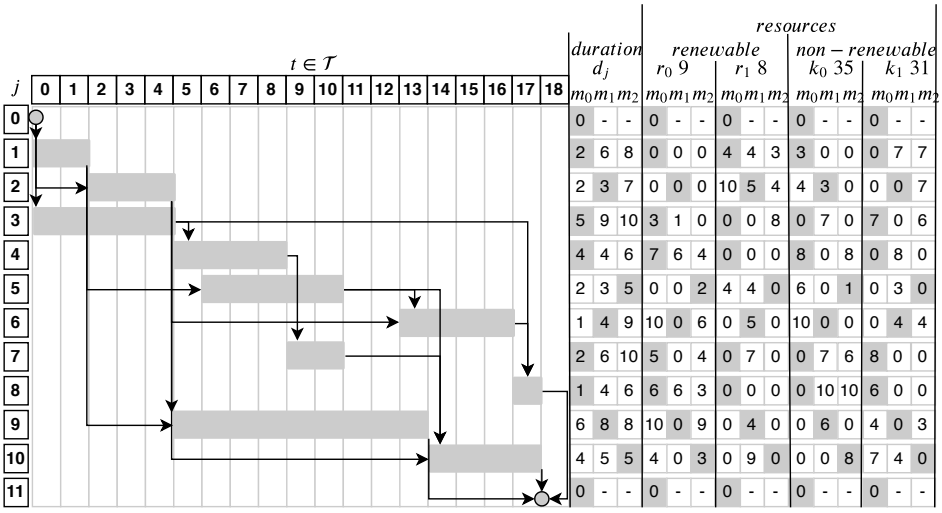

Figure 2.1: An optimal solution for instance $j102\_4.mm$ and its characteristics

The instance illustrated in Figure 2.1 has twelve jobs. The first and the last are artificial jobs representing the start and the end of the project. Each job j can be processed in three different modes m. Artificial jobs 0 and 11 have only one execution mode that does not consume resources and have duration 0; thus columns m_1 and m_2 are filled with "-". Two renewable resources $\{r_0, r_1\}$ with capacities $\{9, 8\}$ and two non-renewable resources $\{k_0, k_1\}$ with capacities $\{35, 31\}$ are available. This figure provides

[6]http://www.om-db.wi.tum.de/psplib/files/j10.mm.tgz

information about resource consumption and duration of job j processing on mode m. The figure shows the characteristics of this instance and one optimal solution. We refer to this example instance to explain the cutting planes presented in the next chapters. The Gantt chart in the figure shows the starting time allocation and duration of jobs for the optimal solution found for this single project. Arcs represent the precedence relationship between jobs. Values emphasized in grayscale represent the active modes for this solution. The CPD corresponds to a value of 15. Note that the earliest starting time ($\breve{e}_j^s$) of job 5 corresponds to 2, but due to the availability of resources at this time period, its allocation had to be postponed. The makespan is 18 time units.

2.3 Objective Function

The most common objective function for the RCPSP is the makespan (Artigues et al., 2008; Demeulemeester and Herroelen, 2002; Kolisch and Sprecher, 1996; Pritsker et al., 1969; Zhu et al., 2006) minimizing the total schedule duration required to finish all jobs. Wauters et al. (2016) proposed a hierarchical objective to minimize the Total Project Delay (TPD) and the total makespan (TMS). The former, denoted formally in Eq.(2.6), is the main objective. The latter, given in Eq.(2.7), is a tiebreaker.

$$TPD = \sum_{p \in P} PD_p \tag{2.6}$$

$$TMS = \max_{p \in \mathcal{P}} f_p - \min_{p \in \mathcal{P}} \sigma_p \tag{2.7}$$

The project delay (PD_p) for a project $p \in \mathcal{P}$ is defined as the difference between its CPD_p (which does not consider any resource constraints), and its makespan $MS_p = f_p - \sigma_p$, the actual project duration taking into consideration resource constraints, the finishing time f_p and the release date σ_p of this project.

$$PD_p = MS_p - CPD_p \tag{2.8}$$

To work with all three problem variants in a unified way, we always report our results considering the TPD, which is generic enough to handle all problem variants considered in this thesis.

2.4 Constraints

The RCPSP involves the allocation of jobs at time periods on a horizon and the assignment to a mode, taking into account some constraints that must be satisfied.

In the problems addressed in this work, the following constraints are considered:

- precedence constraint: each job j can not start until all predecessor jobs have been finalized;
- resources constraint: resource limits must be respected.

The resources are classified as follows:

- renewable resources: the available quantities are renewed from period to period (hour, day, week, month), availability per period is constant - example: labor, machines;
- non-renewable resources: the consumption of non-renewable resources is limited for the whole project - example: money, energy and raw materials;
- global resources: in the case of the multi-project problem, global resources are still considered, which are renewable and shared among all projects.

2.5 Benchmark Datasets Characteristics

Several public benchmark datasets were established, including realistic, but computationally challenging problem instances to facilitate the comparison of algorithmic improvements. In 1996 a public and well-known repository of benchmark datasets, devoted to this class of problems, was established: the Project Scheduling Problem Library - PSPLIB (Kolisch and Sprecher, 1996). In 2013 a new dataset based on PSPLIB instances, considering multiple projects, emerged from the MISTA Challenge (Wauters

et al., 2016). Then, in 2014, another dataset based on PSPLIB was proposed for the multi-mode scheduling problem called MMLIB (Van Peteghem and Vanhoucke, 2014). For all problem variants, many of those instances are still open. In particular, by the time of writing this thesis, we found that 729 of the PSPLIB instances, 2842 of the MMLIB instances, and 27 of the MISTA instances, were still open[7].

We now define in Table 2.1 the benchmark datasets used in this work based on (Schnell and Hartl, 2017; Toffolo et al., 2016; Zhu et al., 2006), composed by instances from the PSPLIB, MISTA and MMLIB repositories. For each library, the total number of problem instances defined for each problem variant, the dataset name, the number of instances used in our benchmark datasets, and the number of instances for which optimality has not yet been proven in the literature.

Table 2.1: Benchmark datasets

library	variant/num. inst.	group	in dataset	open
PSPLIB	SMRCPSP/2040	J60	79	79
		J90	105	105
		J120	514	514
	MMRCPSP/3931	J30	640	31
MMLIB	MMRCPSP/4300	J50	540	118
		J100	540	176
		Jall50+	1620	1178
		Jall100+	1620	1370
MISTA	MMRCMPSP/30	A	10	7
		B	10	10
		X	10	10

These are the most known and studied instance datasets; the format used by PSPLIB allows a large number of real problems to be represented. Kolisch et al. (1995) considered two essential issues to control the characteristics of the instances generated for the PSPLIB; they are: how restricted is the relationship between resource availability and consumption and what is the precedence relation factor between jobs. There are procedures for the construction that takes into account: constraints on the network topology, a resource factor that reflects the density of the consumption, and a resource strength

[7] 1339 considering the PSPLIB website and 30 considering the MISTA website. There is no information concerning the optimality of the solutions obtained by Schnell and Hartl (2017) and Toffolo et al. (2016).

that measures the availability of resources.

Table 2.2 presents the basic characteristics of the instances from all datasets: the minimum, maximum and average values of the number of projects $|\mathcal{P}|$, jobs $|\mathcal{J}_p|$, the average number of modes per job $|\overline{\mathcal{M}_j}|$ and average duration jobs per modes $\overline{d_{jm}}$. We can observe that the benchmark consists of instances with varying characteristics, some with more jobs such as J120, J100, Jall100+, A, B, X datasets, and some with different modes and duration of executing it, especially Jall50+, Jall100+ datasets.

Table 2.2: Basic characteristics of instances from the benchmark datasets {min, max, avg}

group	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{J}_p\rvert$	$\lvert\overline{\mathcal{M}_j}\rvert$	$\overline{d_{jm}}$
J60	{1.00, 1.00, 1.00}	{62.00, 62.00, 62.00}	{1.00, 1.00, 1.00}	{4.92, 6.29, 5.40}
J90	{1.00, 1.00, 1.00}	{92.00, 92.00, 92.00}	{1.00, 1.00, 1.00}	{4.97, 5.87, 5.45}
J120	{1.00, 1.00, 1.00}	{122.00, 122.00, 122.00}	{1.00, 1.00, 1.00}	{4.64, 6.10, 5.40}
J30	{1.00, 1.00, 1.00}	{32.00, 32.00, 32.00}	{2.88, 2.88, 2.88}	{4.64, 6.01, 5.38}
J50	{1.00, 1.00, 1.00}	{52.00, 52.00, 52.00}	{2.92, 2.92, 2.92}	{4.72, 6.16, 5.41}
J100	{1.00, 1.00, 1.00}	{102.00, 102.00, 102.00}	{2.96, 2.96, 2.96}	{4.91, 6.06, 5.46}
Jall50+	{1.00, 1.00, 1.00}	{52.00, 52.00, 52.00}	{2.92, 8.69, 5.81}	{4.86, 17.03, 12.93}
Jall100+	{1.00, 1.00, 1.00}	{102.00, 102.00, 102.00}	{2.96, 8.84, 5.83}	{4.93, 16.62, 12.89}
A	{2.00, 10.00, 6.10}	{24.00, 320.00, 144.20}	{2.67, 2.88, 2.80}	{4.90, 5.62, 5.26}
B	{10.00, 20.00, 15.50}	{120.00, 640.00, 342.00}	{2.67, 2.88, 2.79}	{5.15, 5.46, 5.33}
X	{10.00, 20.00, 15.50}	{120.00, 640.00, 342.00}	{2.67, 2.88, 2.79}	{4.94, 5.38, 5.24}

Table 2.3 shows information of the number of renewable $|\mathcal{R}|$ and non-renewable $|\mathcal{K}|$ resources and the average availability ($\overline{\breve{q}^r}$,$\overline{\breve{q}^k}$). For each dataset, we present the minimum, maximum and average values. We can see that datasets from SMRCPSP (J60, J90 and J120) do not deal with non-renewable resources; all other datasets deal with this type of resource. Ideally, instances that have a larger number of jobs could have a higher resource availability; for non-renewable resources, this feature could ensure the feasibility of solutions to these problems. However, when availability is lower, either to renewable or non-renewable resources, these instances can become more restricted concerning the resource consumption by all jobs. It could be easier to solve this instance if it did not have such a large number of jobs. These restrictive factors and the small number of jobs can enable its optimal solution with suitable exact methods.

Table 2.3: Basic characteristics about resources of the instances from benchmark datasets {min, max, avg}

group	$\lvert\mathcal{R}\rvert$	$\overline{\breve{q^r}}$	$\lvert\mathcal{K}\rvert$	$\overline{\breve{q^k}}$
J60	{4.00, 4.00, 4.00}	{15.25, 21.00, 18.69}	{-, -, -}	{-, -, -}
J90	{4.00, 4.00, 4.00}	{17.50, 24.25, 20.39}	{-, -, -}	{-, -, -}
J120	{4.00, 4.00, 4.00}	{12.00, 57.00, 28.15}	{-, -, -}	{-, -, -}
J30	{2.00, 2.00, 2.00}	{11.00, 57.50, 25.76}	{2.00, 2.00, 2.00}	{49.50, 195.50, 150.83}
J50	{2.00, 2.00, 2.00}	{15.50, 102.00, 40.22}	{2.00, 2.00, 2.00}	{69.00, 319.50, 201.58}
J100	{2.00, 2.00, 2.00}	{10.50, 92.50, 37.43}	{2.00, 2.00, 2.00}	{145.50, 629.50, 404.55}
Jall50+	{2.00, 4.00, 3.00}	{14.25, 384.00, 53.66}	{2.00, 4.00, 3.00}	{218.00, 953.50, 639.72}
Jall100+	{2.00, 4.00, 3.00}	{16.00, 600.50, 53.90}	{2.00, 4.00, 3.00}	{438.00, 1862.50, 1273.24}
A	{2.00, 11.00, 5.30}	{11.50, 42.00, 22.82}	{4.00, 20.00, 12.20}	{44.90, 175.40, 112.13}
B	{2.00, 21.00, 11.80}	{10.00, 28.45, 19.73}	{20.00, 40.00, 31.00}	{44.80, 154.83, 100.18}
X	{2.00, 21.00, 14.20}	{12.50, 28.36, 19.95}	{20.00, 40.00, 31.00}	{47.60, 167.00, 104.43}

Tables 2.4 display average information about the relationship $\left(\frac{\overline{\breve{q^r}}}{\overline{q_{mr}}}, \frac{\overline{\breve{q^k}}}{\overline{q_{mk}}}\right)$ between the number of available renewable and non-renewable resources ($\overline{\breve{q^r}}$,$\overline{\breve{q^k}}$) and the consumption ($\overline{q_{mr}}$,$\overline{q_{mk}}$) indicating how restricted the instances from these dataset are.

Table 2.4: Basic characteristics about availability-consumption resources of instances from the benchmark datasets {min, max, avg}

group	$\overline{q_{mr}}$	$\frac{\overline{\breve{q^r}}}{\overline{q_{mr}}}$	$\overline{q_{mk}}$	$\frac{\overline{\breve{q^k}}}{\overline{q_{mk}}}$
J60	{5.18, 5.74, 5.50}	{2.94, 3.67, 3.39}	{-, -, -}	{-, -, -}
J90	{5.00, 6.25, 5.51}	{2.84, 4.34, 3.71}	{-, -, -}	{-, -, -}
J120	{5.07, 5.94, 5.49}	{2.23, 10.40, 5.13}	{-, -, -}	{-, -, -}
J30	{4.94, 6.30, 5.54}	{1.91, 9.85, 4.67}	{4.74, 6.56, 5.52}	{9.26, 33.63, 27.29}
J50	{4.49, 5.90, 5.28}	{2.91, 18.88, 7.62}	{4.39, 6.03, 5.19}	{14.48, 56.97, 38.55}
J100	{2.60, 5.67, 3.20}	{3.49, 31.69, 11.74}	{4.55, 5.75, 5.20}	{31.30, 113.93, 77.30}
Jall50+	{2.41, 17.25, 7.41}	{3.13, 144.91, 10.02}	{4.64, 17.34, 12.70}	{40.47, 59.28, 50.41}
Jall100+	{2.48, 16.05, 4.69}	{4.15, 222.06, 14.97}	{4.88, 16.80, 12.64}	{84.73, 118.96, 100.80}
A	{5.15, 5.60, 5.42}	{2.05, 8.16, 4.24}	{5.38, 5.88, 5.62}	{8.17, 31.41, 19.99}
B	{5.18, 5.66, 5.53}	{1.82, 5.07, 3.57}	{5.45, 5.69, 5.57}	{8.00, 28.43, 18.05}
X	{5.42, 5.79, 5.55}	{2.16, 5.09, 3.61}	{5.42, 5.68, 5.55}	{8.54, 30.05, 18.85}

Instances with the same number of jobs that have a high consumption of resources

and low availability are more restricted than others. We can observe by analyzing the minimum values, that there are really restricted instances with factors lower than 3.00 if we consider renewable resources. For instances with 60 and 90 jobs, it can be seen that the average relation between renewable resources availability and consumption is narrower for some sets of parameter instances[8], being able to run a maximum of 5 jobs per time period. Also, by analyzing the minimum values of availability-consumption for non-renewable resources, we have instances with small factors such as instances from J30, J50, A, B datasets.

The two box and whisker plots, also called box plots, in Figure 2.2 demonstrate the relationship between availability and consumption of renewable and non-renewable resources for all instances from the benchmark datasets.

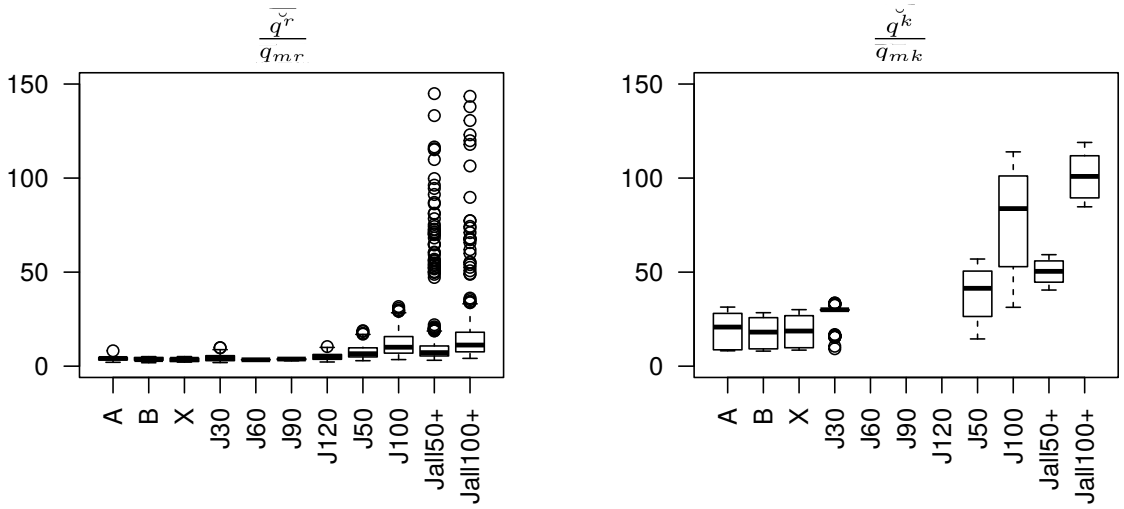

Figure 2.2: Characteristics of the relationship between availability and consumption of resources

There are instances of datasets Jall50+ and Jall100+ with high factors when it comes to the availability-consumption of renewable resources. This specific feature of some outliers can make it difficult for mathematical models to solve the problem to optimality. Instances with larger numbers of jobs from MMLIB have a higher availability of non-renewable resources. Note that instances from MISTA have a large number of jobs, but a small number of jobs per project, and the non-renewable resources are not

[8]for example, the factors availability-consumption of renewable resources from set instances $j905, j909, j9021, j9025, j9037, j9041, j9045, j609, j6025, j6029, j6041$ and $j6045$ are less than 4

shared between projects.

Table 2.5 shows information about the average number of starting ($\overline{|\breve{s}|}$) and finishing ($\overline{|\breve{f}|}$) jobs connected to the artificial jobs and the average number of predecessors per job ($\overline{|\breve{p}_j|}$). The larger number of starting and finishing jobs connected to the artificial jobs, plus the larger number of the predecessors, can difficult the solution of the models. We can observe that instances from MMLIB (J50, Jall50+, J100 and Jall100+) have a large factor concerning the network connections.

Table 2.5: Basic characteristics about network connections of instances from the benchmark datasets {min,max,avg}

group	$\overline{\lvert\breve{s}\rvert}$	$\overline{\lvert\breve{f}\rvert}$	$\overline{\lvert\breve{p}_j\rvert}$
J60	{3.00, 3.00, 3.00}	{3.00, 3.00, 3.00}	{1.50, 1.50, 1.50}
J90	{3.00, 3.00, 3.00}	{3.00, 3.00, 3.00}	{1.50, 1.50, 1.50}
J120	{3.00, 3.00, 3.00}	{3.00, 3.00, 3.00}	{1.50, 2.11, 1.76}
J30	{3.00, 3.00, 3.00}	{3.00, 3.00, 3.00}	{1.81, 1.81, 1.81}
J50	{1.00, 22.00, 10.72}	{2.00, 22.00, 10.47}	{2.88, 5.21, 3.99}
J100	{5.00, 41.00, 19.02}	{4.00, 36.00, 18.36}	{4.48, 7.97, 6.48}
Jall50+	{1.00, 23.00, 10.41}	{2.00, 23.00, 10.49}	{2.67, 5.21, 3.95}
Jall100+	{3.00, 41.00, 18.48}	{4.00, 37.00, 18.60}	{4.48, 7.97, 6.50}
A	{3.00, 3.00, 3.00}	{3.00, 3.00, 3.00}	{1.50, 1.82, 1.72}
B	{3.00, 3.00, 3.00}	{3.00, 3.00, 3.00}	{1.50, 1.82, 1.72}
X	{3.00, 3.00, 3.00}	{3.00, 3.00, 3.00}	{1.50, 1.82, 1.72}

Figure 2.3 presents three box plots that show the characteristics of the network connections for all instances from the benchmark datasets. Instances from MMLIB have a higher number of starting and finishing jobs that are directly connected to the artificial ones; also, they have a higher number of predecessors per job. Instances from PSPLIB and MISTA have low values that do not vary so much.

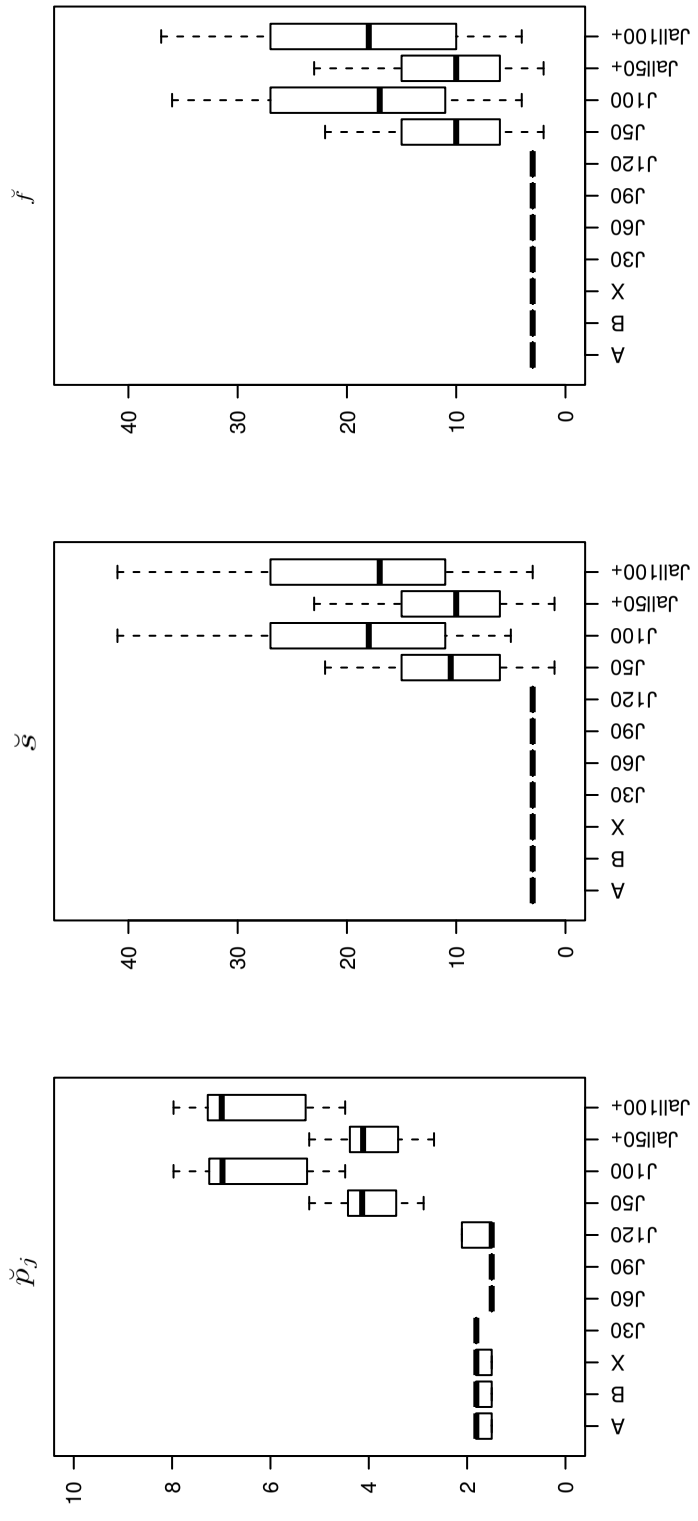

Figure 2.3: Characteristics of the network connections

Chapter 3

Background and Literature Review

Mixed-Integer Linear Programming (MILP) (Cook and Koch, 2008; Jünger, Naddef, Pulleyblank, Rinaldi, Liebling, Nemhauser, Reinelt and Wolsey, 2010) is the main technique used for the exact resolution of difficult problems in combinatorial optimization. The advances made in the last decades in this area allowed the exact resolution of problems considered computationally intractable previously. Progress in this area, however, has not occurred uniformly for different problems. This chapter presents concepts and techniques for understanding and solving the problems under study. Also, we present a literature review.

3.1 Background

Quantitative problems can be expressed mathematically through models. There are methods able to solve these problems, the most common techniques used are those from the discipline of operational research, a branch of human knowledge highlighted by its mathematical modeling techniques (Churchman et al., 1957; Luenberger and Ye, 2016; Wagner, 1969). Shapiro (1993) classifier several categories of mathematical programming models: linear programming, network optimization, mixed integer programming, nonlinear programming, dynamic programming, multiple criteria optimization, and stochastic programming. Mathematical programming is of great use for solving optimization problems (Bradley et al., 1977) including for scheduling applications (Shapiro, 1993).

The solution techniques of mathematical programming usually are grouped in the following subareas:

- Linear Programming (LP): the variables are continuous and present linear behavior, both in relation to the constraints, and the objective function (Dantzig, 1998);
- Integer Programming (IP): some or all of the variables are restricted to be integers, IP (Wolsey, 1998a) is an extension of LP;
- Nonlinear Programming (NLP): they can have continuous and integer variables, but the objective function or the constraints may be non-linear (Bazaraa, 2013).

In optimization, the mathematical models represent a particular decision problem using a set of equations or mathematical expressions (Bradley et al., 1977). The effectiveness of those techniques can be measured by a mathematical function of the decision variables, called the objective function. Equations and inequalities express the constraints that correspond to the limitations of the values of the variables.

IP allows modeling and solving a wide range of problems, including production planning (Pochet and Wolsey, 2006), protein structure prediction and drug design (Xu et al., 2003), communication networks project (Koster et al., 2008), among others. The modeling power, as well as the availability of solver programs (ILO, 2008), including some open-source (Linderoth and Ralphs, 2005; Lougee-Heimer, 2003), makes IP a handy tool for operational research applicators in solving countless combinatorial optimization problems.

Even thought some IP is NP-Hard (Garey and Johnson, 1979), in which the difficult depends of the structure of the problem, the worst-case analysis, however, did not prevent progress in the last decades in the development of computational algorithms capable of consistently solving several relevant instances of various problems in combinatorial optimization problems. Advances in Linear Programming (Johnson et al., 2000b) and the study of the polyhedral theory (Aardal and van Hoesel, 1996, 1999) allowed IP to be applied successfully in large problems. Problems with millions of binary variables have been solved. These results, however, are related to specific problems. For most of the problems with relevance in industry and academia, the results still need to be improved.

Cook (2019) has established in his work a wish list of research directions with 5 topics. The first topic contemplates the importance of improving the simplex method so that re-optimizing after adding cuts be more faster, a problem we face in this thesis. The second topic is the language of algorithms; the author understands that the development and adoption of a more nuanced way of expressing achievements in the analysis of algorithms are something fundamental. The third topic is about understanding heuristic algorithms;

he said that heuristic algorithms are widely used in operations research and many other areas, but not understood. The fourth topic is about the analysis of exact algorithms for hard problems; the research in approximation methods and computational methods, combining heuristic search with lower-bound techniques like cutting planes and the study exponential-time exact solution algorithms, can provide effective means to handle these problems. Finally, the fifth topic comprises the complexity of cutting plane methods; the author says that is fundamental examining bounds on the complexity of the overall method, investigating algorithms for separation problems to deliver cutting planes for particular models, and obtaining insights into the selection of cutting planes to speed the convergence of the process.

3.2 Related Works

In this section, related works that solve RCPSPs through heuristic and exact methods are presented.

3.2.1 Formulations

Various MILP based formulations have been proposed in the literature to model resource constrained project scheduling problems. Pritsker et al. (1969) proposed the first binary programming formulation where variables x_{jt} indicate whether a job j ends at time t ($x_{jt} = 1$) or not ($x_{jt} = 0$). This formulation is known as the discrete-time or time-indexed formulation. The number of binary decision variables in this formulation is related to an upper bound $\bar{t}$ for the number of time periods required to complete the project. Thus the number of variables is $\mathcal{O}(n \times \bar{t})$, where n is the number of jobs. Kolisch and Sprecher (1996) extended this formulation to handle different execution modes, adding one additional index m to the binary variables and incorporating this index in the resource-related constraints. In Kone et al. (2011), a new formulation was proposed based on events called on/off event-based (OOE) with $\mathcal{O}(n^2)$ variables.

Time-indexed formulations have been extensively studied and applied to the RCPSP (see, for example, Baptiste and Demassey (2004); Christofides et al. (1987); de Souza and Wolsey (1997); Demassey et al. (2005); Hardin et al. (2008); Sankaran et al. (1999)). Six time-indexed formulations were studied in Artigues (2017). These formulations are initially categorized into three groups according to the meaning of variables, as follows:

Pulse: pulse discrete time - PDT (the most used) formulation with binary variables $x_{jt}\ \forall j \in \mathcal{J}, \forall t \in \mathcal{T}$, such that $x_{jt} = 1$ if job j starts at time t, otherwise $x_{jt} = 0$;

Step: step discrete time - SDT formulation with binary variables $y_{jt}\ \forall j \in \mathcal{J}, \forall t \in \mathcal{T}$, such that $y_{jt} = 1$ if job j starts at time t or before, otherwise $y_{jt} = 0$;

On/Off: on/off discrete time - OODT (the lesser used) formulation with binary variables $z_{jt}\ \forall j \in \mathcal{J}, \forall t \in \mathcal{T}$ such that $z_{jt} = 1$ if job j is processed at time t, otherwise $z_{jt} = 0$.

For each one of these categories, it is possible to model the precedence constraints weakly or strongly according to the linear programming (LP) relaxation strength.

Aggregated: this is a weak way to model precedence constraints, using variable coefficients greater than or equal to 1; aggregated constraints generate $\mathcal{O}(n^2)$ inequalities, each one with $\mathcal{O}(\bar{t} \times m)$ variables, where m is the maximum number of modes;

Disaggregated: this is a stronger way to model precedence constraints, using variable coefficients equal to $\{-1, 1\}$; disagregated constraints generate $\mathcal{O}(n^2 \times \bar{t})$ inequalities, each one with $\mathcal{O}(\bar{t} \times m)$ variables.

Artigues (2017) concluded that PDDT (pulse disaggregated discrete-time), SDDT (step disaggregated discrete-time) and OODDT (on/off disaggregated discrete-time), which are formulations with disaggregated constraints, are all equivalent in terms of the strength of their LP-relaxations and belong to the family of strong time-indexed formulations. He also concluded that the formulations with their aggregated counterparts, PDT, SDT, and OODT, belong to a family of weak formulations and are also all equivalent in terms of their LP relaxation.

Even though the aggregated constraints are weaker, they are much less dense than the disaggregated. For this reason, papers from the literature (see, for example, Christofides et al. (1987); Zhu et al. (2006)), begin the formulation with the aggregated constraints and add the disaggregated ones as cutting planes. As in previous works, our algorithm starts with the weak time-indexed formulation pulse discrete-time (PDT) based on formulations proposed in (Kolisch and Sprecher, 1996; Pritsker et al., 1969) for the MMR-CMPSP version. In the next paragraphs, we review some computational approaches to handle MILP formulations for the RCPSPs.

Most of the exact algorithms for the RCPSPs (Brucker et al., 1998; Chakrabortty et al., 2015; Christofides et al., 1987; Demeulemeester and Herroelen, 1992) are built upon LP-based Branch and Bound (B&B) (Christofides et al., 1987; Land and Doig, 2010) algorithms. A key component in the design of these algorithms is which formulation is employed: compact formulations, with a polynomial number of variables and constraints, are usually able to quickly provide valid lower bounds since their LP relaxations are easily solved. The LP relaxation bounds are relatively weak but can be significantly improved by adding cutting planes in a Branch-and-Cut (B&C) algorithm.

Cutting planes and strengthening constraints in MILP models are explored for RCPSP and to other variants of scheduling problems. Applegate and Cook (1991) proposed clique cuts, half cuts, and other cuts specific to job-shop scheduling problems (JSSP). Hardin et al. (2008) proposed a lifting procedure to cover-clique inequalities to a resource-constrained scheduling problem with uniform resources (URCSP) requirement. Cavalcante et al. (2001) applied cover cuts to the labor constrained scheduling problem (LCSP), based on practical requirements arising in industry.

3.2.2 Solution Methods for the SMRCPSP

Sankaran et al. (1999) proposed a cutting-plane algorithm for the SMRCPSP with minimal cover inequalities and clique inequalities to test problems provided by Patterson and Huber (1974). Also, they introduced three preprocessing techniques: reduction of lower and upper bounds, the identification of redundant constraints between resource and precedence constraints, and the coefficient strengthening in constraints Johnson et al. (1985).

Christofides et al. (1987) proposed a B&B algorithm that uses disjunction arcs to handle resource conflicts. Four lower bounds are examined: the first one is based on the longest path in the precedence graph; the second one is based on an LP relaxation strengthened with cuts; the third one is based on Lagrangian relaxation and the fourth one is based on disjunctive arcs. Their LP-based method incorporated the dynamic inclusion of disaggregated precedence constraints and resource-based conflict constraints for pairs of jobs. The first lower bound, based on precedence constraints, can be computed quickly and was used in the B&B algorithm with other bounds. The second lower bound was promising but was not used within the B&B method. The Lagrangian relaxation technique was found to provide less competitive results. They provide additional inequalities for the time-indexed version. Finally, the fourth lower bound performed

quite well, especially for problems with tight resource constraints. For the SMRCPSP, Christofides et al. (1987) were able to prove the optimality of instances involving up to 25 jobs and 3 resources.

Rahmania et al. (2015) present a study on the performance of the differential search algorithm for SMRCPSP; this algorithm is an evolutionary algorithm inspired by the migration of superorganisms using the Brownian-like random-walk motion concept. The proposed method provides an initial set of scheduling and attempts to improve using the migration behavior of the superorganisms. To create a new population, they use the serial and/or parallel generation scheme. The authors define a rate called "success", in which it shows the number of instances in which they were solved by the proposed method and also the average deviation of the optimal solution was analyzed. The set of instances used was PSPLIB. The method was efficient for the benchmark with instances of 30, 60 and 120 jobs comparing it to the state-of-the-art algorithms for these problems, including the work of Koulinas et al. (2014), in this case for the benchmark with 60 it got second place. The authors concluded that for the 60 and 120 jobs, due to the higher number of jobs, the method had greater difficulty in solving the problems, so the success rates were lower when compared to the benchmark of 30 jobs.

Koulinas et al. (2014) proposed the Particle Swarm Optimization - PSO algorithm based on the hyper-heuristic algorithm that works as the high-level algorithm that controls the other heuristics at a lower level operating on a solution space. The authors construct scheduling from the serial generation scheme. The instance set used on the experiments are from the PSPLIB, the algorithm proved to be better compared with others from the state-of-the-art, until then, for all benchmarks.

Brucker et al. (1998) presented a Branch and Bound algorithm for SMRCPSP, based on a schematic representation of schedules. The branching scheme starts from a graph representing a set of conjunctions of constraints between predecessors and a set of resource constraint disjunctions and represents a set of disjunctions between pairs of jobs or instants in which these jobs may or may not be in parallel. The authors use a notation that represents the relations of conjunctions, disjunctions, flexibilities and parallels in a feasible set of schedules, where each schedule represents a node in the numbering tree. The priority, in the immediate selection, is to eliminate the flexible relationships by moving during branching for parallel relations or disjunctions. To correct the disjunctions, one tries to move the relations to conjunctions. The authors introduced a concept to allow to move the flexibilities without the branching. Two ideas for immediate selection were considered: feasibility and dominance criteria. The first is that the movements

can lead to infeasibility, so it is necessary to apply a new movement, the second is that modifications can generate solutions that are dominated by the best current solution, so a complementary movement is necessary if the current can be improved. To compute the upper bound was used the Tabu Search. To calculate the lower bound, the procedure LB_2 based on a linear program that relaxes the precedence and allows preemption was used to compute the best result. They presented computational results for instances of 30 and 60 jobs generated with complexity factor ≥ 1, which represents the average of immediate successors, resource factor and capacity $[0, 1]$, which treat the average number of resources limited and the capacity in relation to the demand for resources.

Möhring et al. (2003) propose a method in which a relaxed problem is solved by calculating the minimum cut in a directed graph. The authors based on the idea that Lagrangian relaxation of an integer-indexed programming formulation and schedule-based relaxation, as well as important approximation algorithms, are efficient methods used in scheduling. Basically, the authors propose the Lagrangian relaxation for the resource constraints to calculate the lower limits. The focus of this approach is a straightforward transformation of the problem into a minimal cut problem in an appropriately defined directed graph. To solve the problem of the minimum cut, authors use a maximum flow code. The authors also used a set of instances generated by ProGen, with instances of size 30, 60 and 90. According to the authors, the limits obtained can be calculated quickly and are particularly strong for scenarios with scarce resources.

Kochetov and Stolyar (2003) developed an evolutionary algorithm that combines the genetic algorithm, path-relinking and tabu search. Solutions are evolved by choosing two solutions and building a path through the connections between the selected ones. The best solution of the way is chosen and improved by tabu search. The best solution from the tabu search is added to the population and the worst solution is removed.

Alcaraz et al. (2004) developed a genetic algorithm based on the job list representation. A gene indicates the SGS (Schedule Generation Scheme) used; SGS defines how to generate a job schedule. An additional gene indicates the methods between forward and backward (Toffolo et al., 2016) to be used to time-shift jobs in some list. The crossover operator for job lists extends so that it can be constructed bidirectionally, by scheduling jobs backward and forwards.

3.2.3 Solution Methods for the MMRCPSP

Among the approaches presented in the literature, we choose the following papers, since they are a well-known work and consider instances from the same benchmark datasets.

Zhu et al. (2006) presented a B&C algorithm for the MMRCPSP, including cuts derived from resource conflicts, where all resource constraints are in the form of generalized upper bound (GUB) constraints. Besides, disaggregated cuts from the precedence relationship for pairs of jobs (j, s) in the precedence graph are included. An adaptive branching scheme, to speed up the solution process, is developed along with a bound adjustment scheme that is always executed iteratively after branching. To optimize the solutions found in the first stage, the authors use a high-level neighborhood search strategy called Local Branching (Fischetti and Lodi, 2007). For the MMRCPSP, the authors were able to prove the optimality of 554 instances with 20-jobs and 506 instances with 30-jobs.

Different techniques such as CP and SAT have also been used to solve the MMRCPSP. In this context, Demassey et al. (2005) use CP techniques to provide valid inequalities to strengthen LP relaxations. A recent work using cutting planes as valid clauses for SAT is presented in Schnell and Hartl (2017). Schnell and Hartl (2017) propose three formulations based on Constraint Programming to solve the MMRCPSP, using the G12 CP platform and the Solving Constraint Integer Programs (SCIP) as an optimization framework, both making use of solution techniques combining CP and SAT. They further combine MILP by inserting a new global constraint on the domain of renewable resources for SCIP. The authors achieved the same results with better computational times than Zhu et al. (2006). They also were able to prove the optimality of 1428 instances with 50-jobs and 100-jobs and improve various lower and upper bounds.

3.2.4 Solution Methods for the MMRCMPSP

Among the heuristic approaches presented in the MISTA Challenge, the finalist proposals are discussed in this section.

Asta et al. (2014), winners of the competition, proposed a approach combining the Monte-Carlo methods i.e., statistical method, in which a sequence of random numbers is used to perform a simulation (Metropolis, 1949) and Hyper-Heuristics i.e., heuristic that can be used to deal with any optimization problem, provided some parameters, such as

structures and abstractions, are given, related to the problem considered (Burke et al., 2003).

In this approach is used a sequence representation defined through the job allocations in a schedule. So it is possible to build another schedule by allocating jobs one by one as early as possible. This representation is advantageous because it is easier to obtain viable schedules. Two steps are required to produce feasible, high-quality solutions.

In the first stage of construction, through inspections on good solutions, Asta et al. (2014) defined that a proper constructor must create initial sequences that copy partial programming ordering (start, middle and end). Therefore, a fast method that generates random schedules was used, where the order of jobs within each partial programming is chosen by chance, respecting the precedence. These schedules are used as sampling in decision making.

In the second stage of improvement, movements, also known as neighborhoods, are controlled by a combination of metaheuristic and hypertext-heuristic, with a population-based and memetic algorithm (incorporating cultural and learning concepts, allowing faster precision and convergence (Ong et al., 2010)). All movements generate sequences that respect precedence relations.

Basically, this approach consists of constructing a method that works the sequence in which the jobs are passed to a programming constructor. Building a sequence of the first job is done by a hybrid that generates random schedules and project partitioning. The improvement phase uses a large number of motions through neighbors controlled by metaheuristics, memetic and hyper-heuristic algorithms in a multi-thread context.

Geiger (2013), second winner in the competition, proposed a Variable Neighborhood Search (VNS) (local search method that explores the solution space through systematic neighborhood structure exchanges (Mladenović and Hansen, 1997)). In this approach he present a set of viable schedules X associated to two vectors, $M = (m_1, ..., m_n)$ and $S = (S_1, ..., S_n)$, where M represents the chosen execution mode m_j for each job j and S is the permutation of the job indexes.

The approach generates feasible initial schedules, randomly assigns modes to the $\mathcal{M}$ vector. If $\mathcal{M}$ is unviable concerning non-renewable resources, a procedure is applied that randomly changes the values until viability is guaranteed and an allowed iteration number is reached. If feasibility is not obtained $\mathcal{M}$ is rebuilt again. Subsequently, the sequence S of the programming is constructed through a scheme of generation in serial.

The priority is the job that has the shortest start time.

The local search is applied to the X program constructed, based on the VNS and Iterated Local Search (ILS), where, through initial programming, neighbors are generated through several moves and tested for acceptance. If programming that can no longer be improved is reached, a perturbation in the current solution is made and the search continues. Four neighborhood moves were proposed.

Local search is performed in parallel. Once the optimal location is reached, the best alternative found until then is updated and the search continues with this best-known alternative. Through parallelism, all the nuclei are concentrated in a single solution, dividing only in the investigation of the neighborhoods.

In work proposed by Toffolo et al. (2013) the MMRCMPSP was approached from the use of integer programming and a hybrid heuristic. Initially, an exact mode selection model is created, which evaluates the use of non-renewable resources and enables valid allocations for jobs, disregarding renewable resources.

The following are the decision variables of the model:

$\boldsymbol{x_{jm}}$: binary variable that assumes value 1 if job js allocated in mode m and 0 otherwise;

$\boldsymbol{t_j}$: integer variable that indicates the start time of job j;

$\boldsymbol{z_p}$: integer variable that indicates the end time of the project p;

Minimize:

$$\omega_1 \sum_{p \in P} z_p \; + \; \omega_2 \sum_{c \in C} \sum_{j \in J_c} \sum_{m \in M_j} d_{jm} \cdot x_{jm} \; + \; \omega_3 \sum_{j \in J} \sum_{m \in M_j} d_{jm} \cdot x_{jm} \tag{3.1}$$

Subject to:

$$\sum_{m \in M_j} x_{jm} = 1 \quad \forall j \in J \tag{3.2}$$

$$\sum_{j \in J} \sum_{m \in M_j} q_{kjm} \cdot x_{jm} \leq q_k^{disp} \quad \forall k \in K \tag{3.3}$$

$$t_j + \sum_{m \in M_j} d_{jm} \cdot x_{jm} \leq t_l \quad \forall (j,l) \in Pred \tag{3.4}$$

$$z_p \geq t_j \quad \forall p \in P,\ \forall j \in J_p \tag{3.5}$$

The objective function (3.1) is composed of three parts. For each parcel, a weight was assigned, indicating the priority of the parcel. The weights are defined in a hierarchical fashion, such that $\omega_1 \gg \omega_2 \gg \omega_3$.

The first part is responsible for minimizing the end time of each project p. The second and third parts are responsible for minimizing the duration of jobs. The second installment differs from the third one by prioritizing jobs that appear more often in the C path set.

The equation (3.2) is responsible for ensuring that each job is allocated only once. The equation (3.3) is included to ensure the available amount of non-renewable resources. Finally, the equation (3.5) is responsible for ensuring that the variable z_p is equal to the allocation time of the artificial end-project job p. Artificial jobs have a duration of 0.

Once the modes in which jobs can be executed are defined, a constructive heuristic based on an exact model was used to generate initial solutions. The authors made use of a local search hybrid heuristic that combines integer programming with the Forward-Backward Improvement (FBI) method by applying mode and job perturbations in time windows through entire programming and non-ascending random search. With the results, the authors achieved third place in the MISTA Challenge.

Chapter 4

Heuristic Strategies

Stochastic Local Search (SLS) methods obtained the best results for many optimization problems. In school timetabling, da Fonseca et al. (2014) won the Third International Timetabling competition with a hybrid Simulated Annealing (SA) algorithm incorporating eight neighborhoods. In project scheduling, Asta et al. (2014) won the MISTA Challenge (Wauters et al., 2016) with a Monte Carlo based search method with nine neighborhoods. In both methods, neighborhoods are explored stochastically instead of the much popular deterministic best/first fit alternatives where local optimality is usually reached at every iteration. In these SLS algorithms, a random neighbor is generated in one of the neighborhoods and its acceptance is immediately decided. In this section, we focus on SLS algorithms with these characteristics, instead of considering the broader definition (Hoos and Stützle, 2005) of SLS.

The reasons for the good performance of these methods are the subject of study. Some insights may come from theoretical models such as in Ochoa et al. (2014) or experimental studies such as in Hansen and Mladenović (2006). As both studies point out, more connected search spaces provide a landscape where it is easier to escape from attractive basins in the search landscape. This is a very favorable point of SLS since the different neighborhoods and the flexibility used in their exploration contribute to increased connectivity of the search space graph.

In the remainder of this chapter, we explain the proposed approach to generate tight upper bounds. Initially, a *matheuristic*, i.e., a method that uses IP to generate improved solutions with high time constraints combined with heuristics (Maniezzo et al., 2010), which produce improved upper bounds are explained. We then focus on the

representation and decoding of the solution.

4.1 Solution Representation and Decoding

A solution is represented by an ordered pair $(\pi, \mathcal{M})$, where π indicates an allocation sequence and $\mathcal{M}$ is a feasible set of modes, i.e. an allocation of modes which respects the availability of non-renewable resources. A complete solution is decoded with a Serial SGS algorithm (Demeulemeester and Herroelen, 2002) from $(\pi, \mathcal{M})$, allocating each job in the first timeslot where precedence constraints are respected and sufficient renewable resources are available.

The first step to constructing the initial solution is to build a mode set that satisfies the consumption of non-renewable resources. The binary program to build this initial set of modes considers the set $\mathcal{J}$ of jobs with its processing times d_{jm} and the set $\mathcal{K}$ of non-renewable resources. Each job j has a set $\mathcal{M}_j$ of possible modes. Also, the consumption of the non-renewable resource k by job j in mode m is denoted as q_{kjm}. Thus, the following binary program is solved to select the mode m to execute job j, considering its respective decision variables x_{jm} and resource availability $\breve{q}_k$ for each non-renewable resource:

Minimize:

$$\sum_{j \in \mathcal{J}} d_{jm}.x_{jm} \tag{4.1}$$

Subject to:

$$\sum_{j \in J} \sum_{m \in \mathcal{M}_j} q_{kjm} x_{jm} \leq \breve{q}_k \;\; \forall k \in \mathcal{K} \tag{4.2}$$

$$\sum_{m \in \mathcal{M}_j} x_{jm} = 1 \;\; \forall j \in \mathcal{J} \tag{4.3}$$

$$x_{jm} \in \{0, 1\} \;\; \forall j \in \mathcal{J}, m \in \mathcal{M}_j \tag{4.4}$$

This binary program corresponds to the $\mathcal{NP}$-hard problem of the 0-1 multidimensional knapsack problem. Fortunately, modern integer programming solvers (Johnson et al., 2000a; Jünger, Liebling, Naddef, Nemhauser, Pulleyblank, Reinelt, Rinaldi and

Wolsey, 2010) consistently solve this binary program to all instances of MISTA to optimality, always in a fraction of a second.

The feasible mode set is used to build a complete solution by using a greedy randomized constructive (GRC) algorithm do decide the processing order of jobs. The GRC iteratively selects the next job to be allocated by randomly selecting among all available jobs. The probability of selecting jobs with shorter earliest starting times $\breve{e}_j^s$ is exponentially larger, as in Bresina (1996).

As most of the processing time of this algorithm is spent in the local search phase, the ability to quickly decode a $(\pi, \mathcal{M})$ pair is a fundamental aspect in the performance of the algorithm. We implemented all optimizations proposed in Asta et al. (2014), such as prefix detection and early exploration of resource insufficiency. Differently from Asta et al. (2014), we do not guarantee a valid topological sort in π. This speeds up the generation of valid moves, since some validations may be disabled, but the cost saved is moved to the decoding phase. To transform the sorting in π into a valid topological sorting, at each new allocation, one has to check the available job with the highest priority (smallest desired position), which yields an $O(n^2)$ algorithm to decode π. Fortunately, we devised a simple heap-based algorithm (Algorithm 4.1) to speed up this to $O(n \log n)$.

Algorithm 4.1: `convert_vector_valid_topological_sort`

Data: $\mathcal{J}$ jobs, $\mathcal{S}_j^+$ successors of each job j, $\mathcal{S}_j^-$ predecessors of each job j and desired positions π
Result: valid topological sorting π^* based on π

```
1  H ← create_heap();
2  for j ∈ J do
3      add_to_heap( H, j, |S⁻j| × |J| + π_j );
4  p ← 0;
5  while H not empty do
6      j ← remove_first( H );
7      for j' ∈ S_j^+ do
8          decrease_value( H, j', |J| );
9      π*_p ← j;
10     p ← p + 1;
11 return π*;
```

Initially, all jobs are inserted into the heap with priority $|\mathcal{S}_j^-| \times |\mathcal{J}| + \pi_j$ (line 3). The operation `remove_first` returns and removes from the heap the element with a smallest

priority value.

4.2 Neighborhoods

Neighborhoods operate either on π or $\mathcal{M}$. Moves that violate precedence relationship and non-renewable resource consumption constraints are discarded. Fourteen neighborhoods are considered: Most neighborhoods were proposed by Asta et al. (2014). Some of these neighborhoods aim at the minimization of the TPD.

4.2.1 Neighborhoods Operating on π

An illustrative example will be used to represent neighborhood moves operating on π considering the following data: a set of projects $\mathcal{P}$, in which $|\mathcal{P}| = 5$. Each project $p \in \mathcal{P}$ contains the following jobs: $p_1 = \{8, 12\}$, $p_2 = \{3, 13, 1\}$, $p_3 = \{6, 7\}$, $p_4 = \{2, 5, 14, 10, 9\}$ and $p_5 = \{25, 26\}$.

Squeeze Project on Extreme - SPE

All jobs of a project p are compacted around a reference job, while jobs of other projects are moved either before or after project p. The project to be compacted, is represented by project p_j in which job j allocated at position i belongs. In the first instance, all positions are accessed to identify the indexes of the first and last job of p_j. The indexes of these jobs are stored, respectively, in $\underline{i}$ and $\overline{i}$. After setting indexes, it's possible compress all jobs in p_j around the reference job in position i. The procedure to compact the project it is simple, just shifting to the right all jobs from p_j where $\underline{i} < i < \overline{i}$ and to the left where $i < i < \overline{i}$.

Figure 4.1 shows a sample neighbor s' of s in the `SPE` neighborhood, with position 4 of π being used as a reference job, thus all jobs pertaining to the same project of job 14 are squeezed immediately at its side these jobs are colored in gray.

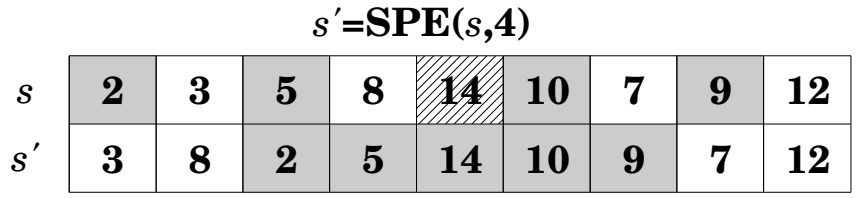

Figure 4.1: Example of a neighbor s' generated in the `SPE` neighborhood of s

Swap and Compact two Projects - SCP

Relative positions of projects are swapped and all jobs of both projects are sequentially allocated, starting with jobs from the project which was previously starting at a later timeslot. The neighborhood purpose is to relocate a sequence based on jobs belonging to two projects p_1 and p_2, furthermore jobs of p_1 and p_2 will be compacted. The swap is done by first allocating all jobs of the project that finishes last. It is necessary to receive as a parameter the projects p_1 and p_2.

At the first step, all positions are accessed in reverse order to identify between p_1 and p_2, which finishing last and what are the early indexes allocations of each project. During this process, jobs of each projects p_1 and p_2 are stored, respectively, in vectors $M1$ and $M2$. The second step, comprises the swap of sequence, for this it is needed to set a index $\widehat{i}$, representing the first job index between the projects p_1 and p_2. So, all jobs not belonging to projects p_1 and p_2 and with smaller indexes than $\widehat{i}$ are stored in vector L and those with bigger indexes than $\widehat{i}$ are stored in vector R.

After completing the second step, it is necessary unite all jobs contained in $M1, M2, L$ and R. Initially the sequence is rebuilt including in the first place all jobs of L, representing jobs allocated before $\widehat{i}$ and not belonging to projects p_1 and p_2. Soon after, are united the project jobs that finishing by last (M_1 or M_2), then jobs project that finishing first (M_1 or M_2). Finally, all jobs of R are added, representing jobs not belonging to projects p_1 and p_2 allocated after $\widehat{i}$.

Figure 4.2 shows a sample neighbor s' of s in the `SCP` neighborhood; the method takes as parameter two projects $(2, 3)$; all jobs belonging to projects p_2 and p_3 are colored in different grayscale intensities, 3 and 2 respectively, those jobs are summarized and swapped, respecting the relative positions.

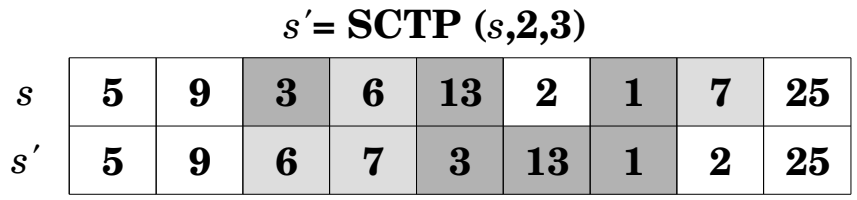

Figure 4.2: Example of a neighbor s' generated in the SCP neighborhood of s

Offset Project - OP

All jobs of a given project are shifted by the same number of positions. It receives as a parameter the project p and the size of offset k that can be positive or negative to represents the direction, where $k < 0$ represents the offset to the left and any other number represents the offset to the right.

Before performing the offset is important to establish the limits, identifying the index pid of the last or first job of p and check following the direction if it is possible to apply the offset k. The offset to the left is invalid when $pid - k < 0$, the offset to the right becomes invalid when $pid + k \geq |s|$. In both cases of infeasible move, the size of k is decremented until it finds a possible size of k, when k is 0. Only after checking the feasibility of offset all jobs of p are displaced.

Figure 4.3 shows a sample neighbor s' of s in the OP neighborhood; project p_4 and the right offset size $k = 1$ are received as parameter, all jobs belonging to p_4 are grayscale.

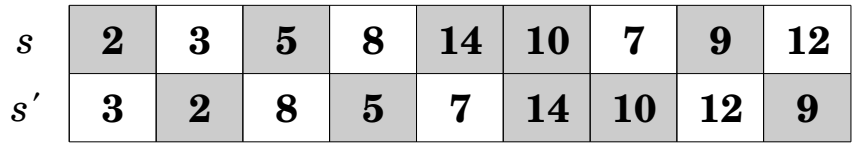

Figure 4.3: Example of a neighbor s' generated in the OP neighborhood of s

Compact Project on Percentage - CPP

A percentage of jobs of a project p are justified to the left, starting from the end of π. It receives as a parameter the project p and the compaction percentage $perc$. At first step, it is calculated the number of jobs α in p that will be compacted relative to

perc, represented by $\alpha = |\ (|N| \cdot\ perc)\ |$, just after are identified the indexes of the last $\bar{i}$ and first $\underline{i}$ jobs of p allocated, taking into account the number of jobs that will be compacted starting at the end. Jobs belonging to p allocated at this interval are stored in the vector $M1$. All jobs with smaller indexes than $\underline{i}$ are added to vector L and all jobs with bigger indexes than $\bar{i}$ are added to vector R. The other jobs within the range $\underline{i}$ and $\bar{i}$ not belonging to project p are added to vector $M2$.

After completing this first step, it is necessary to unite all jobs contained in $M1, M2, L$ and R. Initially, the sequence is rebuilt, including all jobs of L, representing jobs allocated before $\underline{i}$. Soon after, are added all jobs of $M1$, containing jobs compacted of p and then are added jobs of $M2$, containing jobs that prevented the compression of project jobs p. Finally, jobs of R are added, containing all jobs allocated after $\bar{i}$.

Figure 4.4 shows a sample neighbor s' of s in the `CPP` neighborhood considering project p_4 (jobs shown in shaded cells) and compaction percentage 0.5; note that the project has 5 jobs, so $\lfloor 5 \times 0.5 \rfloor = 2$ and thus the last two jobs are contiguously allocated in π.

s'=CPP(s,4,0.5)

s	**2**	**3**	**5**	**8**	**14**	**10**	**7**	**9**	**12**
s'	**2**	**3**	**5**	**8**	**14**	**10**	**9**	**7**	**12**

Figure 4.4: Example of a neighbor s' generated in the `CPP` neighborhood of s

Invert Sequence of Jobs - ISJ

A subsequence of jobs in π is inverted. It receives as a parameter an index i indicating the beginning of the subsequence L and a parameter k indicating the size of the subsequence L. After setting the sub-sequence L, all jobs are reversed on the sequence. Due to the fact it is possible to generate invalid sequences to reverse a sequence of jobs that have precedence relationships with each other, it is necessary to check the feasibility of the move.

Figure 4.5 shows a sample neighbor s' of s in the `ISJ` neighborhood; parameters (1,4) represent the position and size to invert the substring shaded; jobs are colored in grayscales.

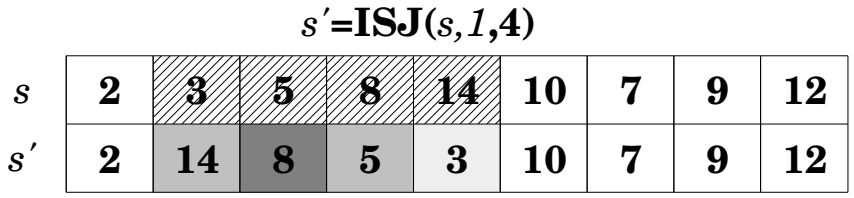

Figure 4.5: Example of a neighbor s' generated in the `ISJ` neighborhood of s

Offset Job - OJ

Shifts the position of one job in the sequence π. It receives as a parameter the index i of job j to be displaced, the size k of offset and the direction dir, where 1 represents the offset to the right and any other number represents the offset to the left. Before performing the offset is important to establish the limits for moving j without violating precedence constraints. The offset to the left is invalid when $i - k < 0$, the offset to the right becomes invalid when $i + k \geq |s|$. In both cases of infeasible move, the size of k is decremented until it finds a possible size for k or $k = 0$.

Figure 4.6 shows a sample neighbor s' of s in the `OJ` neighborhood; in addition to the sequence, the method receives the position regarding the job and the size of the displacement, the position to be shifted is 3 for job 8 and will be pushed right to 3 positions.

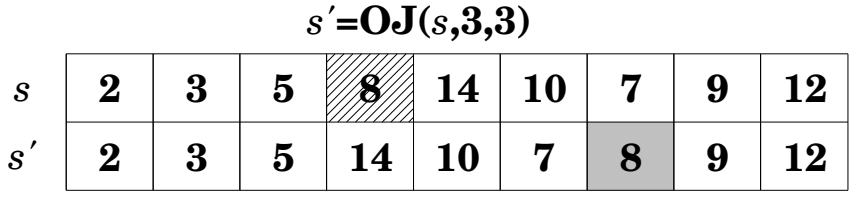

Figure 4.6: Example of a neighbor s' generated in the `OJ` neighborhood of s

Swap Two Jobs - STJ

Swaps the position of two jobs in the sequence. It receives as a parameter the jobs j_1 and j_2 to be swapped. Only after checking the feasibility of offset the swap will occur.

Figure 4.7 shows a sample neighbor s' of s in the `OJ` neighborhood and takes as its parameter the positions 2 and 4, referring to jobs 5 and 14 to be swapped.

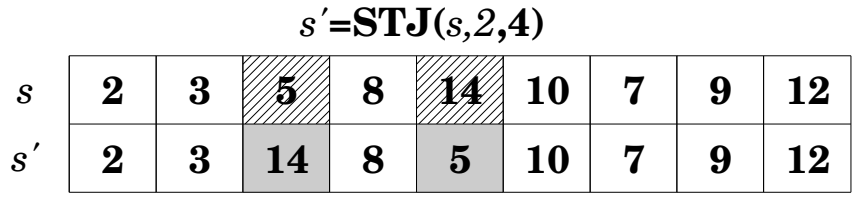

Figure 4.7: Example of a neighbor s' generated in the `STJ` neighborhood of s

Compact Subsequent Projects - CSP

Compress a contiguous list of projects in a sequence. It receives as a parameter a number of projects to be compressed on direction *dir* and the index of the first project. Firstly, the subsequent projects are identified, when $dir = 0$ all subsequent projects are compacted starting the shift from right to left, otherwise the projects are compacted starting the shift from left to right. All jobs outside the subsequent projects that are in the range will be moved to after the last project adjacent allocated.

Figure 4.8 shows a sample neighbor s' of s in the `OJ` neighborhood; the method takes as a parameter the first project p_1 and the subsequent project number 1, both projects will be compressed and sequenced without interfering with other project jobs, jobs of each project are in different grayscales.

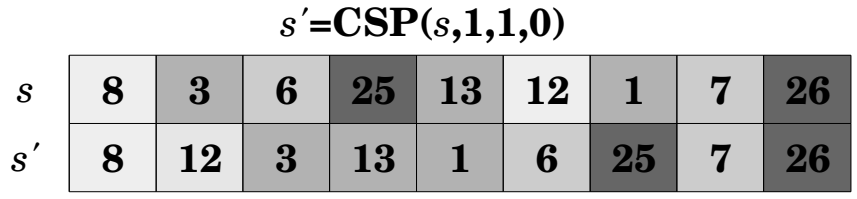

Figure 4.8: Example of a neighbor s' generated in the `CSP` neighborhood of s

Successive Swap of a Job in a Window - SSJW

All possible swap positions for a job in a window are explored in a first-fit fashion; The neighborhood purpose is to find the first swap of position between two jobs that improve the objective function. For this, the method receives a position i which represents the job j. Subsequently, a window W is set between the position $\underline{i}$ of the last predecessor job of j and position $\bar{i}$ the first successor job of j. The size of W was limited to l which

is defined by $(\bar{i} - \underline{i} - 1)/2$. Then, job j on position i is sequentially changed with every job in W, until a first improvement in the objective function occurs. Only after checking the feasibility of the move it will occur.

Figure 4.9 shows a sample neighbor s' of s in the `OJ` neighborhood which takes the $i = 2$ position as a parameter and attempts to switch to the first position of the W window, if no improvement in objective function is reached the method makes a new exchange with the next job of the W window, the improvement in the objective function occurs in the second exchange, project jobs are colored in gray, the first predecessor and last successor of the job are emphasized.

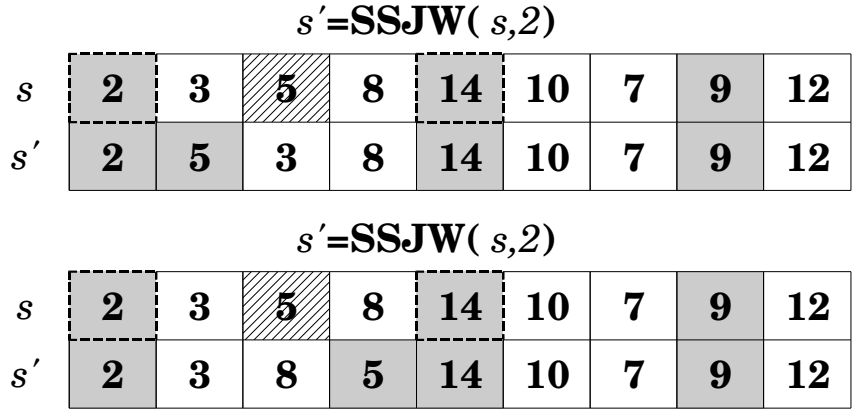

Figure 4.9: Example of a neighbor s' generated in the `SSJW` neighborhood of s

Successive Insertions of a Job in a Window - SIJW

Similar to `SSJW`, but instead of swapping jobs, a job is re-inserted in another position within the explored window until there is an improvement in the objective function.

Figure 4.10 shows a sample neighbor s' of s in the `OJ` neighborhood; the method takes as its parameter the 2 index for jobs 5 and attempts to insert it into each W position, an improvement of the objective function occurred in the third insertion, project jobs are grayed out, boundaries and index are emphasized.

4.2.2 Neighborhoods operating on $\mathcal{M}$

In this subsection, neighborhoods which operate over the mode set are introduced. Since an indirect representation is employed, as described earlier, the satisfaction of renewable

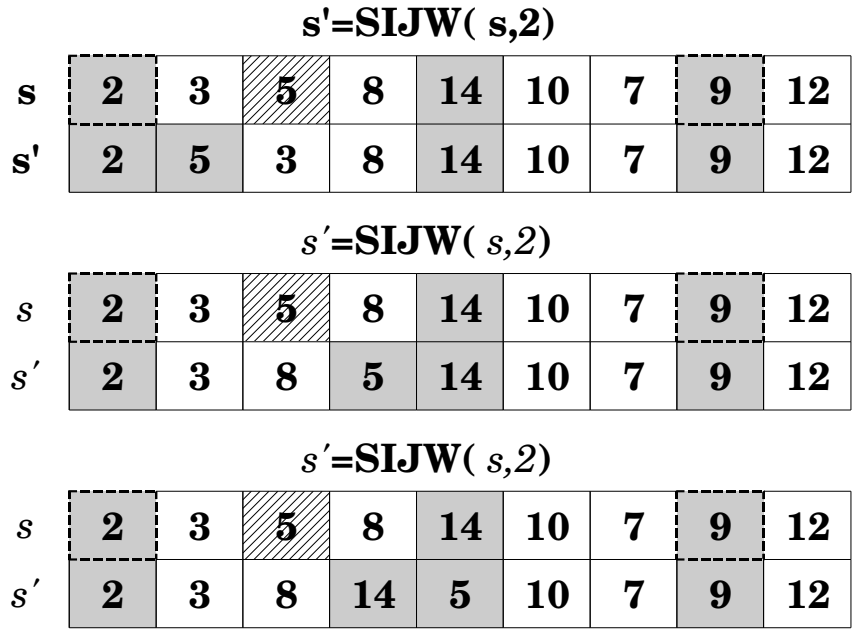

Figure 4.10: Example of a neighbor s' generated in the `SIJW` neighborhood of s

resources is always guaranteed. The same does not holds for non-renewable resources, so that the feasibility of these moves must be checked.

Change One Mode - C1M

Changes the mode of one job. This neighborhood verifies if the changes of modes are valid or not concerning non-renewable resources. Figure 4.11 shows a sample neighbor s' of s in the `C1M` neighborhood; the method receives job 1 and the new mode 0 to be changed;

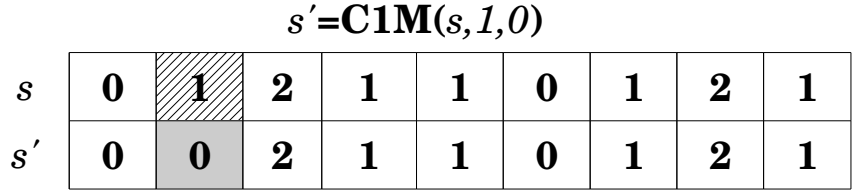

Figure 4.11: Example of a neighbor s' generated in the `C1M` neighborhood of s

Change Two Modes - C2M

Changes the mode of two jobs. This neighborhood is similar to C1M, but its purpose is change two execution modes m_1 and m_2, which corresponds to two jobs j_1 and j_2. Figure 4.12 shows a sample neighbor s' of s in the `C2M` neighborhood where it takes as parameters jobs 1 and 2 and the new modes 0 and 1 for switching;

s'=C2M(*s,1,2*,0,1)

s	**0**	**1**	**2**	**1**	**1**	**0**	**1**	**2**	**1**
s'	**0**	**0**	**1**	**1**	**1**	**0**	**1**	**2**	**1**

Figure 4.12: Example of a neighbor s' generated in the `C2M` neighborhood of s

Change Three Modes - C3M

Changes the mode of three jobs; to reduce the size of this neighborhood only triples of consecutive jobs in the precedence graph are considered; Figure 4.13 shows a sample neighbor s' of s in the `C3M` neighborhood; jobs $1, 2, 4$ were chosen with the respective new modes $0, 1, 0$;

s'=C3M(*s,1*,2,4,0,1,0)

s	**0**	**1**	**2**	**1**	**1**	**0**	**1**	**2**	**1**
s'	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**2**	**1**

Figure 4.13: Example of a neighbor s' generated in the `C3M` neighborhood of s

Change Four Modes - C4M

Changes the mode of four jobs that are consecutive in the precedence graph (similarly to `C3M`); Figure 4.14 shows a sample neighbor s' of s in the `C4M` neighborhood; the method takes as its parameters jobs $1, 2, 4$ and 5 and the respective new modes $0, 1, 0$ and 1.

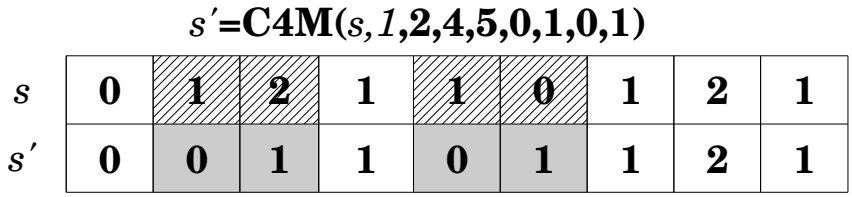

Figure 4.14: Example of a neighbor s' generated in the `C4M` neighborhood of s

4.3 Neighborhood Composition

Given the set of neighborhoods $\{\mathcal{N}_1, \ldots, \mathcal{N}_k\}$ and a search method, we define probabilities $\mathbf{p} = (p_1, \ldots, p_k)$ of selecting each neighborhood at each iteration as *neighborhood composition*. In this work, the search method used is the Late Acceptance Hill Climbing (LAHC) (Burke and Bykov, 2017), i.e., an SLS based on late acceptance. As k increases, it gets harder to find the best configuration of $\mathbf{p}$.

Some important considerations are: should $\mathbf{p}$ remain fixed during the entire search or should it be dynamically updated as the search progresses? Is it possible to learn the best configurations during runtime without sacrificing too much the computational efficiency of the search method? In the next subsections, we try to answer these questions.

4.3.1 Offline neighborhood composition

In this section, we evaluate a metric to define $\mathbf{p}$ *a priori*, i.e., based on a statistical analysis of the neighborhood's efficiency. The performance of all neighborhoods in a previously generated pool of solutions is considered. We call a neighborhood *efficient* when its stochastic exploration produces good results, i.e., if it improves the solution cost or generates sideways moves[1] with minimal computational effort.

In a preliminary set of experiments, we observed that the efficiency of different neighborhoods varied considerably depending on the *stage* of the search. Neighborhoods, which were quite efficient at the beginning of the search, did not present the same functional properties as the search advanced.

We evaluate the efficiency of neighborhoods in two different stages. In the first stage, low quality (initial) solutions are considered, generated by quick constructive

[1]Moves which do not change the objective function value but modify the solution.

algorithms. In the second stage, incumbent solutions obtained after 10,000 iterations without improvement of a state-of-the-art Late LAHC (Soares et al., 2015) algorithm are selected.

Table 4.1 describes the minimum, maximum and average solution quality[2] for solutions of both phases. Higher values of this metric indicate better quality.

Table 4.1: Characteristics of the solution pool used for *offline* neighborhood analysis

	Solution quality		
Iter.	Minimum	Average	Maximum
0	<0.0057	0.3566	0.6666
10000	0.6175	0.8193	1.000

Each value in Table 4.1 considers all instances from the most generalized variant MMRCMPSP and 10 solutions per instance (300 solutions in total), taking into account the best known solutions in the literature to compute the quality.

The efficiency of each neighborhood k was computed considering a random sampling of neighbors (10,000) for each solution in the pool. It was observed that only analyzing the improvement of neighbors was not enough, so we consider the number of neighbors corresponding to improved solutions $\tilde{s}(k)$, the number of neighbors corresponding to sideways moves $\underline{s}(k)$ with factor $\vartheta \in [0, 1]$, and the total CPU time $\tilde{c}(k)$ spent validating and evaluating neighbors. Thus, the efficiency $\tilde{e}_k$ of a neighborhood k is given by:

$$\tilde{e}_k = \begin{cases} \dfrac{\tilde{s}(k) + \vartheta\ \underline{s}(k)}{\tilde{c}(k)} & \text{if } \tilde{c}(k) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

To do the comparison, we normalize the efficiency values. Let $\max(\tilde{e})$ be the maximum $\tilde{e}_k$ for all $k \in N$. Equation (4.6) shows how the normalized efficiency e_k is obtained for a neighborhood $k \in N$:

[2] the quality $q_i = \frac{b_i}{c_i}$ of a solution for instance i with cost c_i, considering the best known solution's cost b_i

$$e_k = \frac{\tilde{e}_k}{\max(\tilde{e})} \tag{4.6}$$

Table 4.2 presents an efficiency comparison between all neighborhoods in the two different stages. To illustrate the impact of the sideways moves on the final solution quality, the first columns are computed with $\vartheta = 0$ and the last ones are computed with $\vartheta = 0.1$. Note that neighborhoods that operate on entire projects, like `OP`, `CPP`, `SCTP`, are usually well ranked in the first stage. Their efficiency dramatically decreases in the second stage, except for the `CSP`, which has very low efficiency in the first stage, increasing in the second one due mainly to sideways moves. Neighborhoods that change one mode, swap, inverts or shifts jobs are the most significant for the second stage, corresponding to a stage of small adjustments in π and $\mathcal{M}$.

Table 4.2: Normalized neighborhoods efficiency computed *offline*, considering the two stages solution pool

$\vartheta = 0.0$				$\vartheta = 0.1$			
First stage		Second stage		First stage		Second stage	
C1M	1.0000	C1M	1.0000	C1M	1.0000	OJ	1.0000
OP	0.7407	C2M	0.2990	OJ	0.6969	CSP	0.7323
C2M	0.7288	OJ	0.2609	OP	0.6661	STJ	0.6651
OJ	0.5143	STJ	0.2076	C2M	0.6427	C1M	0.5709
CPP	0.4729	ISJ	0.2037	STJ	0.4650	ISJ	0.4345
C3M	0.4716	OP	0.1812	C3M	0.4091	OP	0.3474
STJ	0.3677	C3M	0.0950	CPP	0.4053	CPP	0.1959
SCTP	0.3281	SPE	0.0540	ISJ	0.2824	SCTP	0.1921
C4M	0.3058	SCTP	0.0502	SCTP	0.2753	SPE	0.1789
SPE	0.2725	C4M	0.0167	C4M	0.2578	C2M	0.0865
ISJ	0.2304	CPP	0.0023	CSP	0.2540	C3M	0.0371
SSJW	0.0024	SSJW	0.0019	SPE	0.2327	C4M	0.0029
SIJW	0.0005	SSIW	0.0003	SSJW	0.0016	SSJW	0.0001
CSP	0.0000	CSP	0.0000	SIJW	0.0000	SIJW	0.0000

4.3.2 Online neighborhood composition

In this section we consider the *online* neighborhood composition. In this approach, all neighborhoods start with the same probability of being chosen and this probability is dynamically updated considering results obtained during the search into the LAHC. While this approach introduces a learning overhead, it can more easily adapt itself if the neighborhoods' efficiencies change considerably during the search or vary in different instances.

Whenever a sample of z neighbors is explored, the normalized efficiency e_k (Equations (4.5) and (4.6)) of each neighborhood $k \in \mathcal{N}$ is computed and their selection probabilities are updated according to the results obtained exploring this last batch of neighbors. The definition of z is a crucial point: while larger values of z provide a more accurate evaluation of each neighborhood, smaller values offer a more reactive method. The probability p_k of selecting a neighborhood $k \in \mathcal{N}$ is given by Equation (4.7). Note that a constant β is included when calculating the probability. This constant prevents assigning probability zero to a neighborhood, and thus all neighborhoods have a minimal chance of being selected in any stage of the search. All experiments in this paper employs $\beta = 0.01$.

$$p_k = \frac{e_k + \beta}{\sum_{k' \in N} (e_{k'} + \beta)} \tag{4.7}$$

Figure 4.15 shows how the probabilities evolve over time, considering a small instance, A-10.

The first graph assumes $\vartheta = 0.0$ (sideways moves are not rewarded), while the second considers $\vartheta = 0.1$ (sideways moves are partially rewarded). Note that, as one would expect, most improving moves for small instances are executed at the beginning of the search. Therefore, once a local optimum is reached, the probabilities remain unchanged when sideways moves are not considered. When sideways moves are considered; however, the probabilities vary during the entire search. It is noteworthy that more complex and expensive neighborhoods such as `CSP`, `SSJW` and `SIJW` were given lower probabilities when $\vartheta = 0.1$.

Figure 4.16 presents similar graphs to those of Figure 4.15, but considering a larger

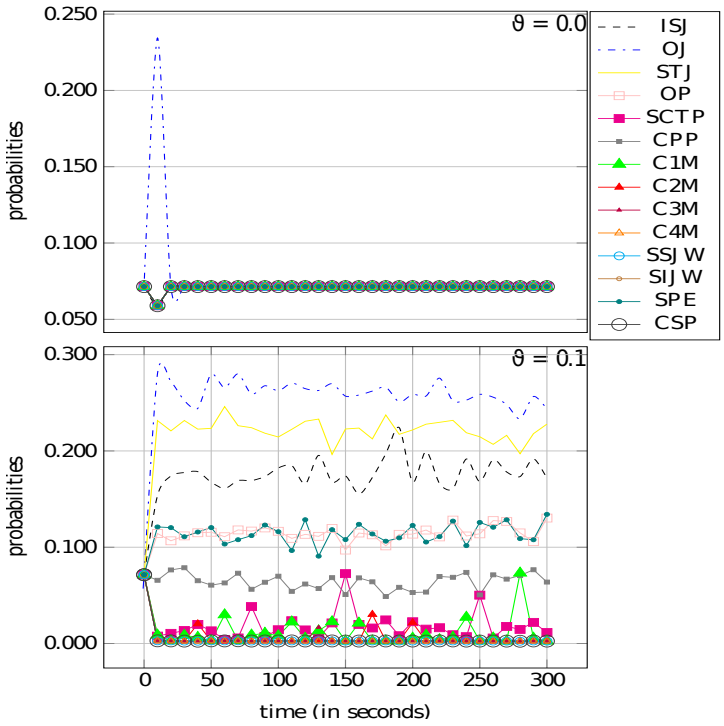

Figure 4.15: Evolution of the neighborhood selection probabilities over time considering instances A-10 and *online* tuning ($z = 1,000$ and $\beta = 0.01$)

instance, B-9.

For this instance, neighborhoods `C1M` and `C2M`, which change job execution modes, were highly rewarded when $\vartheta = 0.0$. This emphasizes the importance of the modes selection for this particular instance. Additionally, the simple neighborhoods `OJ` and `OP`, which shifts the position of one job and one project, respectively, were also assigned high probabilities. Note that if $\vartheta = 0.0$ and the current solution is not improved by any neighborhood within z iterations, all neighborhoods are assigned the same probability. Figure 4.16 shows that this situation is recurring. When sideways moves are rewarded ($\vartheta = 0.1$), the neighborhoods `ISJ, OJ` and `STJ`, which change the sequence of jobs, were given high selection probabilities during the entire search.

Figure 4.17 presents how probabilities evolve for a medium size instance, X-10.

We can see that neighborhoods `C1M`, `OJ`, `OP` remain the most significant ones. Other neighborhoods appear with high probabilities at the beginning, like `C2M`. When $\vartheta = 0.1$, the neighborhoods with higher potential to generate sideways moves are assigned higher probabilities.

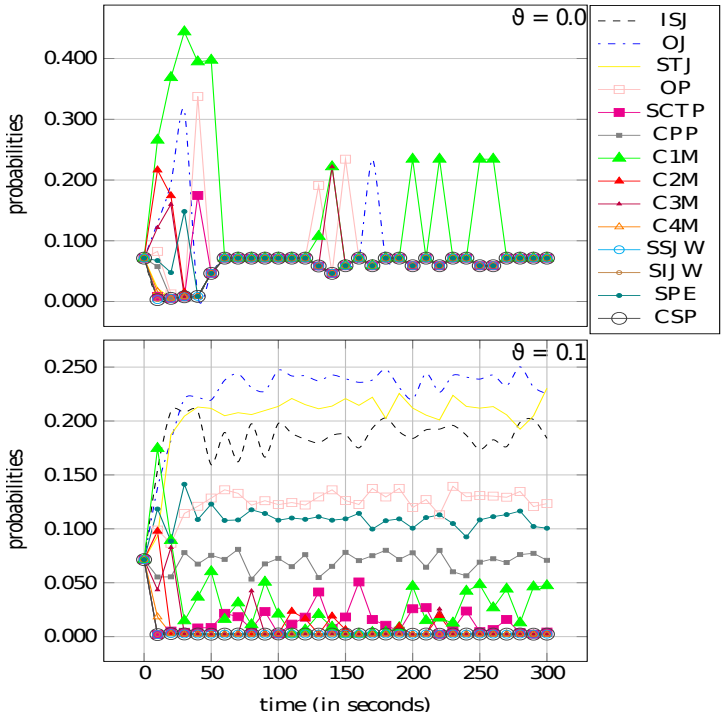

Figure 4.16: Evolution of the neighborhood selection probabilities over time considering instances B-9 and *online* tuning ($z = 1,000$ and $\beta = 0.01$)

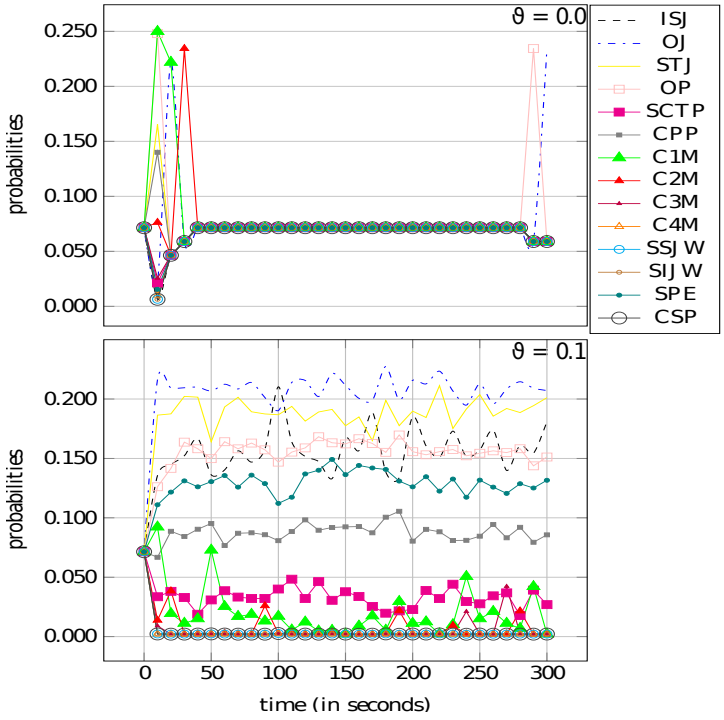

Figure 4.17: Evolution of the neighborhood selection probabilities over time considering instances X-10 and *online* tuning ($z = 1,000$ and $\beta = 0.01$)

Chapter 5

Mixed-Integer Linear Programming Based Methods

In this chapter, we introduce the time-indexed formulation based upon the discrete-time formulations proposed in Kolisch and Sprecher (1996) and Pritsker et al. (1969), as well as the preprocessing approaches. The mixed-integer linear programming approach, which more contributes to the RCPSP, with preprocessing procedures to generating stronger lower bounds and strategies of exact resolutions for the problem, are also presented. We focus on a computational technique called cut generation. We have developed stronger alternative formulations and their treatment with cut generation to improve MILP formulations for RCPSP dynamically.

5.1 Input Data

The following notation is used throughout this paper to describe the input data:

$\mathcal{P}$: set of all projects;

$\mathcal{J}$: set of all jobs;

$\mathcal{M}_j$: set of modes available for job $j \in \mathcal{J}$;

$\mathcal{J}_p$: set of jobs belonging to project p, such that $\mathcal{J}_p \subseteq \mathcal{J}\ \forall p \in \mathcal{P}$;

$\mathcal{K}$: set of non-renewable resources;

$\boldsymbol{\mathcal{R}}$: set of renewable resources;

$\boldsymbol{\mathcal{S}}$: set of direct precedence relationships between two jobs $(j, s) \in \mathcal{J} \times \mathcal{J}$;

$\boldsymbol{\mathcal{T}} \subset \mathbb{Z}^+$: set of time periods in the planning horizon for all projects $p \in \mathcal{P}$;

$\boldsymbol{\mathcal{T}_{jm}} \subset \boldsymbol{\mathcal{T}}$: time horizon for each job $j \in \mathcal{J}$ on mode $m \in \mathcal{M}_j$, defined after preprocessing;

$\boldsymbol{d_{jm}} \in \mathbb{Z}^+$: duration of job $j \in \mathcal{J}$ on mode $m \in \mathcal{M}_j$;

$\boldsymbol{q_{kjm}} \in \mathbb{Z}^+$: required amount of non-renewable resource $k \in \mathcal{K}$ to execute job $j \in \mathcal{J}$ on mode $m \in \mathcal{M}_j$;

$\boldsymbol{q_{rjm}} \in \mathbb{Z}^+$: required amount of renewable resource $r \in \mathcal{R}$ to execute job $j \in \mathcal{J}$ on mode $m \in \mathcal{M}_j$;

$\boldsymbol{\breve{q}_k} \in \mathbb{Z}^+$: available amount of non-renewable resource $k \in \mathcal{K}$;

$\boldsymbol{\breve{q}_r} \in \mathbb{Z}^+$: available amount of renewable resource $r \in \mathcal{R}$;

$\boldsymbol{\sigma_p} \in \boldsymbol{\mathcal{T}}$: release date of project p;

$\boldsymbol{a_p} \in \boldsymbol{\mathcal{J}_p}$: artificial job belonging to project $p \in \mathcal{P}$, which represents the end of the project.

5.1.1 Preprocessing Input Data

An effective way to reduce the search space is by identifying tight time windows in which it is valid to process jobs. A basic technique to define the earliest starting time $\breve{e}_j^s$ for jobs $j \in \mathcal{J}$ consists of computing the CPD using CPM (Kelley Jr and Walker, 1959) without considering resource constraints. This method allows to compute the $\breve{e}_j^s$ of all jobs, taking into consideration the precedence relationships. The longest path of a project, also known as the critical path, provides a lower bound for the completion time of each project.

Consider, for each project $p \in \mathcal{P}$, the release date σ_p, and lower bound based (i.e., the length of the critical path) λ_p as input data and the value β_p, an upper bound for each project p, obtained from any feasible solution. Optimality conditions can be used to restrict the set of valid time periods when a job can be allocated. We initially consider

the value α computed by Eq.(5.1), that represents an upper bound to the maximum total project delay allowed.

$$\alpha = \sum_{p \in \mathcal{P}} (\beta_p - \sigma_p - \lambda_p) \tag{5.1}$$

Thus, the maximum time period $\breve{t} \in \mathcal{T}$ that needs to be considered int the planning, can be obtained by Eq.(5.2).

$$\begin{aligned} \breve{t} &= \max_{p \in P} (\sigma_p + \lambda_p + \alpha) \\ \mathcal{T} &= \{0, ..., \breve{t}\} \end{aligned} \tag{5.2}$$

Analogously, upper bounds can be computed for processing times of jobs. The upper bounds can be strengthened if the selection of modes with different durations is also considered. The upper bounds are used, along with the duration of each job and without considering the resource constraints, to define the latest starting times ($\breve{l}^s_j$) for jobs $j \in \mathcal{J}$. A job j from a project p when processed at mode m will push forward (i.e., postpone) all successor jobs by exactly d_{jm} time units. Consider set $\overline{\mathcal{S}}_j$, containing the entire chain of successors of job j on the longest path from job j to the artificial job a_p (indicating the project completion). Let lower bound $\mathcal{L}_{jm}$ be the total duration in this path, computed considering only the fastest processing modes for each job in this chain. The maximum allocation time or latest starting time ($\breve{l}^s_{jm}$) for a job j from a project p when processing on mode m to $\mathcal{T}_{jm}$ is given by Eq.(5.3).

$$\begin{aligned} \breve{l}^s_{jm} &= \sigma_p + \lambda_p - \mathcal{L}_{jm} + \alpha \\ \mathcal{T}_{jm} &= \{\breve{e}^s_j, ..., \breve{l}^s_{jm}\} \end{aligned} \tag{5.3}$$

Similar bounds can be derived for any two jobs in this path also considering the fastest processing modes for all jobs except the first one:

$\breve{\boldsymbol{d}}_{\boldsymbol{jms}}$: the shortest path in the precedence graph considering the length of the arcs between job j and successor job $s \in \overline{\mathcal{S}}_j$ considering mode $m \in \mathcal{M}_j$;

$\breve{\boldsymbol{d}}^{\boldsymbol{*}}_{\boldsymbol{js}}$: the shortest path in the precedence graph considering the length of the arcs between job j and successor job $s \in \overline{\mathcal{S}}_j$ considering j fastest mode.

5.2 Formulation

Binary decision variables are used to select the mode and starting times for the jobs. They are defined as follows:

$$x_{jmt} = \begin{cases} 1 & \text{if job } j \in \mathcal{J} \text{ is allocated on mode } m \in \mathcal{M}_j \\ & \text{at starting time } t \in \mathcal{T}_{jm}; \\ 0 & \text{otherwise.} \end{cases}$$

We introduce in this formulation on/off discrete time variables studied in Artigues (2017) to allow resources constraints and cutting planes, detailed in the next sections, to be expressed with fewer variables. The following binary decision variables indicate during which time periods jobs are being processed:

$$z_{jmt} = \begin{cases} 1 & \text{if the job } j \in \mathcal{J} \text{ is allocated on mode } m \in \mathcal{M}_j \text{ and} \\ & \text{is being processed during time } t \in \mathcal{T}_{jm}; \\ 0 & \text{otherwise.} \end{cases}$$

The objective function minimizes the total project delay over the project completion times for projects and their critical paths. Consider the following integer variable included in the objective function:

$\boldsymbol{h} \in \mathbb{Z}^+$: integer variable used to compute the makespan, included in the objective function with a small coefficient ϵ to break ties.

Minimize:

$$\sum_{p \in \mathcal{P}} \sum_{m \in \mathcal{M}_{a_p}} \sum_{t \in \mathcal{T}_{a_p m}} \left[t - (\sigma_p + \lambda_p) \right] x_{a_p mt} + \epsilon h \tag{5.4}$$

subject to:

$$\sum_{m\in\mathcal{M}_j}\sum_{t\in\mathcal{T}_{jm}} x_{jmt} = 1 \quad \forall j \in \mathcal{J} \tag{5.5}$$

$$\sum_{j\in\mathcal{J}}\sum_{m\in\mathcal{M}_j}\sum_{t\in\mathcal{T}_{jm}} q_{kjm}x_{jmt} \leq \breve{q}_k \quad \forall k \in \mathcal{K} \tag{5.6}$$

$$\sum_{j\in\mathcal{J}}\sum_{m\in\mathcal{M}_j} q_{rjm}z_{jmt} \leq \breve{q}_r \quad \forall r \in \mathcal{R}, \forall t \in \mathcal{T} \tag{5.7}$$

$$\sum_{m\in\mathcal{M}_j}\sum_{t\in\mathcal{T}_{jm}} (t + d_{jm})\, x_{jmt} \; - \sum_{z\in\mathcal{M}_s}\sum_{i\in\mathcal{T}_{sz}} i x_{szi} \leq 0 \quad \forall j \in \mathcal{J}, \forall s \in \mathcal{S}_j \tag{5.8}$$

$$z_{jmt} - \sum_{t'=(t-d_{jm}+1)}^{t} x_{jmt'} = 0 \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}_j, \forall t \in \mathcal{T}_{jm} \tag{5.9}$$

$$h - \sum_{m\in\mathcal{M}_{a_p}}\sum_{t\in\mathcal{T}_{a_pm}} t x_{a_pmt} \geq 0 \quad \forall p \in \mathcal{P} \tag{5.10}$$

$$x_{jmt} \in \{0,1\} \quad \forall j \in \mathcal{J}, \forall m \in \mathcal{M}_j, \forall t \tag{5.11}$$

$$z_{jmt} \in \{0,1\} \; \forall j \in \mathcal{J}, \forall m \in \mathcal{M}_j, \forall t \in \mathcal{T}_{jm} \tag{5.12}$$

$$h \geq 0 \tag{5.13}$$

Constraints (5.5) ensure that each job is allocated to exactly one starting time and one mode. Constraints (5.6) and (5.7) are capacity constraints for non-renewable and renewable resources, respectively . Constraints (5.8) force precedence relationships to be satisfied. Constraints (5.9) link variables z and variables x. Constraints (5.10) compute the total makespan. Finally, constraints (5.11), (5.12) and (5.13) respectively ensure that variables x and z can only assume binary values and h can only assume nonnegative values.

5.2.1 Preprocessing MILP Formulation

Johnson et al. (1985) introduce an interesting preprocessing method to strengthen constraint coefficients using the knapsack structure of resource constraints. This preprocessing was used in Sankaran et al. (1999) for the SMRCPSP and analyzes one resource usage constraint at time. In this paper, we propose a preprocessing technique that considers various constraints (precedence and the usage of other renewable and non-renewable resources), besides the renewable resource constraint, which will be strengthened.

The proposed procedure to strengthen resource usage constraints (5.7) was inspired by Fenchel cutting planes (Boyd, 1994, 1992). Fenchel cutting planes are based on the enumeration of incidence vectors to find the most violated inequality for a subset of binary variables. In our paper, we enumerate feasible subsets of jobs and modes to create a linear problem to find the best possible strengthening of a given resource constraint.

First, it computes, for each t, a set $\mathcal{G}_t$ composed of all jobs and modes (j, m) available for processing at time $t \in \mathcal{T}_{jm}$ (see Algorithm 5.1, lines 1–2). Formally, these sets can be computed as stated in Eq.(5.14).

$$\mathcal{G}_t = \{\ (j,m)\ \in\ \mathcal{J} \times \mathcal{M}_j\ \mid\ t \in \mathcal{T}_{jm}\} \tag{5.14}$$

To illustrate the coefficient strengthening technique, consider for $t = 4$, the jobs and modes that make up the set $\mathcal{G}_4 = \{(1,0),(1,1),(1,2),(2,1),(2,2),(3,0),(5,0),(5,1),$ $(5,2),(6,1),(6,2),(9,1),(9,2)\}$. Consider also, the following original constraint restricting the usage of renewable resource r_0 at time $t = 4$:

$$0z_{1,0,4} + 0z_{1,1,4} + 0z_{1,2,4} + 0z_{2,1,4} + 0z_{2,2,4} + 3z_{3,0,4} + 0z_{5,0,4} + 0z_{5,1,4} + 2z_{5,2,4} + 0z_{6,1,4} + 6z_{6,2,4} + 0z_{9,1,4} + 9z_{9,2,4} \leq 9.$$

The next step is to enumerate all valid combinations of jobs and modes (j, m) that can be processed in parallel at time t, i.e., that satisfy all resources constraints and do not have precedence relations among each other. Mingozzi et al. (1998) designated these valid combinations as *feasible subsets*.

Let $\mathcal{E}_t = (\bar{e}_1, \bar{e}_2, \ldots, \bar{e}_n)$ be the set of all these feasible subsets. This set can be computed by using a simple backtracking algorithm that recursively proceeds, from level 0 to $|\mathcal{G}_t|$; tentatively fixing the allocation of each respective job and mode to 0 or

1; proceeding to the next level on the search space only when the partial fixation to the current level does not violate any resource or precedence constraint.

If we enumerate all the possibilities over $\mathcal{G}_4$, we could have $2^{13} = 8192$ feasible subsets. However, due to multiple constraints, there are only 51 feasible subsets in $\mathcal{E}_4$:

$\mathcal{E}_4$ = [{(1,0)}, {(2,2),(3,0),(1,0)}, {(2,2),(1,0)}, {(3,0),(1,0)}, {(1,1)}, {(2,2),(3,0),(1,1)}, {(2,2),(1,1)}, {(3,0),(1,1)}, {(1,2)}, {(2,1),(3,0),(1,2)}, {(2,1),(1,2)}, {(2,2),(3,0),(1,2)}, {(2,2),(1,2)}, {(3,0),(1,2)}, {(2,1)}, {(3,0),(5,2),(2,1)}, {(3,0),(2,1)}, {(5,2),(2,1)}, {(2,2)}, {(3,0),(5,0),(2,2)}, {(3,0),(5,1),(2,2)}, {(3,0),(5,2),(2,2)}, {(3,0),(2,2)}, {(5,0),(2,2)}, {(5,1),(2,2)}, {(5,2),(2,2)}, {(3,0)}, {(5,0),(9,1),(3,0)}, {(5,0),(3,0)}, {(5,1),(9,1),(3,0)}, {(5,1),(3,0)}, {(5,2),(9,1),(3,0)}, {(5,2),(3,0)}, {(6,1),(3,0)}, {(6,2),(9,1),(3,0)}, {(6,2),(3,0)}, {(9,1),(3,0)}, {(5,0)}, {(9,1),(5,0)}, {(9,2),(5,0)}, {(5,1)}, {(9,1),(5,1)}, {(9,2),(5,1)}, {(5,2)}, {(9,1),(5,2)}, {(6,1)}, {(9,2),(6,1)}, {(6,2)}, {(9,1),(6,2)}, {(9,1)}, {(9,2)}].

As an example of the impact of considering multiple constraints for reducing the number of valid incidence vectors, if we do not consider precedence constraints and we just consider one renewable resource constraint in the enumeration process, 182 feasible subsets would be built. For the maximal feasible subset {(2,2),(3,0),(1,0)}, highlighted above inside the box, if we do not consider all resources and precedence constraints, it will be extended to {(2,2),(3,0),(5,2),(1,0)}, {(2,2),(3,0),(6,2),(1,0)}.

The i^{th} feasible subset $\bar{e}_i$ contains ordered pairs $(j, m) \in \mathcal{G}_t$. For each renewable resource r with capacity c and time t the following linear program (W_{rt}) can be used to strengthen constraints (5.7) if the enumeration process is successful, i.e., a pre-defined maximum number of iterations (it) was not reached, (see Algorithm 5.1, lines 5–9). Consider the continuous variables u_{jm}, indicating the number of consumed units of resource r by job j at mode m in the strengthened constraint of the following linear programming (W_{rt} model):

Maximize:

$$\sum_{(j,m)\in\mathcal{G}_t} u_{jm} \tag{5.15}$$

Subject to:

$$\sum_{(j,m)\in\bar{e}} u_{jm} \leq c \ \forall \bar{e} \in \mathcal{E}_t \tag{5.16}$$

$$q_{rjm} \leq u_{jm} \leq c \ \forall (j,m) \in \mathcal{G}_t \tag{5.17}$$

Consider $\forall (r,t) \in \mathcal{R} \times \mathcal{T} : \bar{q}_{rjmt} = u^*_{jm}$ from W_{rt}, where u^*_{jm} is the optimal solution in W_{rt}. This value is introduced as a new input data for the main formulation:

$\bar{\boldsymbol{q}}_{\boldsymbol{rjmt}}$: new values for required amount of renewable resource $r \in \mathcal{R}$ to execute job $j \in \mathcal{J}$ on mode $m \in \mathcal{M}_j$ at time t.

Constraints (5.18) are created with the new values $\bar{q}_{rjmt}$, yielding improved capacity constraints for renewable resources.

$$\sum_{j\in\mathcal{J}} \sum_{m\in\mathcal{M}_j} \bar{q}_{rjmt} z_{jmt} \leq \breve{q}_r \quad \forall r \in \mathcal{R}, \ \forall t \in \mathcal{T} \tag{5.18}$$

Due to the bounds on variables u, constraints (5.18) always dominate the original constraints (5.7), since $\bar{q}_{rjmt} \geq q_{rjm}$. In particular, whenever $u_{jm} > q_{rjm}$ it dominates strictly. The following defines dominance between two generated cuts (Wolsey, 1998b):

Definition 5.2.1. Let ${c^1}^T x \leq r^1$ and ${c^2}^T x \leq r^2$ be two inequalities. We say ${c^1}^T x \leq r^1$ dominates ${c^2}^T x \leq r^2$ if $c^1_i \geq c^2_i \ \forall i$ and $r^1 \leq r^2$; if at least one of these inequalities is satisfied as an strict inequality, then there is a strict dominance.

An interesting property of this procedure is that it may strengthen constraints of resources that are *not scarce*, given the scarceness of other resources and/or precedence constraints.

Example By solving the MILP model $\mathcal{W}_{0,4}$ to our example introduced above, the coefficient of the emphasized variable $\boldsymbol{z_{5,2,4}}$ corresponding to job 5 on mode 2 processed at $t = 4$ can be strengthened to 6, without excluding any feasible integer solution. We

can generate the following strengthened constraint:

$$0z_{1,0,4} + 0z_{1,1,4} + 0z_{1,2,4} + 0z_{2,1,4} + 0z_{2,2,4} + 3z_{3,0,4} + 0z_{5,0,4} + 0z_{5,1,4} + \mathbf{6z_{5,2,4}} + 0z_{6,1,4} + 6z_{6,2,4} + 0z_{9,1,4} + 9z_{9,2,4} \leq 9.$$

For the SMRCPSP and MMRCPSP, the original constraints (5.7) can be replaced by new constraints presented in (5.18) in the case that the latter are stronger. For the MMRCMPSP, the new constraints (5.18) are created to strengthen renewable resources for each project separately, and the original constraints remain in the model.

The strengthening procedure is presented in Algorithm 5.1. We consider a time limit tl, which will be checked after the subset enumeration procedure for each t. If the time limit is reached, the remaining time periods t will be skipped and the algorithm is terminated (see Algorithm 5.1, lines 10–11). We continue to find feasible subsets while it does not reach the last element of $\mathcal{G}_t$. If the maximum number of iterations it is reached, we stop the process (by returning $\emptyset$, see Algorithm 5.1, lines 4–5) and continue to the next value of t on the strengthening algorithm.

Algorithm 5.1: `strengthening_procedure`

Data: RCPSP model $\mathcal{M}$, it iteration limit, tl time limit, set $\mathcal{J}$, set $\mathcal{M}_j$, set $\mathcal{T}$, set $\mathcal{R}$, set $\mathcal{K}$, set $\overline{\mathcal{S}}_j$

1 **for** *(each $t \in \mathcal{T}$)* **do**
2 $\quad \mathcal{G}_t \leftarrow$ `compute_by_Eq.5.14(`$\mathcal{J}$`,`$\mathcal{M}$`,`t`)`;
3 $\quad \mathcal{E}_t \leftarrow \emptyset$; $ite \leftarrow 0$;
4 $\quad \mathcal{E}_t \leftarrow$ `subset_bt(`it`,`ite`,`tl`,`$\mathcal{G}_t$`,`$\mathcal{R}$`,`$\mathcal{K}$`,`$\overline{\mathcal{S}}_j$`,`$\mathcal{E}_t$`)`;
5 $\quad$ **if** *($\mathcal{E}_t \neq \emptyset$)* **then**
6 $\quad\quad$ **for** *(each $r \in \mathcal{R}$ with capacity c)* **do**
7 $\quad\quad\quad \mathcal{W}_{rt} \leftarrow$ `create_strengthening_MIP(`r`,`c`,`$\mathcal{G}_t$`,`$\mathcal{E}_t$`)`;
8 $\quad\quad\quad \bar{q}_{rjmt} \leftarrow$; `opt(`$\mathcal{W}_{rt}$`)`;
9 $\quad\quad\quad$ `create_replace_constraints(Ineq.(5.7),` $\bar{q}_{rjmt}$`, Ineq.(5.18))`;
10 $\quad$ **if** *time limit tl is reached* **then**
11 $\quad\quad$ **break**;

5.3 The Cutting Plane Algorithm

The performance of general purpose MILP solvers on a given formulation strongly depends on how tight is the LP relaxation (dual bound) to the optimal solution (Wolsey, 1998b). Cutting planes are commonly used to improve this bound by iteratively adding violated cuts.

The proposed cutting plane algorithm uses traditional RCPSP cuts enhanced with new lifting techniques (see Subsections 5.3.1 and 5.3.2, respectively for cover and precedence cuts), conflict-based cuts (see Subsection 5.3.3 for clique and odd-holes cuts) and strengthened Chvátal-Gomory cuts (see Subsection 5.3.4) generated from an implicit dense conflict dynamic graph.

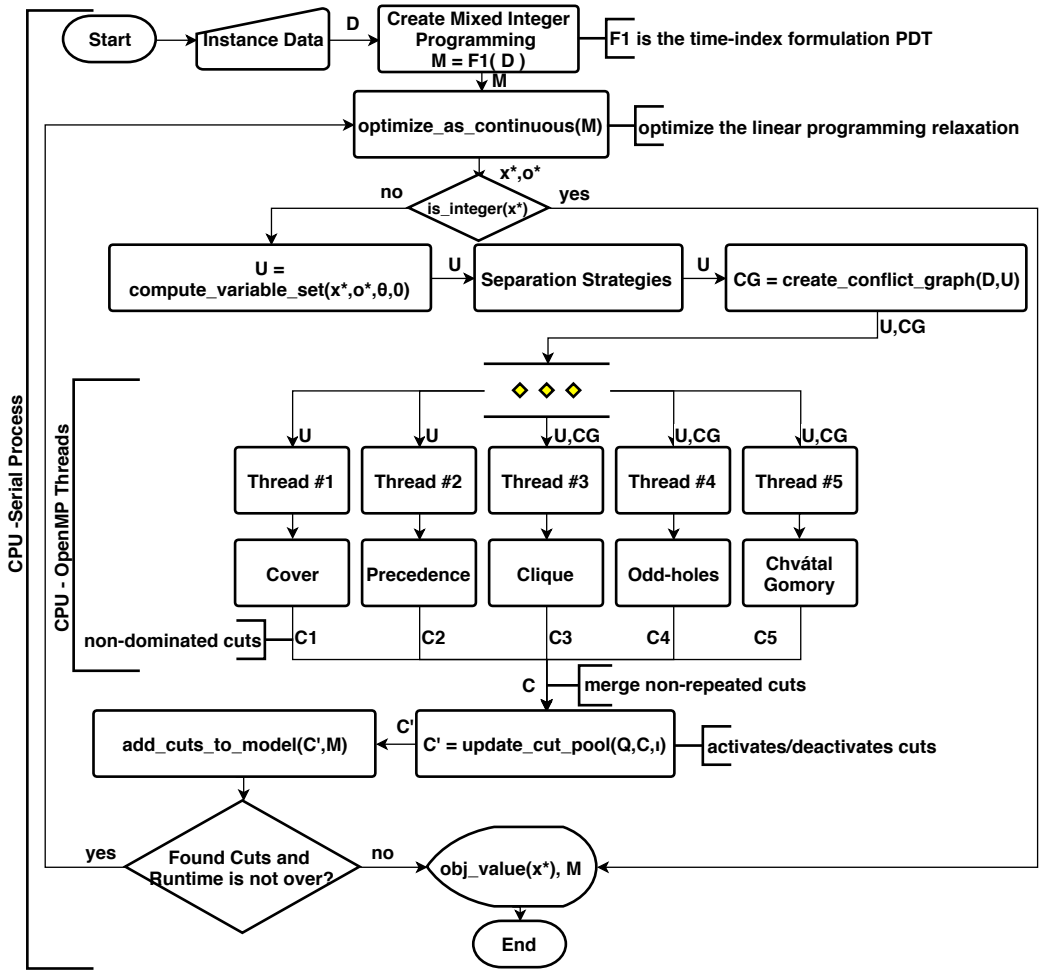

Figure 5.1: Execution flow of the proposed cutting plane method

An outline of the proposed method is depicted in Figure 5.1. After the instance data is read, we create the mathematical programming model (**M**) explained in Section 5.2 and execute the preprocessing routines. Then, the optimal LP relaxation is computed and if a fractional solution is obtained, different search methods are started to separate violated inequalities. Since the separation of these inequalities may involve the solution of $\mathcal{NP}$-hard problems, we execute the separation procedures in parallel. This allows us to save some processing time in order to process a larger number of iterations in the cutting plane algorithm within the given time limit.

All generated cuts are inserted into a cut pool where repeated inequalities are discarded. Our algorithm quickly discards repeated cuts by using a hash table. While checking for repeated cuts is very fast, the dominance check is slower since it requires checking the contents of the cuts. We only check the dominance in the pool of cover separation since they generated less cuts compared with other types of cuts.

If new cuts have been found after the separation procedure, a stronger formulation is obtained, and the process is repeated. When the time limit is reached or when no further cut is generated, the strengthened model and its objective function value are returned. In the following, we present the different inequalities that are separated as well as the algorithmic aspects involved in their separation.

To better understand the separation strategies, remember the example introduced in Chapter 2, in which Figure 5.2 illustrates the example instance $j102\_4.mm$, a small instance for the MMRCPSP variant.

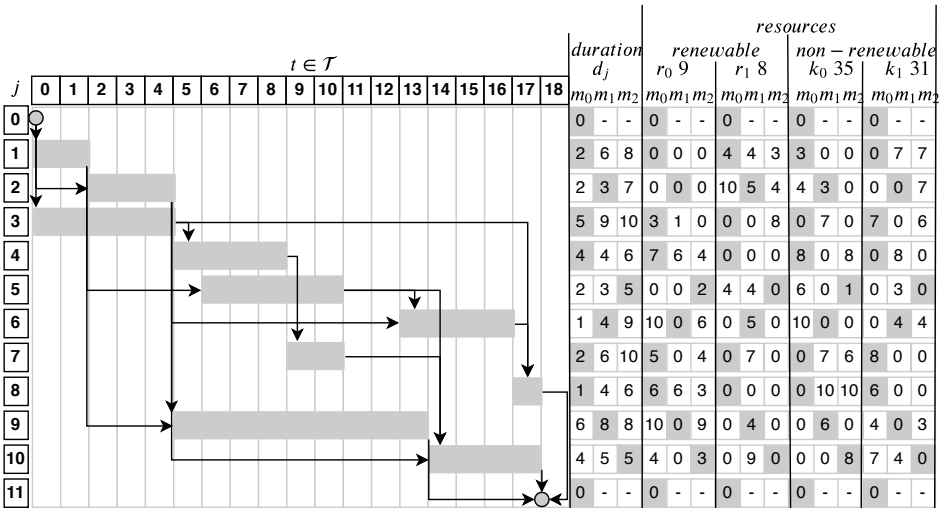

Figure 5.2: An optimal solution for instance $j102\_4.mm$ and its characteristics

5.3.1 Lifted RCPSP Knapsack Cover Cuts (LCV)

Resource usage constraints present a knapsack structure which was exploited in Zhu et al. (2006) to generate GUB cover (CV) cuts. Consider the general case of a constraint in the form $\sum_{j\in\mathcal{N}} b_j z_j \leq c, (b,c) \in \mathbb{N}^{+n} \times \mathbb{N}^{+}$ for a set $\mathcal{N}$ of binary variables. The following knapsack problem can be solved to generate a valid inequality that cuts fractional point z^*:

Minimize

$$\gamma(z^*) = \sum_{j\in\mathcal{N}} (1 - z_j^*) v_j \tag{5.19}$$

subject to:

$$\sum_{j\in\mathcal{N}} b_j v_j > c \tag{5.20}$$

$$v_j \in \{0,1\} \forall j \in \mathcal{N} \tag{5.21}$$

Whenever a solution with $\gamma(z^*) < 1$ is found involving a set $\mathcal{V} = \{j \in \mathcal{N} : v_j = 1\}$ a violated inequality in the form $\sum_{j\in\mathcal{V}} z_j \leq |\mathcal{V}| - 1$ is generated. Since only active variables ($z_j^* > 0$) are considered in this separation, many dominated inequalities can be generated in different iterations.

While general purpose lifting strategies (Gu et al., 2000) can be used, specific problem information can provide an effective procedure to produce lifted cover inequalities. A variable in a traditional cover inequality represents whether a job j and mode m were allocated or not at time t. The proposed lifted RCPSP knapsack cover cut (LCV) separation routine may include, for each job j, variables of additional processing modes without increasing the right-hand-side, producing much stronger cuts. In the single-mode case the strengthening will be similar to the one obtained with traditional lifting (see Balas (1975); Balas and Zemel (1978); Nemhauser and Vance (1994)) techniques.

The new lifted knapsack cover separation problem for the RCPSP is solved for each renewable resource and each time period. Consider period t^*, resource r with capacity $\breve{q}_r$ and a fractional solution with variable values $z_{jmt^*}^*$. The decision variables are:

$\boldsymbol{v_{jm}} \in \mathbf{\{0,1\}}$: if variable of job j at mode m is selected (1) or not (0);

$\boldsymbol{o_j} \in \mathbf{\{0,1\}}$: if at least one mode for job j is included in the cut (1) or not (0);

$\boldsymbol{\underline{w}_{jm}} \in \mathbf{\{0,1\}}$: if job j has m as the selected mode with the smallest resource consumption from the selected ones (1) or not (0);

$\boldsymbol{\underline{e}} \in \mathbb{Z}^+$: resource consumption excess if modes with smallest resource consumption are selected.

$\boldsymbol{\overline{v}} \in \mathbb{R}^+$: cut violation.

The LCV separation problem is given by:

Maximize

$$\omega\overline{v} + \sum_{j\in\mathcal{J}}\sum_{m\in\mathcal{M}_j} \mu v_{jm} \tag{5.22}$$

subject to:

$$o_j = \sum_{m\in\mathcal{M}_j} \underline{w}_{jm} \quad \forall j \in J \tag{5.23}$$

$$\underline{e} = \sum_{j\in\mathcal{J}}\sum_{m\in\mathcal{M}_j} q_{rjm}\underline{w}_{jm} - \breve{q}_r \tag{5.24}$$

$$v_{jm} \leq \sum_{m'\in\mathcal{M}_j : q_{rjm'} \leq q_{rjm}} \underline{w}_{jm'} \quad \forall j \in \mathcal{J}, m \in \mathcal{M}_j \tag{5.25}$$

$$\overline{v} = \sum_{j\in\mathcal{J}}\sum_{m\in\mathcal{M}} z^*_{jmt^*} v_{jm} - \sum_{j\in\mathcal{J}} o_j + 1 \tag{5.26}$$

$$0.005 \leq \overline{v} \leq \infty \tag{5.27}$$

$$1 \leq \underline{e} \leq \infty \tag{5.28}$$

$$o_j, v_{jm}, \underline{w}_{jm} \in \{0,1\} \quad \forall j \in \mathcal{J},\ \forall m \in \mathcal{M}_j \tag{5.29}$$

The objective function (5.22) maximizes a hierarchical objective function composed of, first, the cut violation and, second, the inclusion of additional jobs and modes in the generated inequality to produce stronger cuts ($\omega \gg \mu$). These weights cannot be too large or too small to prevent numerical instability in the solvers. Constraints (5.23) ensure that o_j is only activated when some mode is selected for job j as the mode with

lower resource usage. Constraints (5.24) and (5.28) ensure that a cover is produced. Constraints (5.25) ensure that only modes with resource usage greater than or equal to the mode with the smallest resource usage selected are allowed for lifting. Equality (5.26) computes the cut violation and constraint (5.27) ensures that a violated cut is produced. Finally, constraints (5.29) ensure that variables o_j, v_{jm} and $\underline{w}_{jm}$ can only assume binary values.

Let C be the set of selected jobs and modes (with $v_{jm} = 1$) in the solution of the problem above. Further, let o_j indicate its corresponding solution values. We may generate they following LCV cut:

$$\sum_{(j,m)\in C} z_{jmt} \leq \sum_{j\in J} o_j - 1 \quad \forall t \in \mathcal{T} \tag{5.30}$$

For a valid cover cut the value on the right-hand-side would be the size of the set minus 1, with the lifting strategy whenever it chooses more than one mode per job the sum of the right-hand-side will be strictly smaller than the size of the set minus 1, generating a stronger cut as in the example below.

Example Consider the following cut for resource r_0 on $t = 8$, and jobs {4,8} processing respectively on modes {0,2} from Figure 2.1, generated using the separation described in (5.19)-(5.21):

$$(CV) = z_{4,0,8} + z_{8,2,8} \leq 1.$$

This cut can be lifted since job 8 has other modes $\{0, 1\}$, each of them consuming 6 units of resource r_0, more than the current mode {2} that consumes 3 units, forming still a valid inequality:

$$(LCV) = z_{4,0,8} + \boldsymbol{z_{8,0,8}} + \boldsymbol{z_{8,1,8}} + z_{8,2,8} \leq 1.$$

It is important to emphasize that the first component of the objective function maximizes the violation considering the consumption of the other modes, and the second component, even for the single-mode version, will add additional variables that do not contribute to the cut violation but which contribute to generate a stronger inequality.

5.3.2 Lifted Precedence Based Cuts (LPR)

In addition to the cover cut introduced above, it is possible to further strengthen the formulation by analyzing the precedence between jobs and using precedence (PR) cuts similar to those used by Zhu et al. (2006). Consider a job j, its (direct or indirect) successor s and a time t. Also, consider the following constants $e_j = \min_{m \in \mathcal{M}_j}(\mathcal{T}_{jm})$ and $l_j = \max_{m \in \mathcal{M}_j}(\mathcal{T}_{jm})$ to limit the time period. We introduce a new lifted version for the precedence cuts, shown in Eq.(5.31). Consider the lengths of the shortest paths on the precedence graph, $\breve{d}_{jms}$ and $\breve{d}^*_{js}$, introduced in Section 5.1.1. The following inequalities are valid:

$$\sum_{m \in \mathcal{M}_j} \sum_{t'=e_j}^{t+\breve{d}^*_{js}-\breve{d}_{jms}} x^*_{jmt'} \geq \sum_{m \in \mathcal{M}_s} \sum_{t'=e_s}^{min(l_s, t+\breve{d}^*_{js})} x^*_{smt'}$$

$$\forall j \in \mathcal{J}, s \in \overline{\mathcal{S}}_j, t \in \{\max(e_j, e_s - \breve{d}^*_{js}), \ldots, min(l_j, l_s - \breve{d}^*_{js})\} \quad (5.31)$$

Example Consider the following two cuts. The first one, (PR), was generated with the RCPSP precedence cut as proposed in Zhu et al. (2006). The second one, (LPR), is the lifted inequality (5.31) to predecessor job 5 and its successor 10 at time period 9:

$$\begin{aligned}
(PR) = &-x_{5,0,2} - x_{5,0,3} - x_{5,0,4} - x_{5,0,5} - x_{5,0,6} - x_{5,0,7} - x_{5,0,8} - x_{5,0,9} - \\
&\boxed{x_{5,0,10} - x_{5,0,11}} - x_{5,1,2} - x_{5,1,3} - x_{5,1,4} - x_{5,1,5} - x_{5,1,6} - x_{5,1,7} - \\
&x_{5,1,8} - \boxed{x_{5,1,9} - x_{5,1,10} - x_{5,1,11}} - x_{5,2,2} - x_{5,2,3} - x_{5,2,4} - x_{5,2,5} - \\
&x_{5,2,6} - \boxed{x_{5,2,7} - x_{5,2,8} - x_{5,2,9}} + x_{10,0,11} + x_{10,2,11} \leq 0; \\
(LPR) = &-x_{5,0,2} - x_{5,0,3} - x_{5,0,4} - x_{5,0,5} - x_{5,0,6} - x_{5,0,7} - x_{5,0,8} - x_{5,0,9} - \\
&x_{5,1,2} - x_{5,1,3} - x_{5,1,4} - x_{5,1,5} - x_{5,1,6} - x_{5,1,7} - x_{5,1,8} - x_{5,2,2} - \\
&x_{5,2,3} - x_{5,2,4} - x_{5,2,5} - x_{5,2,6} + x_{10,0,11} + x_{10,2,11} \leq 0.
\end{aligned}$$

Notice the change in the coefficients of the variables on the left-hand-side of the lifted cut (LPR), based on Eq.(5.31) and corresponding to the variables of job 5. In the case of different durations for processing modes of a job j, there will be an increase in the sum of the coefficients of the left-hand-side of (LPR) since we use $\breve{d}_{jms}$ instead of just considering the fastest mode given by $\breve{d}^*_{js}$ (as proposed in Zhu et al. (2006)). The original cut variant (PR) from Zhu et al. (2006) considers the fastest time. The variables highlighted in the boxes in the (PR) cut are not included in our cut approach (LPR).

The lifted cut therefore strictly dominates the original cut, given that the highlighted variables have coefficients 0 (instead of -1) in the cut stated as $\leq$ inequality.

Our separation procedure (see Algorithm 5.2) therefore selects different paths (line 2) that connect job j and the artificial project completion job a_p. In particular, it is desirable to avoid redundant constraints, i.e., constraints of predecessor/successor jobs belonging to the same path in the dependency graph with a similar meaning, for jobs on subpaths that have already been used in previously added precedence cuts. Whenever a violated precedence cut is found on this path (lines 8–13), the remaining jobs in this path are skipped (line 15). We limit the number ζ of precedence cuts that can be added per round and only add the most violated cuts (line 18).

Algorithm 5.2: `lifted_precedence_based_cuts`

Data: set of jobs $\mathcal{J}$, artificial project completion job a_p, maximum number of cuts ζ

```
1  for (each j ∈ 𝒥) do
2      P̆ ← compute_paths(j, a_p);
3      pos ← 0;
4      for (each p̆ ∈ P̆) do
5          s ← successor(j, p̆, pos);
6          found ← 0;
7          for (t ∈ {max(e_j, e_s − d̄*_js), ..., min(l_j, l_s − d̄*_js)}) do
8              c ← init_cut();
9              (V, C) ←identifies a set of variables and their coefficients that generate
                   a violated precedence cut for the predecessor j and successor s at time
                   period t;
10             if (V ≠ ∅) then
11                 add_set_var(c, V, C);
12                 C ← C ∪ {cut(c, ≥, 0 )};
13                 found ← 1;
14         if (found) then
15             skipp the remaing jobs in the path from j to a_p;
16         else
17             pos ← pos + 1;
18 sort_cuts_by_highest_violation_and_filters(𝒞, ζ);
19 return 𝒞;
```

5.3.3 Conflict-Based Cuts: Cliques (CL) and Odd-Holes (OH)

According to Padberg (1973), LP relaxations for problems that mostly contain binary variables linked to generalized upper bound (GUB) constraints can be significantly strengthened by the inclusion of inequalities derived from the set packing polytope (SPP). Generally, clique and odd-holes cuts can be generated using Conflict Graphs ($\mathcal{CG}$). The denser the $\mathcal{CG}$, the more inequalities can be generated. The disadvantage of having dense $\mathcal{CG}$ is that they can be prohibitively large (Atamtürk et al., 2000), so our algorithm creates the $\mathcal{CG}$ dynamically at each iteration by considering a set ($\mathcal{U}$) that contains the variables of interest, i.e., variables that have a non-zero value in the LP relaxation or variables set to zero but with a small reduced cost.

Several well-known inequalities can be generated considering pairwise conflicts between binary variables stored in $\mathcal{CG}$. Some conflicts can be easily detected by solvers considering constraints such as (5.5). Other conflicts can be implied from optimality conditions or by analyzing problem specific structures. Overall, the denser is the $\mathcal{CG}$, stronger are the produced cuts.

The dynamic dense $\mathcal{CG}$ created is used in a separation procedure for inequalities derived from a common class of cuts for the SPP: cliques and odd-holes. A clique inequality for a set C of conflicting variables has the form $\sum_{j \in C} x_j \leq 1$ and an odd-hole inequality for a cycle C can be defined as: $\sum_{j \in C} x_j \leq \lfloor \frac{|C|}{2} \rfloor$ (Santos et al., 2016).

Santos et al. (2016) present a clique separation routine that separates all violated cliques into a conflict subgraph induced by fractional variables. The authors then present a lifting that extends generated cliques considering the original $\mathcal{CG}$. They also present a strengthening of odd-holes inequalities by the inclusion of a so-called wheel center. For an odd-hole with variables C and W being the set of candidates to be included as wheel centers of C, the inequality (5.32) is valid:

$$\sum_{j \in W} \lfloor \frac{|C|}{2} \rfloor x_j + \sum_{j \in C} x_j \leq \lfloor \frac{|C|}{2} \rfloor \tag{5.32}$$

Our approach presented in Algorithm 5.3 considers four conflict types for variables x_{jmt}:

1. conflicts between variables of the same job (lines 6–7);

2. conflicts involving jobs that if allocated at the same time exceed the capacity of available renewable resources (lines 8–10);

3. conflicts based on precedence relations, in which the time window between the predecessor job j on mode m and some $s \in \overline{\mathcal{S}}_j$ is smaller than $\breve{d}_{jms}$ (lines 11–14);

4. conflicts considering jobs of different projects, where the sum of the delays generated by allocating these jobs in specific positions implies a total delay greater than α (lines 15–16).

Algorithm 5.3: `creating_conflict_graph`

Data: variables set $\mathcal{U}$, set $\mathcal{R}$, set $\bar{\mathcal{S}}$, delay α
Result: Conflict Graph $\mathcal{CG}$

1 $\mathcal{CG} \leftarrow \emptyset$;
2 **for** *($v_1 \in \mathcal{U}$)* **do**
3 $j_1 \leftarrow \texttt{job}(v_1)$; $m_1 \leftarrow \texttt{mode}(v_1)$; $t_1 \leftarrow \texttt{time}(v_1)$;
4 **for** *($v_2 \in \mathcal{U}$)* **do**
5 $j_2 \leftarrow \texttt{job}(v_2)$; $m_2 \leftarrow \texttt{mode}(v_2)$; $t_2 \leftarrow \texttt{time}(v_2)$;
6 **if** *($j_1 = j_2$)* **then**
7 `add_var_conf`$(\mathcal{CG}, v_1, v_2)$; **continue**;
8 **if** *($t_1 = t_2$)* **then**
9 **for** *($r \in \mathcal{R} : (q_{rj_1m_1} + q_{rj_2m_2} > \breve{q}_r)$)* **do**
10 `add_var_conf`$(\mathcal{CG}, v_1, v_2)$; **continue**;
11 **if** *($j_2 \in \bar{\mathcal{S}}_{j_1}$)* **then**
12 $w \leftarrow \bar{d}^{*}_{j_1,j_2} - \texttt{job\_min\_dur}(j_1)$;
13 **if** *((**end_time**$(j_1) > t_2$) or ($t_2 -$ **end_time**$(j_1) < w$))* **then**
14 `add_var_conf`$(\mathcal{CG}, v_1, v_2)$; **continue**;
15 **if** *(**proj**(j_1) != **proj**(j_2) & (**delay**(j_1, m_1, t_1) + **delay**$(j_2, m_2, t_2)) > \alpha$)* **then**
16 `add_var_conf`$(\mathcal{CG}, v_1, v_2)$;
17 **return** $\mathcal{CG}$;

After the creation of the $\mathcal{CG}$, cuts can be generated considering the current fractional solution. In this paper, we use the routines described in Brito et al. (2015), where cliques and odd-holes are exactly separated and lifted.

Example Considering the following clique cut:

$$(CL) = x_{5,0,12} + x_{5,1,11} + x_{5,2,9} + x_{10,0,11} + x_{10,0,12} + x_{10,0,13} + x_{10,2,11} + x_{10,2,12} + x_{10,2,13} \leq 1.$$

It is possible to observe, from Figure 2.1, that the variables corresponding to jobs 5 and 10 have conflicts at different time periods considering different modes, given that they have a precedence relation.

Example Consider the following odd-hole cut:

$$(OH) = x_{1,0,0} + x_{1,0,5} + x_{2,1,0} + x_{2,1,3} + x_{5,0,3} \leq 2.$$

Still referring to job 5 in Figure 2.1, we can observe conflicts from the precedence relationship with job 1. Also, a low amount of resources available at times when jobs 5 and 2 intersect on the variables of the example above reflect conflicts between them. Thus, these three jobs can not be allocated in parallel, considering the amount of resources consumed by their modes.

5.3.4 Strengthened Chvátal-Gomory Cuts (SCG)

Chvátal-Gomory (CG) (Chvátal, 1973) cuts are well-known cutting planes for MILP models. The inclusion of these cuts allows to significantly reduce the integrality gaps, even when only rank-one cuts are employed, i.e., those obtained from original problem constraints (Fischetti and Lodi, 2007).

Consider the integer linear programming (ILP) problem as $\min\{\vec{c}^T\vec{x} : A\vec{x} \leq \vec{b}, \vec{x} \geq 0$ integer$\}$, where $A \in \mathbb{R}^{mxn}$, $\vec{b} \in \mathbb{R}^m$, and $\vec{c} \in \mathbb{R}^n$, with the two associated polyhedra $P := \{\vec{x} \in \mathbb{R}^n_+ : A\vec{x} \leq \vec{b}\}$ and $P_{\mathcal{I}} := conv\{\vec{x} \in \mathbb{Z}^n_+ : A\vec{x} \leq \vec{b}\} = conv(P \cap \mathbb{Z}^n)$ with $\vec{x}$ being integer variables. Consider $\mathcal{I}$ and $\mathcal{H}$ the sets of constraints and variables, respectively.

A CG cut is defined as a valid inequality for $P_{\mathcal{I}}$: $\lfloor \vec{u}^T A \rfloor \vec{x} \leq \lfloor \vec{u}^T \vec{b} \rfloor$, where $\vec{u} \in \mathbb{R}^m_+$ is a multiplier vector. The choice of $\vec{u} \in \mathbb{R}^+$ is crucial to deriving useful inequalities. Fischetti and Lodi (2007) propose the MILP model for CG separation. The maximally violated $\sum_{j \in \mathcal{H}(x^*)} a_j x_j \leq a_0$ inequality can be found by optimizing the following separa-

tion MILP model:

Maximize:

$$\sum_{j \in \mathcal{H}(x^*)} a_j x_j^* - a_0 \tag{5.33}$$

subject to:

$$f_j = \vec{u}^T A_j - a_j, \ \forall j \in \mathcal{H}(x*) \tag{5.34}$$

$$f_0 = \vec{u}^T \vec{b} - a_0 \tag{5.35}$$

$$0 \leq f_j \leq 1 - \delta \ \forall j \in \mathcal{H}(x*) \cup \{0\} \tag{5.36}$$

$$-1 + \delta \leq u_i \leq 1 - \delta \ \forall i = 1, \ldots, m \tag{5.37}$$

$$a_j \ \in \ \mathbb{Z}, \ \forall j \in \mathcal{H}(x*) \tag{5.38}$$

where $\mathcal{H}(x^*) := \{j \in 1, \ldots, \breve{n} : x_j^* > 0\}$ and x^* are fractional values for all $\breve{n}$ variables of an LP solution for a general problem fixed in the MILP. To strengthen the cut, a penalty term $\sum_i w_i u_i$, with $w_i = 10^{-4}$ for all i, is applied to the objective function. To improve the numerical accuracy of the method, multipliers too close to 1 are forbidden ($u_i \leq 0.99, \forall i$).

Example Consider the following cut generated with the CG for jobs from Figure 2.1:

$$(CG) = 3x_{1,0,8} + x_{5,0,5} + 2x_{5,0,6} + 4x_{6,1,4} + 2x_{6,1,8} + 2x_{8,1,7} + 2x_{9,1,7} \leq 5.$$

On the one hand, the larger the set of non-redundant and tight constraints considered in the Chvátal-Gomory separation the more likely it is that violated inequalities will be found. On the other hand, large separation problems can be hard to solve and the overall performance of the cutting plane method can degrade. The following subsection will therefore consider specific strategies to find suitable sets of constraints.

Finding a Set of Constraints

In our approach we consider a tuple $(\bar{s}, \bar{f})$ to indicate the starting time and the finishing time of a given interval with size η to filter the constraints and variables sets. We com-

pute, for different intervals $(\bar{s}, \bar{f})$, the summation of all infeasibilities for the integrality constraints for all their variables x_{jmt} where $\bar{s} \leq t \leq \bar{f}$, x^*_{jmt} are the fractional values for the RCPSP variables. The value $\hat{f}$ is composed of the sum of the nearest integer distance, to indicate how fractional the variables in that interval are.

$$\hat{f}_{(\bar{s},\bar{f})} = \sum_{j \in \mathcal{J}} \sum_{m \in \mathcal{M}_j} \sum_{t=\bar{s}}^{\bar{f}} |round(x^*_{jmt}) - x^*_{jmt}| \quad \forall\ (\bar{s}, \bar{f}) \in \mathcal{T} \tag{5.39}$$

To find the most fractional interval $\hat{f}_{(\bar{s},\bar{f})}$ in the scheduling, in which the constraints with their respective variables will be chosen, we start from the beginning of the scheduling, and we slide the interval $(\bar{s}, \bar{f})$ until the end of the scheduling (see Algorithm 5.4, line 2). The parameters on the algorithm indicate the interval size η, the jump size ι to go to the next interval and a percentage ζ that allows sliding the interval. The slide only occurs if the violation is greater than the current value, plus the percentage of ζ. Another input data is the dynamic conflict graph $\mathcal{CG}$.

Preliminary experiments showed that cuts separated using constraints from the beginning of the time horizon were more effective for improving the dual bound, jobs allocated in the first time period are responsible for pushing the allocations of the others in the precedence graph. It is therefore desirable to find integer values for the first ones. Once the most fractional interval is found, the most important constraints are identified to compose the set for the Chvátal-Gomory separation.

The renewable resources directly impact the duration of the projects; therefore, all the variables of these constraints into the interval $(\bar{s}, \bar{f})$ are considered in the set $\mathcal{V}$ (see Algorithm 5.4, lines 3-7). Further constraints from the original problem are included in the separation problem whenever they are related to the current time window: constraints that restrict the choice of only job (lines 8-12) and non-renewable resource constraints (lines 13-17). Functions `nonrenewable_resources`($\mathcal{V}$) and `jobs`($\mathcal{V}$) returns, respectively, the set of non-renewable resources and jobs where variables of $\mathcal{V}$ appear. A good strategy is to find additional constraints that represent conflicts between the variables of set $\mathcal{V}$. A main conflict is analyzed, whereby the time window comprising variables coming from the conflict graph $\mathcal{CG}$ generated dynamically at each iteration of the cutting plane received as input parameter (lines 18-22). Function `pairs_of_conflicting_variables`($\mathcal{CG}, \mathcal{V}$) returns the pairs of these conflicting vari-

Algorithm 5.4: `finding_set_constraints`

Data: fractional RCPSP solution x^*, rows set $\mathcal{I}$, conflict graph $\mathcal{CG}$, size interval η, size jump ι, percentage allowed to slide the interval ζ
Result: Set $\mathcal{Z}$ of selected rows

$\mathcal{V} \leftarrow \emptyset$;
$(\bar{s}, \bar{f}) \leftarrow$ `identify_interval`$(x^*, \eta, \iota, \zeta)$;
for $((r_1 \in \mathcal{I})$ *and* `is_in_interval`$(r_1, \bar{s}, \bar{f}))$ **do**
    $type \leftarrow$ `type_row`(r_1);
    if $(type = Ineq.5.7)$ **then**
        `add_row`$(\mathcal{Z}, r_1)$;
        $\mathcal{V} \leftarrow \mathcal{V} \cup$ `vars_row`(r_1);
$O \leftarrow$ `jobs`$(\mathcal{V})$;
for $(j \in O)$ **do**
    $\mathcal{V}_j \leftarrow$ `vars_jobs`$(\mathcal{V}, j)$;
    $r_2 \leftarrow$ `create_constraints`$(\leq, 1)$;
    `add_set_var`$(r_2, \mathcal{V}_j)$; `add_row`$(\mathcal{Z}, r_2)$;
$\mathcal{K} \leftarrow$ `nonrenewable_resources`$(\mathcal{V})$;
for $(k \in \mathcal{K})$ **do**
    $\mathcal{V}_k \leftarrow$ `vars_nonrenewable_resources`$(\mathcal{V}, k)$;
    $r_3 \leftarrow$ `create_constraints`$(\leq,$ `capacity`$(k))$;
    `add_set_var`$(r_3, \mathcal{V}_k)$; `add_row`$(\mathcal{Z}, r_3)$;
$\mathcal{C} \leftarrow$ `pairs_of_conflicting_variables`$(\mathcal{CG}, \mathcal{V})$;
for $(c \in \mathcal{C})$ **do**
    $\mathcal{V}_c \leftarrow$ `vars_conflicts`$(\mathcal{V}, c)$;
    $r_4 \leftarrow$ `create_constraints`$(\leq, 1)$;
    `add_set_var`$(r_4, \mathcal{V}_c)$; `add_row`$(\mathcal{Z}, r_4)$;
return $\mathcal{Z}$;

ables.

Strengthening Procedure

To produce strengthened CG cuts, a strategy similar to the proposal of Letchford et al. (2016) is employed. The key idea is to take violated CG cuts and then strengthen the right-hand-sides (rhs). Letchford et al. (2016) solve the maximum weight stable set problem for the conflict graph induced by the binary variables of the CG cut to find a (hopefully smaller) new valid rhs. Our approach solves the same problem augmented by additional constraints involving these variables. Consider here that set $\mathcal{H}$ contains all variables of the cut to be strengthened. $A_{\mathcal{H}}$ is the matrix of coefficients of $\mathcal{H}$ including

non-renewable and renewable resource constraints, job allocation constraints and conflict constraints, with rhs values specified in a vector $\vec{b}$. The vector $\vec{c}$ contains the coefficients of the variables that appear in the cut. Consider vector $\vec{x}$ as variables of $\mathcal{H}$ and the integer linear programming as $\max\{\vec{c}^T\vec{x} : A_{\mathcal{H}}\vec{x} \leq \vec{b}\}$. If the optimal solution value of this MILP is smaller than the original rhs of the cut, the original CG can be strengthened with this new value on the rhs.

Example Considering the previous example. It is possible to strengthen the Chvátal-Gomory cut by tightening the value on the right side to 4 based on the MILP presented before. Notice that by applying the strengthening, due to renewable, non-renewable resource constraints and conflicts for these variables, it was possible to reduce the rhs value by 20%:

$$(SCG) = 3x_{1,0,8} + x_{5,0,5} + 2x_{5,0,6} + 4x_{6,1,4} + 2x_{6,1,8} + 2x_{8,1,7} + 2x_{9,1,7} \leq \mathbf{4}.$$

Chapter 6

Computational Results

This chapter presents the results of the experiments obtained by the proposed cutting plane algorithm and the preprocessing routine to strengthen resource-related constraints. All computational experiments for the MILP based approaches have been carried out on a computing cluster (Compute Canada) composed by Intel ® Xeon X5650 Westmere processors with 2,67 GHz and 512 GB of RAM running Scientific Linux release 6.3. All algorithms were coded in `ANSI C 99` and compiled with GCC version 5.4.0, with flags *-Ofast* and solver Gurobi version 8.0.1 (Gurobi Optimization, 2016). For the heuristic strategies experiments an Intel ®Core i7-4790 processor with 3.6 GHz and 16 GB of RAM running SUSE Leap Linux was used during the experiments. All algorithms were coded in ANSI C 99 and compiled with GCC 4.8.3 using flags -Ofast and -flto.

6.1 Heuristic Strategies Experiments

We evaluated the performance of *offline* and *online* tuning strategies by using the LAHC algorithm and the overall quality was evaluated over the instances from MMRCMPSP. After the approaches' evaluation, we also run the LAHC to determinate upper bounds to instances from PSPLIB and MMLIB datasets on the Compute Canada. At each iteration, a random neighborhood is chosen considering the probabilities $\mathbf{p}$ in a roulette scheme (Baker, 1987). *Offline* and *online* tuning strategies were considered over a single value for this parameter[1] so that both strategies receive the same tuning effort.

Each neighborhood composition strategy was evaluated on instances from MMR-

[1]LAHC list size value was fixed to 1,000

CMPSP datasets using five independent executions on each instance. The quality q_i of a solution obtained for an instance i is calculated as $q_i = \frac{b_i}{a_i}$, where b_i is the best known solution for i and a_i is the average solution costs. Therefore, the quality $\omega \in [0, 1]$ of solutions for an instance set I is computed as the average value of q_i for all $i \in I$. The standard deviation σ is also included.

Table 6.1 presents the results obtained with *offline* tuning. Best average results are shown in bold. The first column presents results with uniform probabilities $(1/k)$ for selecting all neighborhoods during the entire search. The second column shows the results when probabilities are defined using a single stage, and the third column presents results obtained with the two stages tuning approach.

Table 6.1: Quality of results for the *offline* tuning strategy

	Offline tuning		
	Uniform	Single stage	Two stages
$\omega_{\vartheta=0}$	0.935	0.949	**0.955**
$\sigma_{\vartheta=0}$	0.041	0.039	0.039
$\omega_{\vartheta=0.1}$	0.935	0.952	0.953
$\sigma_{\vartheta=0.1}$	0.041	0.038	0.037

As can be seen, the efficiency metric used to tune the probabilities of selecting each neighborhood produced good results: better average results were produced, in addition to a smaller standard deviation. The best results were obtained in the two stages approach. These results indicate that it would be probably beneficial to update the computational effort invested in each neighborhood in different phases of the search process. A natural extension of our proposal would be a more granular approach, i.e., the definition of more than two search phases to more properly adjust the probabilities of selecting each neighborhood as the search progresses.

After evaluating the *offline* approach, we evaluated the *online* approach. Table 6.2 shows the average quality and standard deviation obtained with the *online* approach using the metric $\tilde{e}$ and different values of z (interval in which probabilities are updated). The best results were obtained with $z = 1000$ and $\vartheta = 0$. *Online* tuning approaches can be beneficial to simplify the tuning process and to define the best neighborhood composition strategies during *runtime*. While this approach significantly improves the

uniform probabilities strategies, it performs slight worse (1% in our tests) than the best configuration obtained with *offline* tuning.

Table 6.2: Quality of results for *online* tuning $\beta = 0.01$

	Online tuning			
	z=1000	z=10000	z=50000	z=100000
$\omega_{\vartheta=0}$	**0.9468**	0.9428	0.9431	0.9462
$\sigma_{\vartheta=0}$	0.0438	0.0415	0.0444	0.0437
$\omega_{\vartheta=0.1}$	0.9339	0.9358	0.9309	0.9330
$\sigma_{\vartheta=0.1}$	0.0420	0.0422	0.0411	0.0428

Once we understand that the *online* approach is beneficial to the problems of the generalized MMRCMPSP variant, experiments were performed on Compute Canada for the entire benchmark datasets, including instances from PSPLIB and MMLIB. Figure 6.1 shows two box plots representing the improvements obtained by applying the *online* method to all benchmark datasets.

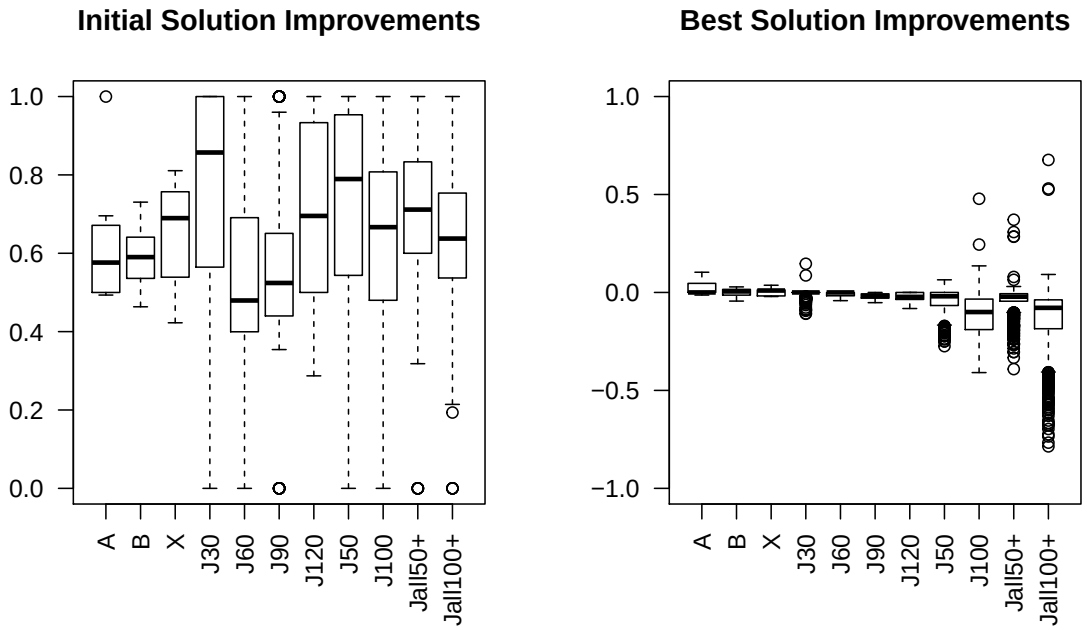

Figure 6.1: Solutions improvements results obtained by running the *online* tuning for all instances from the benchmark datasets

The values [0,1] from the first box plot, represents a factor of improvement comparing the initial solution, built using the binary program and greedy allocation, with the obtained solution. The values [-1,1] from the second box plot, represents a factor of improvement comparing the best-known solution from the literature with the obtained solution.

It can be observed that the online method can better, on average more than 40% the values of greedily designed initial solutions. The improvements obtained in groups J30 and J50 were more significant than other datasets. The online method was able to obtain similar values to the best-known solutions in the literature from PSPLIB and MISTA datasets. For the MMLIB datasets, values similar to those found in the literature were also obtained on average. However, we can observe some lower outliers below than 0, but also outliers representing an improvement in the literature solutions.

Figure 6.2 shows two box plot representing the improvements obtained by applying different intensities, including intensities obtained by offline and online methods, to all benchmark datasets.

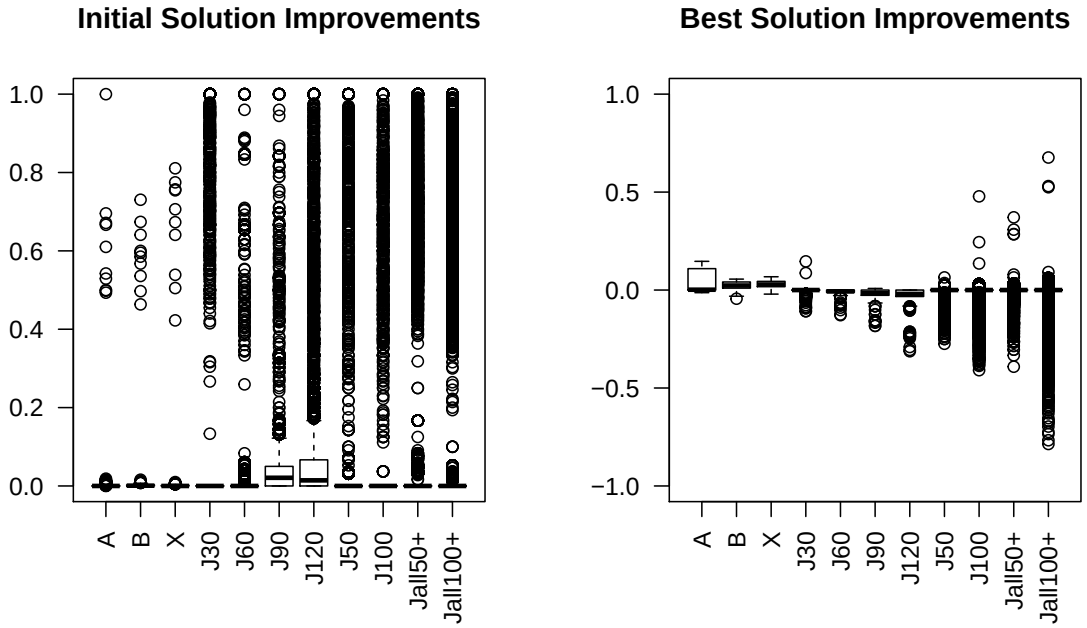

Figure 6.2: Solutions improvements results obtained by running different intensities for all instances from the benchmark datasets

The values [0,1] from the first box plot, represents a factor of improvement comparing

an improved initial solution with the obtained solution. The values [-1,1] from the second box plot, represents a factor of improvement comparing the best-known solution from the literature with the obtained solution. Looking at the similarity between the two box plots about the best solution improvements plotted in Figures 6.1 and 6.2, we can understand that the *online* method can obtain satisfactory results and learn to tune the neighborhoods properly. Besides, even though receiving a better initial solution, the first box plot has outliers with an improvement in these solutions.

Many new best-known solution was obtained, in addition to improved average solution costs for many instances considering Asta's results. Note that in Asta et al. (2014), the best-known results were obtained by running 2,500 independent executions in a computer cluster. All new best solutions were sent to the official website[2] of MISTA Challenge.

6.2 Preprocessing MILP Formulation Experiments

In order to evaluate the performance of the proposed cutting plane in relation to the specific problem cuts and the linear relaxation, preliminary experiments have been performed on the benchmark datasets with $\alpha > 0$ totaling 782 instances[3] from PSPLIB and MISTA. These two sets include instances of the versions SMRCPSP, MMRCPSP and MMRCMPSP. Instances from MMLIB, that contemplate a larger number of varied instances for the MMRCPSP version, were introduced only in experiments carried out using the complete version of our approach. The impact of adding the cuts for instances where $\alpha = 0$ could not be measured, since the LP bound remains at the same value even when the formulation has been improved. We used a time limit of 24 hours for each instance from the MMRCMPSP and 4 hours for each instance from the SMRCPSP and MMRCPSP.

The first experiment was conducted to evaluate the preprocessing MILP formulation by analyzing the LP relaxation (LR) and the strengthened LP relaxation (SLR) using the coefficient strengthening MILP presented in Section 5.2.1 to renewable resources constraints. The reported computing time for the SLR includes the time spent to find the successful feasible sets with parameters to stop the enumeration process as $it = 200,000$ and tl as the maximum allowed time defined above to run the approach. Table 6.3

[2] `https://gent.cs.kuleuven.be/mista2013challenge/`

[3] $\alpha > 0$ is an upper bound to the maximum TPD with value higher then its CPD.

shows the average integrality gaps [4] and the average computing times in seconds that have been obtained with the original LP relaxations and the strengthened LR for the instances from PSPLIB and MISTA.

Table 6.3: Average integrality gaps (1.0 = 100%) and the average computing times (sec.) for solving with the original LP relaxations (LR) and the strengthened LR (SLR)

		LR		SLR	
group	n	gap	time	gap	time
A	10	0.441	85.9	**0.438**	683.6
J30	245	0.663	0.3	**0.658**	2.8
J60	57	0.825	1.0	**0.816**	5.3
J90	80	0.822	2.1	**0.818**	26.9
J120	390	0.840	3.3	**0.835**	37.2
total	782	0.718	18.5	**0.713**	151.2

The results in Table 6.3, shown in bold numbers, indicate that the strengthening formulation slightly improves the integrality gaps. As expected, solving only the weak initial formulation is faster than solving the formulation with the preprocessing routines. However, we note that even small improvements in the root node can result in a large number of nodes pruned later in the search tree. For this reason, and given that the additional computing times are still quite reasonable, we use this strengthening strategy in all further experiments.

According to Kolisch (1995), network precedence relationships and the factor between availability-consumption of resources are the two critical characteristics of the instances. When pruning is applied considering these two characteristics, we note that even for instances with 120 jobs, there are time periods for which it is possible to enumerate the feasible subsets. For example, it is possible to enumerate until $t = 102$ for the large instance $j1201\_1$. This instance is composed of 120 jobs and has restricted relationship between the availability and consumption of renewable resources.

[4] given the best known upper bound $\overline{b}$, from PSPLIB and MISTA websites, and an obtained dual bound $\underline{b}$, the integrality gap is computed as follows: $\frac{(\overline{b}-\underline{b})}{\overline{b}}$, $\overline{b} > 0$

6.3 Cutting Plane Algorithm Experiments

In order to devise an effective cutting plane strategy, we next evaluate the different separation strategies.

6.3.1 Results for Different Cut Families

In Section 6.2, it has been shown that using the SLR instead of the original LR strengthens the formulation while only marginally increasing computing times. We now explore the bound improvement when combining the SLR with each of the different cut types. Tables 6.4 and 6.5 show the average integrality gaps and the average computing times for different approaches. In both tables, the first column SLR presents results obtained by the strengthened LP relaxation without cuts. The LCV[5], LPR, CL, OH ad SCG columns indicate, respectively, results[6] obtained by combining the SLR with one additional cut type: lift operations to cover and precedence cuts, clique cuts, odd-hole cuts and strengthened Chvátal-Gomory cuts.

Table 6.4: Average integrality gaps (1.0 = 100%) and average computing times (sec.) obtained by different cuts for the SMRCPSP and MMRCPSP benckmark datasets with the time limit of 4 hours

		SLR		+{LCV}		+{LPR}		+{CL}		+{OH}		+{SCG}	
group	n	gap	time	gap	time	gap	time	gap	time	gap	time	gap	time
J30	245	0.658	3	0.644	6	**0.549**	5	0.567	101	0.657	3	0.551	14200
J60	57	0.816	5	0.814	11	**0.712**	88	0.806	1093	0.816	12	0.812	14401
J90	80	0.818	27	0.816	45	**0.672**	454	0.807	3251	0.818	35	0.815	14400
J120	390	0.835	37	0.820	114	**0.714**	2111	0.817	7265	0.835	56	0.832	14423
total	772	0.781	18.1	0.774	43.9	**0.661**	665	0.749	2928	0.781	27	0.780	14408

The results in Table 6.4 suggest that the best average values (in bold numbers) of the integrality gaps were obtained by adding the lifted precedence cuts. Those cuts offered the best bound improvement (about 15%), while requiring only additional computing time. For the complete cutting plane, in the final experiment, we execute the separation procedures in parallel, but if a hierarchical implementation approach was used, one

[5] $\omega = 100000$ and $\mu = 0.1$

[6] the maximum number of precedence and clique cuts added to the LP at each iteration corresponds to 20% of the amount of the LP rows

would add the cut families based the order of their integrality gap improvements: LPR, CL, LCV, SCG and OH.

The results of instances from MMRCMPSP are presented in Table 6.5. The results summarized indicate that the LPR cuts are also effective for the multi-project problem variant, even for large instances. SCG cuts have been found to be particularly useful for this class of problem. Experiments for instances as from A-7 typically exceeded the given time or memory limits to insertion of cuts such as CL, OH, and SCG.

Table 6.5: Integrality gaps (1.0 = 100%) and computing times (sec.) obtained by different cuts to the MMRCMPSP benckmark datasets with the time limit of 24 hours

	SLR		+{LCV}		+{LPR}		+{CL}		+{OH}		+{SCG}	
inst.	gap	time	gap	time	gap	time	gap	time	gap	time	gap	time
A-1	1.000	0	1.000	0	0.875	0	0.875	0	1.000	0	**0.000**	1793
A-2	1.000	1	1.000	4	0.987	4	0.989	2	1.000	2	**0.935**	86007
A-3	**0.000**	8	**0.000**	9	**0.000**	15	**0.000**	15	**0.000**	15	**0.000**	62097
A-4	0.414	4	0.411	35	**0.333**	39	0.366	1297	0.414	10	0.396	86005
A-5	0.305	32	0.299	1743	**0.273**	1168	0.301	6157	0.305	254	0.302	86028
A-6	0.428	596	0.427	4118	**0.317**	3303	0.413	34106	0.428	1724	0.424	86005
total	0.5245	221	0.523	985	**0.464**	755	0.491	6930	0.525	334	**0.343**	40983

In Table 6.6 we evaluate the lifting strategies of the traditional RCPSP cuts proposed by Zhu et al. (2006) and the strengthening strategy for Chvátal-Gomory cuts, also combining with the SLR.

Table 6.6: Average integrality gaps (1.0 = 100%) and average computing times (sec.) obtained by lifting the traditional RCPSP cuts, by strengthening the CG cuts and their original versions combining with the SLR with time limit of 24 hours for A group and 4 hours for the others

		+{CV}		+{LCV}		+{PR}		+{LPR}		+{CG}		+{SCG}	
group	n	gap	time	gap	time	gap	time	gap	time	gap	time	gap	time
A	6	0.526	59.9	**0.523**	984.8	0.518	555.9	**0.464**	754.8	**0.342**	71825	0.343	67989
J30	245	0.660	0.7	**0.644**	5.6	0.653	3.7	**0.549**	4.9	**0.548**	14092	0.551	14200
J60	57	0.825	2.1	**0.814**	11.2	0.794	30.1	**0.712**	87.5	**0.811**	14402	0.812	14401
J90	80	0.821	4.3	**0.816**	44.9	0.780	131.8	**0.672**	454.3	**0.815**	14402	**0.815**	14400
J120	390	0.839	66.2	**0.820**	113.9	0.809	1516.2	**0.714**	2111.4	**0.832**	14402	**0.832**	14423
total	782	0.734	26.6	**0.723**	232.1	0.711	447.5	**0.622**	682.6	**0.670**	25825	0.671	25083

The lifting strategies were able to improve the average integrality gaps of all benchmark datasets when comparing with the traditional RCPSP cuts (CV and PR). The results in bold numbers suggest that the lifting strategy is generally successful, substantially improving the average integrality gaps while increasing computing times. Note that even for the benckmark datasets of SMRCPSP, improved lower bounds were achieved, since lifting tends to use more variables in the cut apart from those that contribute to the cover violation. The strengthened CG cuts did not improve upon the original CG cuts, which can be explained by the large computing times spent by the strengthening procedure. When analyzing the average number of iterations for both procedures, the CG without lift did 3118 iterations in the average, while SCG did just 2450 iterations. Even with a reduced number (21%) of iterations it was able to achieve basically the same results.

To analyze the cut generation for each separation strategy, we have computed the number of unique cuts for each strategy to the previous experiment. Figure 6.3 shows box plot of the number of cuts for different instance types and benchmarks.

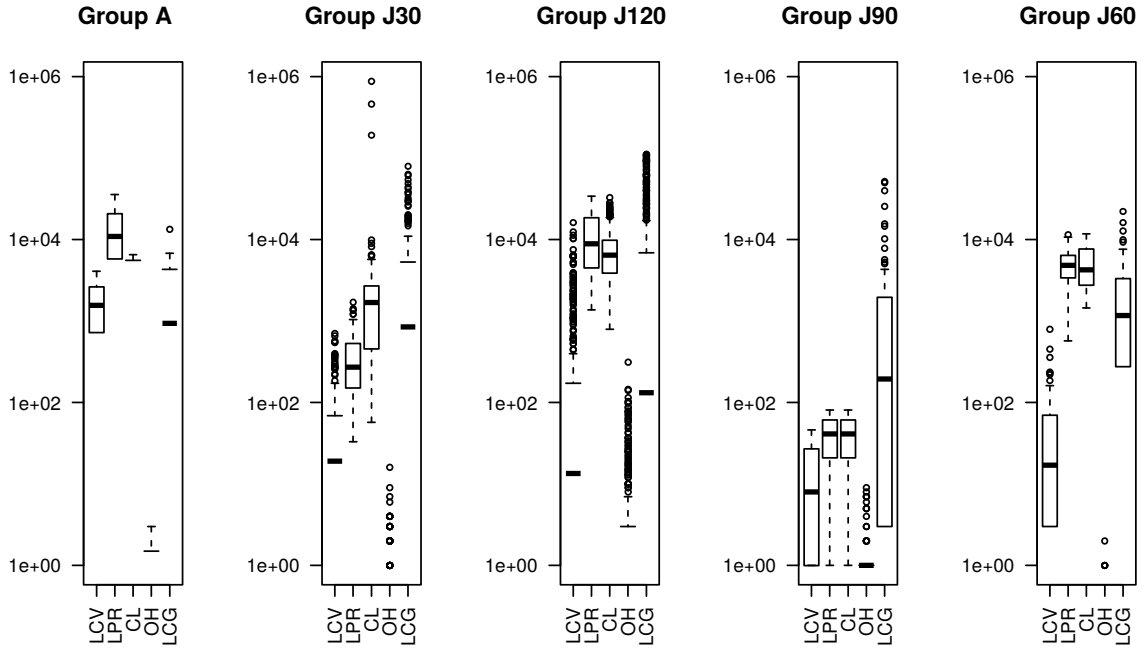

Figure 6.3: Box plot of the number of cuts for different types and benchmark datasets to each separation strategy

The lifted precedence strategy LPR finds a reasonable number of cuts for almost

all benchmark datasets. The strengthened Chvátal-Gomory SCG strategy seems to find cuts for all the benchmark datasets, including J30 and J90 in which LPR finds less. Another peculiarity of SCG is that although it finds few cuts at each iteration it remains to find cuts for a larger number of iterations. It is also possible to observe that a small number of odd-holes cuts are generated, which explains the little impact on the integrality gaps. Finally, the quantities of both cliques and lifted precedence cuts are relatively high. However, their separation requires more time.

6.3.2 Results Removing Cut Families

While the previous experiments explored the impact of adding each cut type, we now explore the impact of omitting each cut type. To this end, we use all cut types, separated in parallel in the cutting plane, and then individually remove each type for the benchmark datasets SMRCPSP and MMRCPSP. Table 6.7 summarizes the results of these experiments, where column pair All Cuts represents the strategy using all cut families, and the other column pairs represent the strategy without each of the cut types.

Table 6.7: Average integrality gaps (1.0 = 100%) and the average computing times (sec.) obtained by the cutting plane using all cut types and then removing one cut type at a time with time limit of 4 hours

		All Cuts		-{LCV}		-{LPR}		-{CL}		-{OH}		-{SCG}	
group	n	gap	time	gap	time	gap	time	gap	time	gap	time	gap	time
J30	245	**0.458**	13546	0.471	13631	0.485	13640	0.459	13645	0.461	13661	0.518	8
J60	57	**0.702**	14402	0.703	14401	0.798	14401	**0.702**	14401	**0.702**	14402	0.71	107
J90	80	0.666	14403	0.667	14403	0.8	14402	**0.665**	14403	0.666	14403	0.669	545
J120	390	0.668	14409	0.713	14436	0.774	14425	**0.667**	14429	0.67	14429	**0.667**	4779
Total	772	0.624	14190	0.639	14218	0.714	14217	**0.623**	14219	0.625	14224	0.641	1360

By removing the LPR cuts, results worsen by about 13%. The results also get a little worse by removing OH, LCV, and SCG respectively. The results using all cuts performed generally well. In addition, removing the clique cuts further improved the gaps on larger instances. This can be explained by the fact that separating cliques requires more time. When these cuts are removed the numbers of iterations increases, as one can observe in Table 6.8. Based on these results, one may define a new hierarchical sequence of the cut types based on the gap increase when the cut type is removed: LPR, SCG, LCV, OH, and CL.

Table 6.8: Average number of iterations and the average computing times (sec.) obtained by the cutting plane comparing with removing some others cuts with time limit of 4 hours

		All Cuts		-{LCV}		-{LPR}		-{CL}		-{OH}		-{SCG}	
group	n	round	time	round	time	round	time	round	time	round	time	round	time
J30	245	2077	13546	1946	13631	1796	13640	2486	13645	2035	13661	12	8
J60	57	618	14402	618	14401	877	14401	832	14401	452	14402	16	107
J90	80	294	14403	300	14403	476	14402	401	14403	293	14403	22	545
J120	390	265	14409	271	14436	347	14425	469	14429	284	14429	170	4779
Total	772	814	14190	784	14218	874	14217	1047	14219	766	14224	55	1360

Table 6.9 shows the results for the same experiments for the MMRCMPSP. Again, column All Cuts represent the strategy were all cut types are used. Instead of reporting the results when each of the other cut types is removed, we only report on removing the cut types that consumed most of the computing time: cliques and odd-holes cuts, or the strengthened CG cuts. The bold numbers show the best average integrality gaps achieved for instances that did not run out of memory.

Table 6.9: Average integrality gaps (1.0 = 100%) and the average computing times (sec.) obtained by the cutting plane comparing with removing some others cuts with time limit of 24 hours

	All Cuts		-{CL&OH}		-{SCG}	
instance	gap	time	gap	time	gap	time
A-1	**0.000**	52.5	0.875	0.7	0.875	0.4
A-2	**0.781**	86015.2	0.924	4.2	0.924	5.6
A-3	**0.000**	86368.5	**0.000**	11.0	**0.000**	9.4
A-4	**0.316**	86001.1	0.327	31.1	0.327	104.2
A-5	**0.263**	86010.4	0.267	832.2	0.266	1280.4
A-6	**0.313**	86089.2	0.314	2130.2	0.314	2660.6
Total	**0.279**	71756.2	0.451	501,6	0.451	676.8

Even though the results are better on average when using all cuts, by removing the cuts, the time was reduced significantly. Even though using all cut types significantly increases the computing times, the gap improvement on some of the instances may still justify their use.

To analyze the number of generated cuts for all separation procedures, we also com-

puted the number of non-repeated cuts to the previous experiments (All Cuts). Figure 6.4 shows the box plot representing the number of cuts for different types and benchmarks.

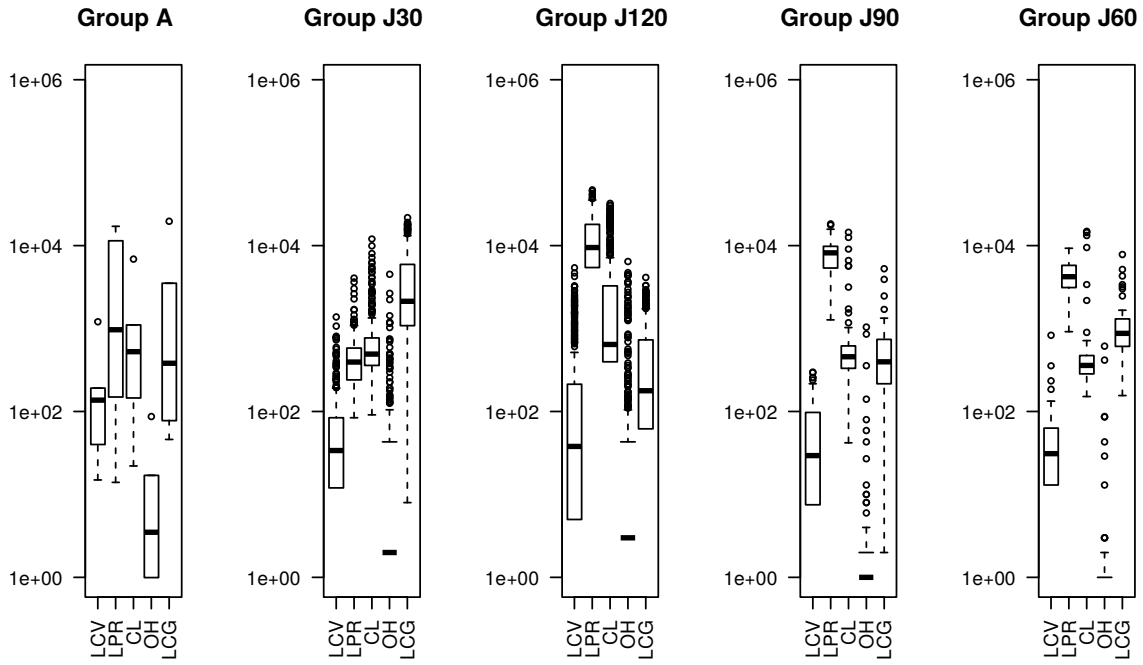

Figure 6.4: Box plot of the number of cuts for different types and benchmark datasets to all separation strategy together

Figure 6.4 shows that LPR cuts are effective for all datasets mainly in the case of the SMRCPSP, where it found, on average, more cuts than others. For the MMRCMPSP, the LPR cuts are still the most effective ones. CL and SCG cuts seem to be effective mainly for instances with multiple modes. The average number of generated LCV cuts seems to be stable for all benchmarck dataset. Few OH cuts are found for all benchmarck datasets.

After analyzing the best strategy for the MMRCPSP obtained through experiments on the PSBLIB instances, we run the experiment with all cuts for the instances from MMLIB, comparing the LP relaxation and the strengthened linear relaxation. The results are presented in Table 6.10. The results show that the cutting plane was very effective for all benchmark datasets, improving the average values by approximately 15%. As in earlier comparisons, the SLR slightly improves the integrality gaps for MMLIB instances too. However, 135 of the JA11 instances ran out of memory, which have been

removed from the results presented in the table to provide a fair comparison.

Table 6.10: Average integrality gaps (1.0 = 100%) and the average computing times (sec.) obtained by the cutting plane comparing with the LP relaxation and its strengthening with time limit of 4 hours

		LR			SLR			+{All Cuts}		
group	n	gap	std	time	gap	std	time	gap	std	time
J50	540	0.750	0.309	0.6	0.749	0.309	6.4	**0.654**	0.367	9669.4
J100	540	0.768	0.293	11.0	0.768	0.293	35.6	**0.688**	0.354	11536.6
JAll50+	1616	0.586	0.250	6.6	0.584	0.277	30.1	**0.463**	0.227	13746.8
Jall100+	1489	0.644	0.272	54.8	0.644	0.272	144.0	**0.530**	0.265	14221.9
Total	4185	0.687	0.281	18.2	0.686	0.288	54.0	**0.584**	0.303	12293.7

6.4 Branch & Cut Experiments

We now explore how the different cut types can be added dynamically in a Branch-and-Cut manner, to improve the solution process using the general purpose MILP solver Gurobi. Experiments were executed in order to compare the results achieved by solving the model with the inclusion of cuts into the root plus precedence cuts into a callback[7] procedure when the lower bound is improved and the results achieved by solving the model without cuts just with the preprocessing input data and Gurobi cuts.

Table 6.11 summarizes the integrality gaps[8] (average and standard deviation), for open instances with $\alpha > 0$ from PSPLIB and MISTA. The experiments for our approach have been limited to 24 hours of computing time. For Gurobi cuts, the first experiments have been limited to 24 hours and the second have been limited to 48 hours, in order to ensure a fair comparison since we not consider the time spent to insert all cuts into the root node for our approach. Initial solutions available at MISTA website were inserted only for the A dataset.

By analyzing Table 6.11 we can see that better results are obtained with the introduction of cuts into the root and with the LPR cut into the callback procedure. The average integrality gaps are reduced. Also, for some particular instances like $j12049\_8$,

[7]a callback is a user function that is called periodically by the Gurobi optimizer in order to allow the user to query or modify the state of the optimization Gurobi Optimization (2016)

[8]the integrality gaps are computed as in the previous experiments, since it was not possible to generate an incumbent solution for all instances, so it is not possible to obtain the optimality gap for some instances

Table 6.11: Average and standard deviation for the integrality gaps (1.0 = 100%) obtained by the B&C with all cuts at root plus LPR in the callback procedure

		Gurobi cuts				+{Our appr.}	
		gap (24h)		gap (48h)		gap (24h)	
group	n	avg	std	avg	std	avg	std
A	10	0.210	0.165	0.158	0.139	**0.139**	**0.124**
J30	245	0.007	0.022	0.006	0.022	**0.005**	**0.020**
J60	57	0.252	0.115	0.237	0.103	**0.184**	**0.084**
J90	80	0.268	0.162	0.253	0.139	**0.201**	**0.084**
J120	390	0.299	0.253	0.292	0.252	**0.249**	**0.224**

$j12021\_1$, $j12022\_8$, $j3037\_1$, $j3037\_7$, $j6046\_5$, $j6030\_2$, among others, the average optimality gap achieved within 24 hours of computing time was quite low (0.09%) when adding the cuts. Almost all open instances with $\alpha = 0$ from PSPLIB have been easily solved to optimality, except for some instances in set with 120 jobs.

We now explore how the two settings compare throughout the optimization process to find feasible solutions and to prove optimality. Table 6.12 summarizes the results for datasets from MISTA and PSPLIB for the three problem variants, also including those instances with $\alpha = 0$. Comparing to Toffolo et al. (2016), we solve to optimality, for the first time, 1 instance from `A` set. Comparing to Schnell and Hartl (2017) we solve the `J30` set, for the first time, 13 instances to optimality, and prove infeasibility for 88 of the instances.

Table 6.12: Solutions obtained by the B&C with all cuts at root plus LPR in the callback procedure

		Gurobi cuts			+{Our appr.}		
group	n	opt	fea	inf	opt	fea	inf
A	10	**4**	6	0	**4**	6	0
J30	640	530	6	**88**	**532**	**15**	**88**
J60	79	24	0	0	**26**	**2**	0
J90	105	**27**	0	0	**27**	0	0
J120	514	160	2	0	**181**	**3**	0

The results indicate that our approach has been able to prove optimality for more instances than using Gurobi without our cuts. For some specific instances the optimal value is found only when cuts are added to the model. In addition, Gurobi solver proves that some instances are infeasible.

Table 6.13 summarizes the results for our approach on the MMLIB datasets. The table shows the number of instances that have been solved to optimality (opt) and the number of instances for which a feasible solution has been found, but optimality has not been proven (fea). Further, the table shows the number of instances for which our solutions improve (imp) upon those reported by the website[9] and those reported in Schnell and Hartl (2017), as well the number of instances for which optimality has been proven for the first time (new opt).

Table 6.13: Solutions obtained by the B&C with all cuts at root plus LPR in the callback procedure

		all cuts at root + LPR callback				
group	n	opt	new opt	fea	imp	imp S&H
J50	540	450	29	32	0	48
J100	540	410	48	43	7	114
JAll50+	1620	689	252	159	59	401
JAll100+	1620	482	177	205	174	440

Figure 6.5 presents box plot of the optimality gaps[10] and computing times for our B&C approach for instances from PSPLIB, MISTA and MMLIB for which feasible or optimal solutions have been found.

The results presented in the first figure suggest that the gap values equals to 0 are represented by the median in the box plot for almost all datasets, except for the A and Jall100+ datasets. The high outliers for the datasets J120, J50, J100, Jall50+ and Jall100+ indicate that the presented algorithms still require improvement in order to deliver robust results on all instances. Analyzing the computing times it can be observed that our approach quickly proves optimality for all but dataset A. Finally, we also note that some of the instances between the second and third quartile hit the time limit.

Table 6.14 shows updated open instance values for the benchmark dataset defined in Chapter 2.

[9] http://mmlib.eu/solutions.php
[10] 1.0 = 100%

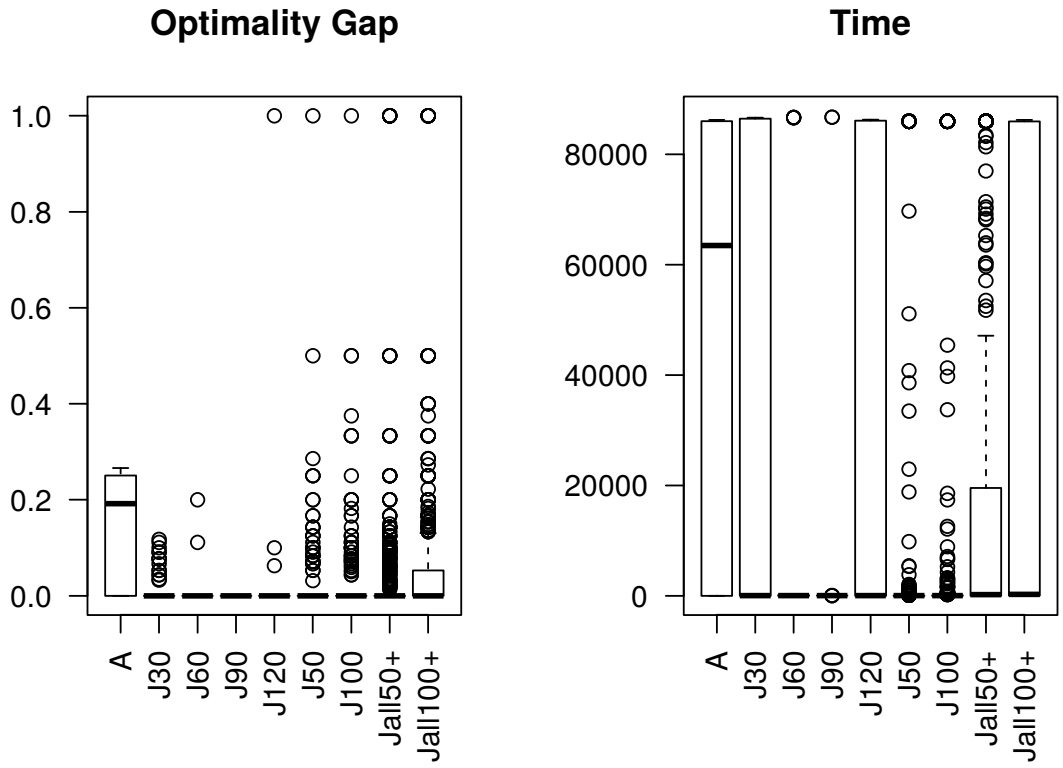

Figure 6.5: Box plot with informations about optimality gap and computing time (sec.) of instances from PSPLIB, MISTA and MMLIB

Table 6.14: Benchmark datasets updated numbers

library	variant/num. inst.	group	in dataset	open	updated open
PSPLIB	SMRCPSP/2040	J60	79	79	**53**
		J90	105	105	**78**
		J120	514	514	**333**
	MMRCPSP/3931	J30	640	31	**20**
MMLIB	MMRCPSP/4300	J50	540	118	**90**
		J100	540	176	**130**
		Jall50+	1620	1178	**931**
		Jall100+	1620	1370	**1138**
MISTA	MMRCMPSP/30	A	10	7	**6**
		B	10	10	10
		X	10	10	10

In summary, optimality was proven for the first time, for 247 instances from PSPLIB,

for 1 instance from MISTA and for 506 instances from MMLIB, totaling 754 instances. The LP and solution files for all benchmark datasets are available for download at `http://professor.ufop.br/janniele/downloads`.

Chapter 7

Final Considerations

The main contribution of this thesis was to improve the state-of-the-art in formulations and mixed-integer linear programming methods to solve the non-preemptive resource constrained project scheduling problems. We proposed strong formulations for the problem of interest, combining cut generation strategies for the management of the possibly exponential amount of rows of the developed models. A careful experimental design to evaluate the optimality gaps provided by the formulations and algorithms in a function of runtime for the benchmark datasets available in the literature was provided. All methods were extensively evaluated in three RCPSP variants.

An effective preprocessing procedure to strengthen renewable resource constraints was devised. This procedure was capable of improving the lower bounds produced at the root node without any increase in the size of the linear programs.

A parallel cutting plane algorithm was developed, including five families of cuts: lifted precedence and cover cuts, cliques, odd-holes and strengthened Chvátal-Gomory cuts. A dense conflict graph, considering feasibility and optimality conditions, was created at each iteration and used by these cut generators in strengthening procedures. All cuts contributed to improving the lower bounds, especially when they are together in the cutting plane. The lifted precedence cuts were the most effective for all variants. The strengthened Chvátal-Gomory cuts were especially useful in a group of multi-project instances. These results indicate that an instance feature-based tuning of the cut generators may be beneficial.

With the improved linear programming formulations produced with our methods, 754 open instances from the literature were solved for the first time: 247 instances from

PSPLIB, 1 instance from the MISTA Challenge and for 506 instances from MMLIB. The experiments confirm the hypothesis raised in this thesis. It is possible to handle the RCPSP formulations with better solvers and produce improved dual limits and solutions closer to the optimal gap.

For the production of upper bounds, a heuristic method, which produces tight upper bounds in limited computational time, was developed. When dealing with composition strategies to generate tight upper bounds, we evaluated different strategies to define the best neighborhood composition in stochastic local search methods. The first important observation was that the trivial implementation with uniform neighborhood selection probabilities performed worse than versions with more elaborated strategies to define these values. We provided some valuable insights: while some neighborhoods were quite useful at the beginning of the search, they may prove themselves ineffective when improved solutions were being processed; others remained active during the entire search. Many new best-known solution have been obtained.

7.1 Future Works

Regarding the problems explored in this thesis, some aspects may be further investigated:

- the exact proposed approaches could be hybridized with constraint propagation, as proposed in previous studies;
- the separation of some inequalities, such as the strengthened Chvátal-Gomory cuts and the re-optimization of the large linear programs, are still quite expensive; the development of new procedures to accelerate the separation and re-optimization process and to improve the creation of the dynamic conflict graph could be beneficial;
- the development of a B&C with improved branch and node selection strategies for the RCPSP, including our preprocessing and cutting planes routines, is also a promising future path since the root node resolution time is still substantial, this is an obstacle to the development of a B&C; continued advances in MILP solver are likely to allow the effective use of these formulations in methods where branching is used most intensively;

- the experiments could be extended with all combinations of cuts, and also the use of machine learning could help to select cuts during the cutting plane;
- understand what makes some neighborhoods special and try to create new neighborhoods inspired in a way that they preserve these good characteristics or discard some insignificant neighborhoods;
- apply additional intensification and diversification strategies based on machine learning techniques;
- resetting the neighborhood application probabilities could help the search when improvements are no longer obtained;
- finally, the presented neighborhood composition may be employed in other heuristics.

Bibliography

Aardal, K. and van Hoesel, C. P. M.: 1996, Polyhedral techniques in combinatorial optimization I: Theory, *Statistica Neerlandica* **50**(1), 3–26.

Aardal, K. and van Hoesel, C. P. M.: 1999, Polyhedral techniques in combinatorial optimization II: applications and computations, *Statistica Neerlandica* **53**(2), 131–177.

Alcaraz, J., Maroto, C. and Ruiz, R.: 2004, Improving the performance of genetic algorithms for the rcps problem, *Proceedings of the Ninth International Workshop on Project Management and Scheduling*, pp. 40–43.

Applegate, D. and Cook, W.: 1991, A computational study of the job-shop scheduling problem, *ORSA Journal on Computing* **3**, 149–1556.

Artigues, C.: 2017, On the strength of time-indexed formulations for the resource-constrained project scheduling problem, *Operations Research Letters* **45**, 154—159.

Artigues, C., Demassey, S. and Néon, E.: 2008, *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, ISTE Ltd and John Wiley & Sons, Inc.

Artigues, C., Demassey, S. and Néron, E.: 2013, *Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications*, ISTE, Wiley.

Asta, S., Karapetyan, D., Kheiri, A., Ozcan, E. and Parkes, A. J.: 2014, Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem, *Journal of Scheduling* **(in review**.

Atamtürk, A., Nemhauser, G. L. and Savelsbergh, M. W. P.: 2000, Conflict graphs in solving integer programming problems, *European Journal of Operational Research* **121**(1), 40–55.

Baker, J.: 1987, Reducing bias and inefficiency in the selection algorithm, *Proc. of the 2nd Intl Conf on GA*, Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA, pp. 14–21.

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

Balas, E. and Zemel, E.: 1978, Facets of the knapsack polytope from minimal covers, *SIAM Journal on Applied Mathematics* **34**(1), 119–148.

Baptiste, P. and Demassey, S.: 2004, Tight lp bounds for resource constrained project scheduling, *OR Spectrum* **26**(2), 251–262.

Bazaraa, M. S.: 2013, *Nonlinear Programming: Theory and Algorithms*, 3rd edn, Wiley Publishing.

Berthaut, F., Pellerin, R., Hajji, A. and Perrier, N.: 2018, A path relinking-based scatter search for the resource-constrained project scheduling problem, *International Journal of Project Organisation and Management* **10**(1), 1–36.

Blazewicz, J., Lenstra, J. and Rinnooy Kan, A.: 1983, Scheduling subject to resource constraints: classification and complexity, *Discrete Appl. Math* **5**, 11–24.

Boyd, E.: 1994, Solving 0/1 integer programs with enumeration cutting planes, *Annals of Operations Research* **50**, 61–72.

Boyd, E. A.: 1992, Fenchel cutting planes for integer programming, *Operations Research* **42**, 53–64.

Bradley, S. P., Hax, A. C. and Magnanti, T. L.: 1977, *Applied Mathematical Programming*, Addison-Wesley.

Bresina, J. L.: 1996, Heuristic-biased stochastic sampling, *Proceedings of the thirteenth national conference on Artificial intelligence - Volume 1*, AAAI'96, AAAI Press, pp. 271–278.

Brito, S. S., Santos, H. G. and Poggi, M.: 2015, A computational study of conflict graphs and aggressive cut separation in integer programming, *Electronic Notes in Discrete Mathematics* **50**, 355–360.

Brucker, P., Knust, S., Schoo, A. and Thiele, O.: 1998, A branch and bound algorithm for the resource-constrained project scheduling problem, *European Journal of Operational Research* **107**, 272–288.

Burke, E. K. and Bykov, Y.: 2017, The late acceptance hill-climbing heuristic, *European Journal of Operational Research* **258**(1), 70 – 78.

Burke, E. K. and Pinedo, M. L.: 2017, Journal of scheduling.
URL: *https://link.springer.com/journal/10951*

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P. and Schulenburg, S.: 2003, *Hyper-Heuristics: An Emerging Direction in Modern Search Technology*, Springer US, pp. 457–474.

Cavalcante, C. C. B., de Souza, C. C., Savelsbergh, M. W. P., Y, W. and Wolsey, L. A.: 2001, cheduling projects with labor constraints, *Discrete Applied Mathematics* **112**(1), 27–52.

Chakrabortty, R. K., Sarker, R. A. and Essam, D. L.: 2015, Resource constrained project scheduling: A branch and cut approach, *International Conference on Computers and Industrial Engineering*, Vol. 45, pp. 552–559.

Chen, W.-n. and Zhang, J.: 2012, Scheduling multi-mode projects under uncertainty to optimize cash flows: A monte carlo ant colony system approach, *Journal of Computer Science and Technology* **27**, 950–965.

Christofides, N., Alvarez-Valdes, R. and Tamarit, J. M.: 1987, Project scheduling with resource constraints: A branch and bound approach, *European Journal of Operational Research* **29**, 262–273.

Churchman, C. W., Ackoff, R. L. and Arnoff, E. L.: 1957, *Introduction to Operations Research*, John Wiley & Sons.

Chvátal, V.: 1973, Edmonds polytopes and a hierarchy of combinatorial problems, *Discrete Mathematics* **4**, 305—337.

Coelho, J. and Vanhoucke, M.: 2011, Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers, *European Journal of Operational Research* **213**(1), 73–82.

Cook, W.: 2019, *Computing in Combinatorial Optimization*, Springer International Publishing, pp. 27–47.

Cook, W. and Koch, T.: 2008, Mathematical programming computation: A new mps journal, *Optima* pp. 1,7–8,11.

da Fonseca, G. H. G., Santos, H. G., Toffolo, T. A. M., Brito, S. S. and Souza, M. J. F.: 2014, GOAL solver: a hybrid local search based solver for high school timetabling, *Annals of Operations Research* .

Dantzig, G.: 1998, *Linear Programming and Extensions*, Princeton University Press.

de Souza, C. C. and Wolsey, L. A.: 1997, Scheduling projects with labour constraints, *Technical report, Instituto de Computação, Universidade Estadual de Campinas* .

Demassey, S., Artigues, C. and Michelon, P.: 2005, Constraint propagation based cutting planes : an application to the resource-constrained project scheduling problem, *INFORMS Journal on Computing* **17**, 52–65.

Demeulemeester, E. L. and Herroelen, W. S.: 1992, Recent advances in branch-and-bound procedures for resource-constrained project scheduling problems, *Paper presented at the Summer School on Scheduling Theory and Its Applications*, pp. 1–32.

Demeulemeester, E. L. and Herroelen, W. S.: 2002, *Project Scheduling: A Research Handbook*, Kluwer Academic Publishers.

Fischetti, M. and Lodi, A.: 2007, Optimizing over the first Chvátal closure, *Mathematical Programming* **110**(1), 3–20.

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

Geiger, M. J.: 2013, Iterated variable neighborhood search for the resource constrained multi-mode multi-project scheduling problem. some comments on our contribution to the mista 2013 challenge, *Multidisciplinary International Scheduling Conference (MISTA) 2013 Proceedings*, Vol. 27-29, pp. 807–811.

Gu, Z., Nemhauser, G. L. and Savelsbergh, M. W. P.: 2000, Sequence independent lifting in mixed integer programming, *Journal of Combinatorial Optimization* **4**(1), 109–129.

Gurobi Optimization, I.: 2016, Gurobi optimizer: Reference manual.
URL: *http://www.gurobi.com/documentation/7.0/refman.pdf*

Habibi, F., Barzinpour, F. and Sadjadi, S. J.: 2018, Resource-constrained project scheduling problem: review of past and recent developments, *Journal of Project Management* **3**, 55–88.

Hansen, P. and Mladenović, N.: 2006, First vs. best improvement: An empirical study, *Discrete Applied Mathematics* **154**(5), 802–817.

Hardin, J. R., Nemhauser, G. L. and Savelsbergh, M. W.: 2008, Strong valid inequalities for the resource-constrained scheduling problem with uniform resource requirements, *Discrete Optimization* **5**, 19–35.

Hartmann, S.: 2002, A self-adapting genetic algorithm for project scheduling under resource constraints., *Naval Research Logistics* pp. 433–448.

Hoos, H. H. and Stützle, T.: 2005, *Stochastic Local Search: Foundations & Applications*, Morgan Kaufmann Publishers Inc.

ILO: 2008, *CPLEX 11.0 User's Manual*.

Johnson, E. L., Kostreva, M. M. and Suhl, U. H.: 1985, Solving 0-1 integer programming problems arising from large scale planning models, *Operations Research* **33**, 803–819.

Johnson, E. L., Nemhauser, G. and Savelsbergh, W. P.: 2000a, Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition, *INFORMS Journal on Computing* **12**.

Johnson, E., Nemhauser, G. and Savelsbergh, W.: 2000b, Progress in linear programming-based algorithms for integer programming: An exposition, *INFORMS Journal on Computing* **12**.

Jünger, M., Liebling, T. M., Naddef, D., Nemhauser, G. L., Pulleyblank, W. R., Reinelt, G., Rinaldi, G. and Wolsey, L. A.: 2010, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, Springer.

Jünger, M., Naddef, D., Pulleyblank, W. R., Rinaldi, G., Liebling, T. M., Nemhauser, G. L., Reinelt, G. and Wolsey, L. A.: 2010, *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*, Springer.

Kelley Jr, J. and Walker, M. R.: 1959, Critical-path planning and scheduling, *Eastern Joint IRE-AIEE-ACM Computer Conference*, ACM, pp. 160–173.

Kochetov, Y. A. and Stolyar, A. A.: 2003, Evolutionary local search with variable neighborhood for the resource constrained project scheduling problem, *Proceedings of the 3rd International Workshop of Computer Science and Information Technologies*.

Kolisch, R.: 1995, *Project Scheduling under Resource Constraints: Efficient Heuristics for Several Problem Classes*, number IX, 212 in *Production and Logistics*, 1 edn, Physica-Verlag Heidelberg.

Kolisch, R. and Sprecher, A.: 1996, PSPLIB - a project scheduling problem library, *European Journal of Operational Research* **96**, 205–216.

Kolisch, R., Sprecher, A. and Drexl, A.: 1995, Characterization and generation of a general class of resource-constrained project scheduling problems, *Management Science* **41**(10), 1693–1703.

Kone, O., Artigues, C., Lopez, P. and Mongeau, M.: 2011, Event-based MILP models for resource-constrained project scheduling problems, *Computers and Operations Research* **38**, 3–13.

Koster, A. M. C. A., Orlowski, S., Raack, C., Baier, G. and Engel, T.: 2008, *Single-layer Cuts for Multi-layer Network Design Problems*, Vol. 44, Springer-Verlag, chapter 1, pp. 1–23. Selected proceedings of the 9th INFORMS Telecommunications Conference.

Koulinas, G., Kotsikas, L. and Anagnostopoulos, K.: 2014, A particle swarm optimization based hyper-heuristic algorithm for the classic resource constrained project scheduling problem, *Information Sciences* .

Land, A. H. and Doig, A. G.: 2010, *An Automatic Method for Solving Discrete Programming Problems*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 105–132.

Letchford, A. N., Marzi, F., Rossi, F. and Smriglio, S.: 2016, Strengthening Chvátal-Gomory Cuts for the Stable Set Problem, *Combinatorial Optimization*, Springer International Publishing, pp. 201–212.

Linderoth, J. T. and Ralphs, T. K.: 2005, Noncommercial software for mixed-integer linear programming, *in* J. Karlof (ed.), *Integer Programming: Theory and Practice*, Vol. 3 of *Operations Research Series*, CRC Press.

Liu, M., Shan, M. and Wu, J.: 2014, Multiple R&D projects scheduling optimization with improved particle swarm algorithm, *The Scientific World Journal* **2014**.

Liu, S.-S. and Wang, C.-J.: 2008, Resource-constrained construction project scheduling model for profit maximization considering cash flow, *Automation in Construction* **17**(8), 966–974.

Lougee-Heimer, R.: 2003, The common optimization interface for operations research: Promoting open-source software in the operations research community, *IBM Journal of Research and Development* **47**, 57–66.

Luenberger, D. G. and Ye, Y.: 2016, *Linear and Nonlinear Programming*, 4 edn, Springer International Publishing.

Maniezzo, V., Stutzle, T. and Vo, S.: 2010, *Matheuristics: Hybridizing Metaheuristics and Mathematical Programming*, Vol. 10, Springer.

Metropolis, N.; Ulam, S.: 1949, The monte carlo method, *Journal of the American Statistical Association* **44**, 335–341.

Mingozzi, A., Maniezzo, V., Ricciardelli, S. and Bianco, L.: 1998, An Exact Algorithm for the Resource-Constrained Project Scheduling Problem Based on a New Mathematical Formulation, *Management Science* **44**(5), 714–729.

Mladenović, N. and Hansen, P.: 1997, Variable neighborhood search, *Comput. Oper. Res.* **24**(11), 1097–1100.

Möhring, R. H., Schulz, A. S., Stork, F. and Uetz, M.: 2003, Solving project scheduling problems by minimum cut computations, *Management Science* **49**(3), 330–350.

Muritiba, A. E. F., Rodrigues, C. D. and da Costa, F. A.: 2018, A Path-Relinking algorithm for the multi-mode resource-constrained project scheduling problem, *Computers & Operations Research* **92**, 145–154.

Nemhauser, G. L. and Vance, P. H.: 1994, Lifted cover facets of the 0–1 knapsack polytope with GUB constraints, *Operations Research Letters* **16**(5), 255–263.

Nonobe, K. and Ibaraki, T.: 2002, *Formulation and Tabu Search Algorithm for the Resource Constrained Project Scheduling Problem*, Springer US, pp. 557–588.

Ochoa, G., Verel, S., Daolio, F. and Tomassini, M.: 2014, Local optima networks: A new model of combinatorial fitness landscapes, *Recent Advances in the Theory and Application of Fitness Landscapes* **6**, 233–262.

Ong, Y.-S., Lim, M. H. and Chen, X.: 2010, Research frontier: memetic computation-past, present & future, *Comp. Intell. Mag.* **5**(2), 24–31.

Padberg, M. W.: 1973, On the facial structure of set packing polyhedra, *Mathematical Programming* **5**(1), 199–215.

Patterson, J. and Huber, W.: 1974, A horizon-varying zero-one approach to project scheduling, *Management Science* **20**, 990–998.

Pochet, Y. and Wolsey, L. A.: 2006, *Production Planning by Mixed Integer Programming*, Springer.

Pritsker, A. A. B., Watters, L. J. and Wolfe, P. M.: 1969, Multi project scheduling with limited resources: A zero-one programming approach, *Management Science* **3416**, 93–108.

Rahmania, N., Zeighami, V. and Akbari, R.: 2015, A study on the performance of differential search algorithm for single mode resource constrained project scheduling problem.

Riedler, M., Jatschka, T., Maschler, J. and Raidl, G. R.: 2017, An iterative time-bucket refinement algorithm for a high resolution resource-constrained project scheduling problem, *International Transactions in Operational Research* pp. 1–41.

Sankaran, J. K., Bricker, D. L., and Juang, S.: 1999, A strong fractional cutting plane algorithm for resource-constrained project scheduling, *International Journal of Industrial Engineering - Applications and Practice* **6**, 99–111.

Santos, H. G., Toffolo, T. A. M., Gomes, R. A. M. and Ribas, S.: 2016, Integer programming techniques for the nurse rostering problem, *Annals of Operations Research* **239**(1), 225–251.

Schnell, A. and Hartl, R. F.: 2017, 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.

Schwindt, C. and Zimmermann, J.: 2015, *Handbook on Project Management and Scheduling*, Vol. 1 of *International Handbooks on Information Systems*, Springer International Publishing.

Shapiro, J. F.: 1993, *Mathematical programming models and methods for production planning and scheduling*, Elsevier Science Publishers B.V.

Skowroński, M. E., Myszkowski, P. B., Adamski, M. and Kwiatek, P.: 2013, Tabu search approach for multi-skill resource-constrained project scheduling problem, *2013 Federated Conference on Computer Science and Information Systems*, pp. 153–158.

Soares, J. A., Santos, H., Baltar, D. D. and Toffolo, T. A. M.: 2015, Lahc applied to the multi-mode resource constrained multi-project scheduling problem, *7th Multidisciplinary International Conference on Scheduling: Theory and Applications (MISTA 2015)*.

Toffolo, T. A. M., Santos, H. G., Carvalho, M. A. M. and Soares, J. A.: 2013, An integer programming approach for the multi-mode resource-constrained multi-project scheduling problem, *Multidisciplinary International Scheduling Conference (MISTA) 2013 Proceedings*, Vol. 27-29, pp. 840–847.

Toffolo, T. A. M., Santos, H. G., Carvalho, M. A. M. and Soares, J. A.: 2016, An integer programming approach to the multimode resource-constrained multiproject scheduling problem, *Journal of Scheduling* **19**, 295–307.

Van Peteghem, V. and Vanhoucke, M.: 2014, An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances, *European Journal of Operational Research* **235**(1), 62–72.

Vanhoucke, M. and Coelho, J.: 2016, An approach using SAT solvers for the RCPSP with logical constraints, *European Journal of Operational Research* **249**(2), 577–591.

Wagner, H. M.: 1969, *Principles of operations research, with applications to managerial decisions*, Prentice-Hall.

Walędzik, K. and Mańdziuk, J.: 2017, Applying hybrid monte carlo tree search methods to risk-aware project scheduling problem, *Information Sciences* .

Wauters, T., Kinable, J., Smet, P., Vancroonenburg, W., Vanden Berghe, G. and Verstichel, J.: 2016, The multi-mode resource-constrained multi-project scheduling problem, *Journal of Scheduling* **19**, 271–283.

Wolsey, L.: 1998a, *Integer Programming*, Wiley-Interscience series in discrete mathematics and optimization, Wiley.

Wolsey, L.: 1998b, *Integer Programming*, Interscience series in discrete mathematics and optimization, Wiley.

Xu, J., Xu, Y., Kim, D. and Li, M.: 2003, Raptor: Optimal protein threading by linear programming, *Journal of Bioinformatics and Computational Biology* **1**, 95–117.

Zhu, G., Bard, J. F. and Yu, G.: 2006, A branch-and-cut procedure for the multimode resource-constrained project-scheduling problem, *INFORMS Journal on Computing* **18**(3), 377–390.