UNIVERSITÉ DU LUXEMBOURG

PhD-FSTM-2020-34
The Faculty of Science, Technology and Medicine

DISSERTATION

Presented on 08/07/2020 in Luxembourg
to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Willian TESSARO LUNARDI
Born on 13 February 1990 in Marau, Rio Grande do Sul (Brazil)

A REAL-WORLD FLEXIBLE JOB SHOP SCHEDULING PROBLEM WITH SEQUENCING FLEXIBILITY: MATHEMATICAL PROGRAMMING, CONSTRAINT PROGRAMMING, AND METAHEURISTICS

University of Luxembourg

Doctoral Thesis

A Real-World Flexible Job Shop Scheduling Problem With Sequencing Flexibility: Mathematical Programming, Constraint Programming, and Metaheuristics

Author:
Willian Tessaro Lunardi

Supervisors:
Dr. Holger Voos
Dr. Yves Le Traon
Dr. Nicolas Navet

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in Computer Science

in the

Automation & Robotics Research Group
Interdisciplinary Centre for Security, Reliability and Trust

June 8, 2020

Declaration of Authorship

I, Willian Tessaro Lunardi, declare that this thesis titled, "A Real-World Flexible Job Shop Scheduling Problem With Sequencing Flexibility: Mathematical Programming, Constraint Programming, and Metaheuristics" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

"*If I have seen further it is by standing on the shoulders of Giants.*"

Isaac Newton

Abstract

In this work, the online printing shop scheduling problem is considered. This challenging real scheduling problem, that emerged in the nowadays printing industry, corresponds to a flexible job shop scheduling problem with sequencing flexibility that includes several complicating specificities such as resumable operations, periods of unavailability of the machines, sequence-dependent setup times, partial overlapping between operations with precedence constraints, fixed operations, among others. In the present work, a mixed integer linear programming model, a constraint programming model, and heuristic methods such as local search and metaheuristics for the minimization of the makespan are presented. Modeling the problem is twofold. On the one hand, the problem is precisely defined. On the other hand, the capabilities and limitations of a commercial software for solving the models are analyzed. Numerical experiments show that the commercial solver is able to optimally solve only a fraction of the small-sized instances when considering the mixed integer linear programming formulation. While considering the constraint programming formulation of the problem, medium-sized instances are optimally solved, and feasible solutions for large-sized instances of the problem are found. Ad-hoc heuristic methods, such as local search and metaheuristic approaches that fully exploit the structure of the problem, are proposed and evaluated. Based on a common representation scheme and neighborhood function, trajectory and populational metaheuristics are considered. Extensive numerical experiments with large-sized instances show that the proposed metaheuristic methods are suitable for solving practical instances of the problem; and that they outperform the half-heuristic-half-exact off-the-shelf constraint programming solver. Numerical experiments with classical instances of the flexible job shop scheduling problem show that the introduced methods are also competitive when applied to this particular case.

Acknowledgements

First of all, I would like to start by expressing my gratitude towards my academic supervisor Prof. Dr.-Ing. Holger Voos and committee members, Prof. Dr. Nicolas Navet, and Prof. Dr. Yves Le Traon, for their guidance, support, and the knowledge shared with me throughout the years of my Ph.D.

I would like to thank my reviewers Prof. Dr. Ernesto G. Birgin and Dr. Philippe Laborie, for their insightful reviews and valuable feedback, which helped me improve my work.

The Grand Duchy of Luxembourg is a fascinating country with many to discover. I owe enormous thanks to the marvelous group of friends I have found here: Dr. Eric Rohmer, Flavio San Martin, Priscila Custódio, and Dr. S. Amin Sajadi-Alamdari. Also, an especial thanks to all the team members of the SnT Automation & Robotic Research Group for the fantastic working atmosphere.

Some special words of gratitude go to my friends who have always been a major source of support throughout my life: Dr. Everton de Matos, Dr. Henrique P. Martins, Dr. Ramão T. Tiburski, and Tadeu Catani. Thanks guys for always being there for me.

I sincerely thank my parents, Milton and Márcia, whose support and teachings throughout my life made me the person I am today. A special thanks to my sister Stefani for her love and affection.

Finally, my wonderful wife, Luciana, thank you for your support throughout this entire process and for your countless sacrifices to help me get to this point. Thank you for everything you are and everything you help me to be when I am with you.

Contents

List of Figures

List of Tables

Abbreviations

ABC	Artificial Bee Colony ... 100
AGV	Automated Guided Vehicles ... 1
BB	Branch and Bound ... 33
BS	Beam Search ... 131
CBFS	Cost-based Breadth-First Search ... 101
CPO	Constraint Programming Optimizer ... 104
CP	Constraint Programming ... 4
CS	Cuckoo Search ... 131
DAG	Directed Acyclic Graph ... 16
DE	Differential Evolution ... 8
EA	Evolutionary Algorithm ... 33
FA	Firefly Algorithm ... 100
FFS	Flexible Flow Shop ... 11
FJS	Flexible Job Shop ... 1
FMS	Flexible Manufacturing System ... 1
FS	Flow Shop ... 11
GA	Genetic Algorithm ... 8
GWO	Grey Wolf Optimizer ... 100
ICA	Imperialist Competitive Algorithm ... 100
ILS	Iterated Local Search ... 8
IP	Integer Programming ... 34
JS	Job Shop ... 3
LB	Lower Bound ... 90
LP	Linear Programming ... 34
LS	Local Search ... 4
MILP	Mixed Integer Linear Programming ... 4

MIP Mixed Integer Programming 34

MP Mathematical Programming 4

NLP Nonlinear Programming 34

OPS Online Printing Shop 1

OS Open Shop 11

PM Parallel Machine 11

PSD Pooled Standard Deviation 114

PSO Particle Swarm Optimization 100

RE Relative Error 131

RR Random Restart 41

SA Simulated Annealing 41

SI Swarm Intelligence 44

SM Single Machine 11

SPT Shortest Processing Time First 37

TS Tabu Search 8

UB Upper Bound 98

VNS Variable Neighborhood Search 41

WOA Whale Optimization Algorithm 100

Chapter 1

Introduction

This thesis addresses a real scheduling problem observed in the printing industry, the Online Printing Shop (OPS), which can be modeled as a generalization of the Flexible Job Shop (FJS). The purpose of this dissertation is to present and formulate the OPS scheduling problem, and to investigate optimization techniques to produce a production schedule so that the production effectiveness of OPSs can be improved.

This chapter is structure as follows. Section 1.1 presents the research context. Section 1.2 presents the research challenges and motivations to conduct this study. Section 1.3 presents the aim and scope of this study. Section 1.4 presents the scientific and practical contributions of this study. Finally, the outline of the thesis is presented in section 1.6.

1.1 Context of the Study

In 1926, Ford pointed out that *"Any customer can have a car painted any color that he wants so long as it is black"* (Ford and Crowther, 1926). The industry, at its second revolution stage, also known as the Industry 2.0, was not able to support such levels of customization in which Ford's customers were seeking due to its lack of flexibility. Nonetheless, in the late 1980s, during the third industrial revolution, also referred to as Industry 3.0, the customer demand for a large variety of products led to the development of "mass customization" (Pine, 1993). It was based on the development of information, automation technology, and the computer. This led to advanced equipment like computer numerically controlled machines, Automated Guided Vehicles (AGV), and material handling systems like robots that were integrated with computers and assimilated within a manufacturing system, forming what is called Flexible Manufacturing System (FMS). Consequently, flexible production was developed, and systems featured high productivity with minimal machines' setups, low cost, and large varieties. To this day, FMSs play an essential role in the

industry being considered one of the core dimensions of Industry 4.0 (Wang et al., 2017).

An example of a highly flexible, lean, and agile manufacturing system capable of producing a variety of different products in the same facility is shown in the printing industry. Printing shops are FMSs that rely on print media, providing a wide range of customized and personalized products such as newspapers, catalogs, magazines, books, and packaging. These companies offer greater flexibility in the processing of a large variety of substrates, inks, and methods of print finishing. Nowadays, due to challenges such as the decreasing of the demand due to the expanding use of online advertising and the decreasing of the overall printed matter, a new business model emerged, the so-called OPSs. OPSs commenced taking advantage of the Internet and standardization towards mass customization to attract more clients and to reduce costs. This new business model enables printers to leave the inquiry and order process to the customer. Alternatively, to a small number of fixed contracts, OPSs can reach up to several thousand online orders of diverse clients per day. Orders that can be of several types, such as bags, beer mats, books, brochures, business cards, calendars, certificates, Christmas cards, envelopes, flyers, folded leaflets, greeting cards, multimedia, napkins, paper cups, photo books, posters, ring binders, stamps, stickers, tags, tickets, wristbands, to name a few are placed through an online system. The system guides the user through a standardized process that defines details including size, orientation, material, finish, color mode, design, among others.

FMSs, such as the ones found in the OPSs, usually exhibit complicated process plans and a high degree of resource sharing. Consequently, allocating the enormous number of tasks to the resources and scheduling them while respecting several constraints of different jobs' processing plans, and machines availability is a challenging task. Depending on the problem size, represented by the number of tasks and resources, the scheduling task may turn out to be the most time consuming and challenging activity within an enterprise. Thus, the performance of an OPS not supported adequately by efficient scheduling may be significantly limited, and the advantages derived from its flexibility may suffer a sharp decrease, e.g., inefficient scheduling may result in reduced utilization of resources, over-loaded/idle capacity, long production lead time and unreliable due date commitments, which in turn increases production costs and reduces competitiveness in the market place. Furthermore, ineffective scheduling often delays orders and results in unsatisfied customers and may subject the firm to penalties.

1.2 Problem Statement and Motivation

Production scheduling is usually considered one of the most critical issues in the planning and operation of manufacturing systems (Pinedo, 2012). Scheduling involves allocating resources, such as machines, transporters, and personnel, to individual production activities in a time sequence to produce products or parts. Most of the scheduling problems presented in FMSs have to take into account these problem formulations in a quite general framework and hence are NP-hard (Blackstone, Phillips, and Hogg, 1982; Blazewicz et al., 2019). The overall system goals are to maximize resource utilization and minimize production time while respecting a variety of system constraints, such as the precedence between the production activities.

The Job Shop Job Shop (JS) scheduling problem consists of scheduling jobs in an environment with non-flexible machines. Each job is formed of several operations with a linear precedence order where each operation can be processed by only one particular machine. The JS scheduling problem is known to be strongly NP-hard (Garey, Johnson, and Sethi, 1976). The FJS scheduling problem can be seen as a generalization of the JS scheduling problem in which there may be several machines, not necessarily identical, capable of processing each operation. The processing time of each operation on each machine is known, and no preemption is allowed. The problem is to decide on which machine each operation will be processed, and in what order the operations will be processed on each machine.

The OPS scheduling problem can be formulated as a generalization of the FJS scheduling problem with many specificities that capture the critical features of the OPS' manufacturing system. The non-trivial constraints include sequencing flexibility where a directed acyclic graph gives precedence between operations of jobs; partial overlapping, where operations connected by precedence constraints can overlap; machines availability, where machines may present unavailable periods; and sequence-dependent setup times, where machines must be configured in order to process distinct operations. Since the OPS scheduling problem is at least as difficult as the FJS scheduling problem, it is also NP-hard. Note that the OPS' manufacturing system and the OPS scheduling problem are presented in Section 2.2.

The motivation of this study is to enhance the production effectiveness of OPSs by planning a production schedule so that they can gain as much production benefit as possible. The production effectiveness can be represented by an objective function under the constraint of the manufacturing system. The goal is to minimize the makespan (i.e., the time that elapses from the start of work to the end), as in this way, OPSs can reduce the cost of labor and electricity while achieving an increment of production capacity and products' quality.

1.3 Aim and Scope

The objective of this thesis is three-fold. The first objective is the introduction and the definition of the OPS scheduling problem into a realistic yet tractable model. The second objective is the formulation of the OPS scheduling problem using Mixed Integer Linear Programming (MILP) and Constraint Programming (CP). The universality of the MILP model provides in-depth knowledge that leverages the development of ad-hoc heuristic methods to be applied in practice. Formulating the problem with a modern CP-based scheduling language allows the usage of a solver dedicated to scheduling problems. The capabilities and limitations of each commercial software for solving the models must be investigated, and the practical applicability of the CP solver must be examined. Since comparing the CP solver performance for the OPS scheduling problem with other methods in the literature is not possible, the third goal is to propose and develop heuristic methods that fully exploits the problem structure.

In order to reach the objectives, this study addresses the following research questions:

1. Through collaboration with an industrial partner, how to define and formulate a realistic yet tractable model of the OPS manufacturing system?
2. How to solve the problem given the fact that the model presents several complex constraints and that size of instances in practice is substantial?
3. How to precisely formulate the OPS scheduling problem by means of Mathematical Programming (MP) and CP?
4. What is the effectiveness and limitations of the MP and CP formulations with the commercial solvers IBM ILOG CPLEX and IBM ILOG CP Optimizer?
5. What is the size of the instances for which optimality can be proved, and how good is the incumbent solution when it cannot?
6. How to encode and decode OPS scheduling solutions?
7. How to construct feasible OPS scheduling solutions?
8. How to define a set of neighbor solutions of an OPS scheduling solution to allow the applicability of Local Search (LS) heuristics or neighborhood-based methods?
9. Which metaheuristic frameworks are the most suitable for the OPS scheduling problem?
10. How the best performing method behaves when applied to particularizations of the OPS scheduling problem?

A three-step is taken to answer the research questions. The first step is to understand the OPS manufacturing system, identifying the critical constraints that must be taken into account in the formulation of the OPS scheduling problem. (Note that several meetings with an industrial partner conceded a sharp perception of the current printing industry's scenario, allowing the identification of the critical features of the OPSs' manufacturing system. The problem was identified as a generalization of the FJS scheduling problem with sequencing flexibility.) Works in the literature must be reviewed to identify the state-of-the-art and knowledge gap on the FJS scheduling problem with sequencing flexibility. This answers research questions 1 (Chapter 2). A review of optimization techniques for the FJS scheduling problem answers research question 2 (Chapter 3).

In the second step, exact scheduling optimization techniques are addressed. The OPS scheduling problem is precisely formulated as MILP and CP. To access the performance of the solvers, an OPS scheduling problem instance generator is developed, and several small, medium, and large-sized benchmark sets of instances are generated. Numerical experiments are performed with a twofold goal. On the one hand, the correctness of the proposed models is assessed. On the other hand, the effectiveness and limitations of the MILP and CP models are determined, i.e., the size of the instances for which optimality can be proved with the commercial solvers IBM ILOG CPLEX and IBM ILOG CP Optimizer is determined. This step answers research questions 3-5 (Chapter 4). In order to answer the "second part" of the research question 5, ad-hoc heuristic methods are required to compare their results with the incumbent solution obtained with the exact methods.

In the third step, near-optimal scheduling optimization methods such as LS heuristics and metaheuristics are addressed for the OPS scheduling problem. A real-valued representation scheme that encodes both the machine assignment for each operation and the execution order sequence of each operation is proposed. Given the representation scheme, a constructive heuristic is proposed to build a feasible solution that allows assessing the value of its makespan. The movement, neighborhood function, and LS procedure are introduced. Four metaheuristics are proposed and evaluated in connection with the representation scheme and the LS procedure. Computational experiments are performed to validate two different goals: i) investigate the performance of the proposed metaheuristics; ii) by means if real-world instance sizes, compare the performance of the metaheuristics with the commercial software IBM ILOG CP Optimizer. This step answers the research questions 5-9 (Chapter 5). The effectiveness of the method that presented the best results is analyzed with respect to other seven state-of-the-art methods by performing experiments with classical FJS scheduling problem benchmarks. This step answers research question 10 (Section 5.4.4).

1.4 Significance of the Study

The main contributions of this dissertation can be summarized in a similar fashion. These contributions include:

1. The study and modeling of a unique real-world scheduling problem.
2. The first attempt at solving the FJS scheduling problem with sequencing flexibility, operation's release time, operation overlapping, sequence-dependent setup times, fixed operations, and machines' unavailability periods.
3. The formulation of MILP and CP models which allows solving the OPS problem with a certificate of optimality for small and medium instances.
4. The development of a novel encoding scheme for the OPS scheduling problem, the FJS scheduling problem with sequencing flexibility, and its particularizations.
5. The development of a decoding procedure which, by means of a constructive heuristic, computes a feasible OPS scheduling problem solution allowing the assessment of its makespan.
6. The development of a moving operator, a neighborhood function, and a problem-specific LS heuristic method which allows the improvement of an OPS scheduling problem solution by the exploitation of neighboring solutions.
7. The development and adaptation of metaheuristics frameworks for solving the OPS scheduling problem, allowing to effectively produce good near-optimal scheduling solutions for small-, medium-, and large-sized instances.
8. The proposal of a metaheuristic method that exhibits competitive results when compared with other state-of-the-art methods for the FJS scheduling problem.

1.5 List of Publications

During the advancement of this research work and achieving the set of previously stated research questions and objectives, results were presented in the form of papers that have been published or submitted to be published.

1. Lunardi et al. (submitted). Metaheuristics for the Online Printing Shop Scheduling Problem. *European Journal of Operational Research.*
2. Lunardi et al. (to appear). Mixed Integer Linear Programming and Constraint Programming Models for the Online Printing Shop Scheduling Problem. *Journal of Computers & Operations Research.*

3. Lunardi, Voos, and Cherri (2019). An Effective Hybrid Imperialist Competitive Algorithm and Tabu Search for an Extended Flexible Job Shop Scheduling Problem. *ACM Symposium on Applied Computing*, Limassol, Cyprus.

4. Lunardi, Voos, and Cherri (2018). An Imperialist Competitive Algorithm for a Real-World Flexible Job Shop Scheduling Problem". *IEEE International Conference on Emerging Technologies and Factory Automation*, Torino, Italy.

5. Lunardi and Voos (2018a). An Extended Flexible Job Shop Scheduling Problem With Parallel Operations. *ACM Applied Computing Review*.

6. Lunardi, Cherri, and Voos (2018). A Mathematical Model and a Firefly Algorithm for an Extended Flexible Job Shop Problem With Availability Constraints. *International Conference on Artificial Intelligence and Soft Computing*, Zakopane, Poland.

7. Lunardi and Voos (2018b). Comparative Study of Genetic and Discrete Firefly Algorithm for Combinatorial Optimization. *ACM Symposium on Applied Computing*, Pau, France.

The following publications are not related to the problem addressed in this thesis:

1. Faria Jr, Lunardi, and Voos (2019). A Parallel Multi-Population Biased Random-Key Genetic Algorithm for Electric Distribution Network Reconfiguration. *Genetic and Evolutionary Computation Conference Companion*, Prague, Czech Republic.

2. Lunardi et al. (2016). Automated Decision Support Iot Framework. *IEEE International Conference on Emerging Technologies and Factory Automation*, Berlin, Germany.

1.6 Outline of the Thesis

This thesis consists of five further chapters, in which some of these chapters are based on published material.

In Chapter 2, scheduling problem characteristics that are useful in modeling the OPS scheduling problem are identified. Scheduling components such as machine environment, constraints, and optimization criteria are addressed. Given the introduction of required scheduling concepts, OPS manufacturing system is described, and the OPS scheduling problem notation such as constants and sets are introduced. The literature review is presented by considering works that deal with formulations and/or practical

applications of the FJS scheduling problem, and the FJS scheduling problem with sequencing flexibility.

Chapter 3 presents the works of literature review related to the optimization strategies for the JS scheduling problem and the FJS scheduling problem. This detailed look into the techniques supports the choice for the optimization strategies used in this thesis to model and solve the OPS scheduling problem. Additionally, the main concepts related to the methods proposed in the following chapters are described.

In Chapter 4 provide in-depth knowledge of the OPS scheduling problem. The proposed MILP formulation and CP formulation of the OPS scheduling problem are presented. Experiments are performed to verify the size of the instances that IBM ILOG CP Optimizer can solve to optimality in comparison to IBM ILOG CPLEX. Both models are validated and analyzed comparatively, and a benchmark set with known optimal solutions is proposed. Additional experiments are performed to analyze the capability of IBM ILOG CP Optimizer to find good quality solutions when applied to large-sized instances of the size of the instances of the OPS scheduling problem that appears in practice.

Chapter 5 presents the proposed representation scheme, constructive heuristic, neighborhood function, LS procedure, and metaheuristics for the OPS scheduling problem. The representation scheme, the decoder, and the LS strategy are evaluated in connection with four metaheuristics. Two of the metaheuristics, Genetic Algorithm (GA), and Differential Evolution (DE) are population-based methods, while the other two, Iterated Local Search (ILS) and Tabu Search (TS), are trajectory methods. Numerical experiments with even larger instances are performed, and the four metaheuristics are compared to each other and against the IBM ILOG CP Optimizer. Finally, numerical experiments with instances of the FJS scheduling problem and the FJS scheduling problem with sequencing flexibility are performed in order to assess the effectiveness of the proposed method that achieved the best performance concerning state-of-the-art methods in the literature.

The thesis concludes in Chapter 6 with the findings, a discussion of the contributions of this study, and the research direction of future outlook.

Chapter 2

Scheduling

The task scheduling problem refers to allocating resources to tasks, respecting the imposed constraints aiming the optimization of one or a combination of performance measures. In the problem, we must allocate resources, usually limited, to activities to be carried out. In this way, the best way to process tasks is sought by defining the start and end time of processing each task. Several definitions for scheduling can be found in the literature. A widely known definition is given by Pinedo (2012), "*Scheduling concerns the allocation of limited resources to tasks over time. It is a decision-making process that has as a goal the optimization of one or more objectives.*" According to Leung (2004), resources may be machines in an assembly plant, central processing unit, memory, and input/output devices in a computer system, runways at an airport, and mechanics in an automobile repair shop. Activities may be various operations in a manufacturing process such as the execution of a computer program, landings and take-offs at an airport, and car repairs in an automobile repair shop. There are also many different performance measures to optimize. One objective may be the minimization of the makespan, i.e., time difference between the start and finish of a sequence of operations, while another objective may be the minimization of the number of late jobs.

In this chapter, to identify possible problem characteristics that are useful in modeling the OPS scheduling problem, scheduling components such as machine environment, constraints, and optimization criteria are addressed in Section 2.1. The OPS manufacturing system is described in Section 2.2. The OPS scheduling problem notation such as constants and sets are introduced in Section 2.2.1. The literature review is presented in Section 2.2.2 by considering works that deal with formulations and/or practical applications of the FJS scheduling problem, and the FJS scheduling problem with sequencing flexibility.

Table 2.1: Shop models and their symbols classified according to the machine environment and the flow of jobs on the production floor.

Environment	Symbol	Interpretation
Single machine	1	There is only one machine. This case is a special case of all other more complicated machine environments
Parallel machine	Pm	There are m machines in parallel. Each job requires a single operation and may be processed on any machine.
Open shop	Om	Each job needs to be processed exactly once on each of the m machines. The route, i.e., order in which the operations are processed, can vary freely.
Flow shop	Fm	There are m machines linearly ordered and all jobs follow the same route, i.e., they have to be processed first on machine 1, then on machine 2, and so on.
Flexible flow shop	FFc	A flexible flow shop is a generalization of the flow shop and the parallel machine environments. Instead of m machines in series, there are c stages in series with at each stage a number of identical machines in parallel. Each job has to be processed first at stage 1, then at stage 2, and so on.
Job shop	Jm	Each job has its own predetermined route to follow. It may visit some of the m machines more than once and it may not visit some machines at all.
Flexible job shop	FJc	Instead of m machines in series, there are c work centers, where each work center consists of a number of unrelated machines in parallel. When a job on its route through the system arrives at a bank of unrelated machines, it may be processed on any one of the machines, but its processing time now depends on the machine on which it is processed.

2.1 Scheduling Notation and Classification

Problem classification is an essential prerequisite to the selection of a suitable solution strategy since information regarding problem complexity and existing algorithms provide useful points of departure for new algorithm development (T'kindt and Billaut, 2006). Different scheduling notation methods have been proposed; the two most commonly used are the one proposed by Conway, Miller, and Maxwell (2003), which is suitable for basic scheduling problems, and the one proposed by Graham et al. (1979) being more appropriate for non-basic scheduling problems involving complex job constraints. According to Graham et al.'s three field scheduling notation, a scheduling problem is described by a triplet $\alpha|\beta|\gamma$ where α denotes the machine environment, β denotes the constraints placed on the jobs, and γ denotes the scheduling criteria. Note that multiple symbols may appear in each field, e.g., $FJ|\text{chains}, r_i|C_{\max}$ where FJ denotes that the problem is a FJS scheduling problem, "chains" implies that the jobs are linear process plans, i.e., precedence between operations are given by a linear order, r_i implies that job i must start after its release time r_i, and $C_{\max}$ implies that the objective is to minimize the makespan.

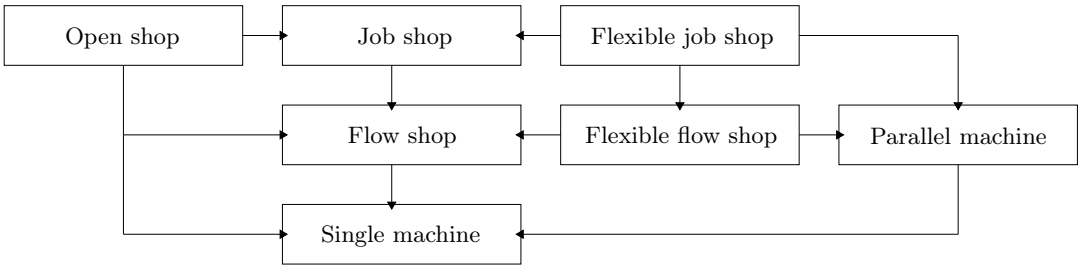


Figure 2.1: The relationships between the different scheduling problems based on machine environments.

2.1.1 Machine Environment

The first field of Graham et al.'s three field notation denotes the configuration of primary resources — usually machines — and the flow of jobs on the production floor. Different flow patterns can be encountered, depending on the number of operations required to process a job and on the number of available machines per operation. When a job requires only one operation for its completion, we characterize it as single-operation; otherwise, we call it multi-operation. In the latter case, the concept of routing may be introduced based on machines; we have Single Machine (SM), Flow Shop (FS), JS, and Open Shop (OS) scheduling problems. When processing stages are considered instead of machines, we have Parallel Machine (PM), Flexible Flow Shop (FFS), and FJS scheduling problems. In Table 2.1 a description of each shop model and its α symbol is presented according to Pinedo (2012) and Leung (2004). When a shop model symbol appears in the α field means that the scheduling problem being addressed can be modeled as the shop that the symbol represents.

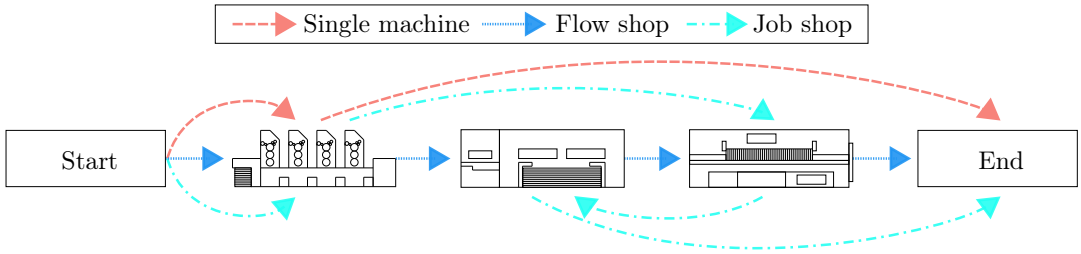
Figure 2.1 illustrates the relationships between scheduling problems based on their distinct machine environments based on the scheduling problems classification introduced in (Zandieh, Ghomi, and Husseini, 2006). The figure shows that all of the models can, in fact, be described as either generalization of, or specific instances of the classical job shop scheduling problem.

The JS scheduling problem can be stated as follows. Consider a set $V = \{1, 2, \ldots, o\}$ of operations, and a set $M = \{1, 2, \ldots, m\}$ of machines. Each job $J_i (i = 1, 2, \ldots, n)$ where $J_i \subseteq V$ consists of a set of n_i operations that must processed by machines in order to produce a product. A precedence relation saying that an operation i must be processed before an operation j is represented by an arc $(i, j) \in A$, where A is the set of precedence relations. In the classical JS scheduling problem, precedence relations between operations belonging to the same job are denoted in form of chains (sequence), e.g., given an instance with two jobs $J_i = \{1, 2, 3, 4\}$ and $J_2 = \{4, 5\}$, we simply have that $A = \{(1, 2), (2, 3), (3, 4), (4, 5)\}$. There is a machine $k_i \in M$ and a processing time p_i associated with each operation. It implies that operation i must be processed

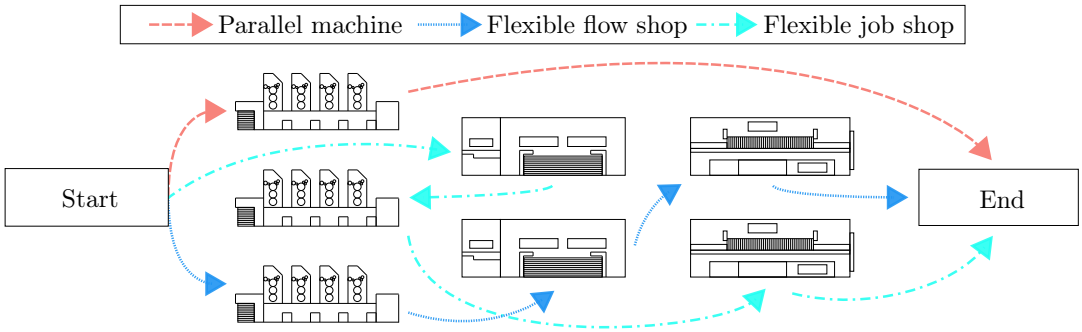
for p_i time units on machine k_i. Thus, considering s_i as the starting time of operation i, we have that the completion time c_i of operation i is equal to s_i+p_i, i.e., $c_i = s_i+p_i$. Moreover, for each $(i, j) \in A$ we have that $c_i \leq s_j$. No operation may be interrupted and each machine can process only one operation at a time. The problem consists in sequencing all operations on their respective machines, i.e, ordering processing of all operations on all machines to obtain a feasible scheduling solution. The definition of feasible scheduling solution will be elucidated in Section 2.1.2.

Grobler (2009) indicates that, by comparing, it becomes more evident that all the models can be described as either a generalization of or specific instances of the JS scheduling problem. Figure 2.2.a illustrates simple flow patterns of "single resource per operation" models, such as the SM scheduling problem, the FS scheduling problem, and the JS scheduling problem. The SM scheduling problem is a simple JS scheduling problem with only one operation per job. The FS scheduling problem is restricted to specific applications where identical routings are defined for each of the jobs. Figure 2.2.b illustrates simple flow patters of "multiple resources per operation" models, such as the PM scheduling problem, the FFS scheduling problem, and the FJS scheduling problem. The PM scheduling problem is limited to jobs consisting of only one operation, which can be performed on any one of a number of resources. The FFS scheduling problem is a generalization of the FS scheduling problem, which in addition to the identical job routes, each stage can be processed by one of the multiple parallel machines. In contrast to most of the previously mentioned problems, the OS scheduling problem formulation results from a relaxation of the operation precedence constraints, i.e., there is no restriction related to the order in which operations of a job are processed.

The FJS scheduling problem, also known as the JS scheduling problem with duplicate machines, first introduced by Nuijten and Aarts (1996), is a generalization of the JS scheduling problem in which it is considered that there may be several machines, not necessarily identical, capable of processing each operation. Additionally to the set V of operations, the set A of precedence relations, and the set M of machines, it is also given a function F that associates a non-empty subset $F(i)$ of M with each operation $i \in V$. The machines in $F(i)$ are those capable of processing operation i. Moreover, for each operation i and each machine $k \in F(i)$, a positive number p_{ik} is given representing the processing time of operation i on machine k. The FJS scheduling problem is twofold, assigning one machine to each operation (routing), and sequencing the operations on the machines (scheduling). Firstly, *routing* is concerned with assigning a machine to each operation of all jobs, i.e., defining the "route" of the jobs through the machines. A *machine assignment* is a function κ that assigns a machine $\kappa(i) \in F(i)$ with each operation i. Given a machine assignment κ, consequently, the processing time of



(a) SM, FS, and JS shop scheduling problems.



(b) PM, FFS, and FJS shop scheduling problems.

Figure 2.2: Flow in which jobs can assume in different shop environments.

operation i is equal to $p_{i\kappa(i)}$. Secondly, *sequencing* is concerned with ordering the processing of the operations to obtain a feasible scheduling solution. Commonly, the objective is to minimize the makespan, however, not limited to, as explained in Section 2.1.3.

The JS scheduling problem is known to be NP-hard, being considered by many one of the most difficult problems in combinatorial optimization (Garey, Johnson, and Sethi, 1976; Lawler et al., 1993; Nuijten and Aarts, 1996; Jain and Meeran, 1999). Therefore, the scheduling problem in an FJS is at least as hard as the JS scheduling problem because it contains an additional problem that is assigning operations to the machines. The option of alternative resources ensures that the FJS scheduling problem is useful for scheduling in a wider range of production systems including FMS and PM shops (Chung et al., 2005).

2.1.2 Job Characteristics

The second field of Graham et al.'s three field scheduling notation denotes the constraints being addressed in the scheduling problem. In the classic JS scheduling problem, once all operations $i \in V$ are sequenced and their starting time s_i and completion time c_i are determined, a scheduling solution is considered feasible if: operations are not overlapping on the same machine; operations of the same job are not overlapping. However, there are several JS scheduling problem variants which encapsulate

Table 2.2: Distinct constraint and their respective symbols which may appear in the β field.

Constraint	**Symbol**	**Interpretation**
Release	r_i	Job i cannot start its processing before r_i
Due dates	d_i	Job i must finish not later than d_i
Recirculation	rcrc	Operations may be processed several times on the same machine
Preemption	prmp	Operation's processing can be interrupted
Fixed operation	$\bar{s}_i$	Operation i must start at a predetermined time point
Batch processing	batch	Operations are grouped into predefined batches
Overlapping	overlap	Operations related by precedence constraints may overlap in time
Significant machine unavailable periods are present		
	fixed	Unavailable periods start at a predetermined time point
	non-fixed	Starting time of the unavailable period must be determined
	resumable	Processing of an operation can be interrupted by an unavailable period
	non-resumable	Processing of an operation cannot be interrupted by an unavailable period
Precedence relations are considered, and it is described in form of		
	dag	an arbitrary acyclic directed graph
	intree	a rooted tree with an outdegree for each vertex of at the most one
	outtree	a rooted tree with an indegree for each vertex of at the most one
	chains	a tree with outdegree and indegree for each vertex is at the most one
Sequence-dependent setup times are required		
	s_{ij}	Setup time between operations i and j (machine independent)
	s_{ijk}	Setup time between operations i and j on machine k (machine dependent)

additional constraints. In these cases, a schedule is considered feasible if it meets a number of problem-specific characteristics. Leung (2004), Brucker and Brucker (2007), Pinedo (2012), and T'kindt and Billaut (2006) identified various types of constraints on the job characteristics, which, when included, may significantly affect the realism of the scheduling model. A number of constraints that may appear in a job shop environment together with its β symbol are presented in Table 2.2.

Chaudhry and Khan (2016) presents a survey of various constraints and techniques that have been employed since 1990 for the FJS scheduling problem. Out of 197

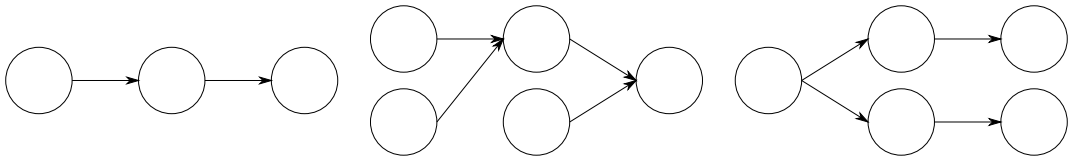
journal papers, only 70 papers (35.53%) considered additional constraints such as: *sequencing flexibility* (Lee et al., 2012a; Birgin et al., 2014; Yu et al., 2017), when an operation can have more than one predecessor or successor; *setup and transportation times* (Zhang, Manier, and Manier, 2012; Rossi, 2014; Karimi et al., 2017), where machines must be configure to process an operation, and transportation times are considered between operations of the same job, respectively; *preemption* (Zhang and Yang, 2016; Khorasanian and Moslehi, 2017), where processing of operations can be interrupted and continued at a later stage; *batches* (Jeong, Park, and Leachman, 1999; Loukil, Teghem, and Fortemps, 2007; Calleja and Pastor, 2014; Ham and Cakici, 2016; Xiong et al., 2017), where processing times are determined by the number of similar products which are produced simultaneously as a single operation; *machine availability, breakdown, and maintenance* (Jensen, 2003; Gholami and Zandieh, 2009; Wang and Yu, 2010; Al-Hinai and ElMekkawy, 2011; Xiong, Xing, and Chen, 2013), where machines may be unavailable for some periods of time; *overlapping* (Leung et al., 2005; Fattahi, Jolai, and Arkat, 2009; Demir and İşleyen, 2014), where operations of the same job related by precedence constraints and processed in different machines may overlap; *uncertain processing times* (Lei, 2012; Xu et al., 2015), which considers fuzzy processing times. *Dynamic events* (Rajabinasab and Mansour, 2011; Nie et al., 2013; Shen and Yao, 2015), where dynamic events such as stochastic job arrivals are considered.

Four of the constraints described above warrant further explanation due to their relevance to the problem considered in this work. These constraints include the sequencing flexibility, overlapping between operations, sequence-dependent setup times, and machine availability.

2.1.2.1 Sequencing Flexibility

According to Saygin and Kilic, 1999, conventional process plans can be considered as linear directed graphs, as shown in Figure 2.3.a. In the related literature, they are also referred to as linear process plans, sequential process plans or even static process plans. Flexible process plans are net-shaped in nature. They are also called nonlinear process plans or dynamic process plans. Based on the fact that increased flexibility will increase the decision domains and provide the potential for improved performance, Hutchinson and Pflughoeft, 1994 defined three classes of process plan flexibility:

1. *Routing flexibility* provides a choice of machine type to perform the operation rather than dictating a specific machine.



(a) Chain (b) Intree (c) Outtree

Figure 2.3: Example of three jobs with different precedence relation types. Nodes represent operations and arrows represent precedence relations.

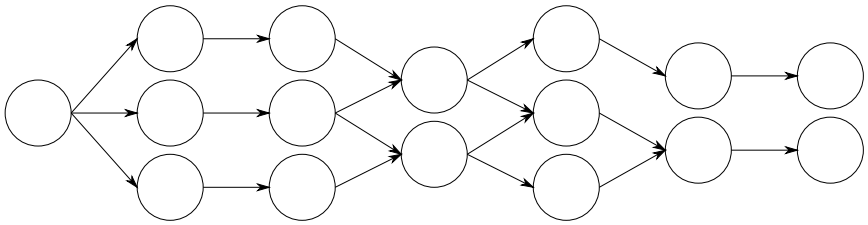


Figure 2.4: Example of a job with its precedence relation is given by an acyclic directed graph. Nodes represent operations and arrows represent precedence relations.

2. *Sequencing flexibility* occurs when an operation can have more than one predecessor or successor.

3. *Process plan flexibility* provides alternative paths for the completion of the plan.

Precedence relations between operations indicate that the start of an operation is conditioned by the end of all the preceding operations, i.e., for all $(i, j) \in A$ we have that $s_i + p_{i\kappa(i)} \leq s_j$. The FJS scheduling problem considers linear process plans. For example, given a job $J_i = \{1, 2, 3\}$ we have that operation 1 precedes 2, and operation 2 precedes 3, i.e., $(1, 2), (2, 3) \in A$. However, as mentioned above, in practice there can be multiple connections and finish-to-start dependencies among the operations of each job. Pinedo (2012) categorizes processes plans into *chains* (linear), *intree*, and *outree* process plan. If each operation has at most one successor, the constraints are referred to as an intree. If each operation has at most one predecessor the constraints are referred to as an outtree. Figure 2.3 presents three jobs with chain, intree, and outtree precedence relations types. Additionally, a job can be categorized as "dag" when precedence relations are given by an Directed Acyclic Graph (DAG). Figure 2.4 shows an example of a job with DAG precedence relation type.

For a comprehensive review in process planning, see (Isnaini and Shirase, 2014). For related works involving scheduling with sequencing flexibility, see (Birgin et al., 2014; Yu et al., 2017; Lunardi, Voos, and Cherri, 2019; Andrade-Pineda et al., 2019).

2.1.2.2 Overlap Between Operations

For any two operations i and j connected by a precedence relation (i, j), we have that an operation i must be processed before an operation j. Besides that, operations'

overlap states that c_i and s_j may overlap in time. Note that the overlapping is only possible when $\kappa(i) \neq \kappa(j)$. The extent of this overlapping is limited by a fraction that defines the proportion of operation i that has to be processed before its successor j can start. Moreover, the precedence implies that j cannot be completed before i. For example, although there is a precedence relationship between a printing and cutting operation, there is no need to wait until the printing of all sheets is completed to begin cutting; clearly, it is not possible to complete the cutting operation before printing. In this work, simmilarly to (Fattahi, Jolai, and Arkat, 2009), each operation $i \in V$ is associated with a quantity $\theta_i \in (0, 1]$ that represents the fraction $p'_i = \lceil \theta_i p_{i\kappa(i)} \rceil$ of operation i that must be completed before starting the execution of any operation j such that $(i, j) \in A$, i.e., for all $(i, j) \in A$, we have that $s_i + p'_i \leq s_j$ and $c_i \leq c_j$.

Related works of scheduling with overlapping constraints can be found in (Wagneur and Sriskandarajah, 1993; Leung et al., 2005; Fattahi, Jolai, and Arkat, 2009; Demir and İşleyen, 2014; Zhang and Yang, 2016; Meng, Pan, and Sang, 2018).

2.1.2.3 Sequence-Dependent Setup Times

Setup time is the time taken to clean and configure a machine that is incurred between the processing of two distinct operations. The integration of setup times into the JS scheduling problem, dated to the early 1960s (Wilbrecht and Prescott, 1969), has a significant impact on its applicability and performance for practical manufacturing systems (Wilbrecht and Prescott, 1969; Krajewski et al., 1987). This additional complexity is particularly useful in modeling situations where cleaning operations and tool changes play an important role in production (Grobler, 2009). Pinedo (2012) categorizes setup times into *machine-independent*, where s_{ij} time units are incurred between the processing of operation i and j; and *machine-dependent*, where s_{ijk} time units are incurred between the processing of operation i and j on the machine k. The length of the setup time on the machine depends on the similarities between the two consecutive orders, e.g., the number of colors in common, the differences in size, and so on.

Detailed reviews of scheduling with setup times are presented in (Yang, 1999; Allahverdi, Gupta, and Aldowaisan, 1999; Allahverdi et al., 2008).

2.1.2.4 Machine Availability

The continuous availability of machines during the whole scheduling horizon is an assumption that might be justified in some cases but cannot apply to all industrial

settings. Machine unavailable periods might be consequent of pre-scheduling, preventive maintenance, shift pattern, or the overlap of two consecutive time horizons in the rolling time horizon planning algorithm. According to Gao, Gen, and Sun (2006), availability constraints are categorized into two types:

- *Fixed*, when the starting time of unavailable periods are predetermined;
- *Non-fixed*, when the starting time unavailable periods must be determined.

Additionally, machine availability constraints are also categorized into:

- *Resumable*, when the processing of an operation can be interrupted by an unavailable period and resumed afterward;
- *Semiresumable*, similar to the resumable case with an additional that states that the operation must be partially restarted;
- *Non-resumable*, when the processing of an operation cannot be interrupted by an unavailable period.

Detailed reviews of scheduling problems with machine availability constraints can be found in (Ma, Chu, and Zuo, 2010; Tamssaouet, Dauzère-Pérès, and Yugma, 2018).

2.1.3 Optimality Criteria

The third and final field of Graham et al.'s three field scheduling notation is related to the objective function to be minimized. The objective to be minimized is always a function of the completion times of the operations, which, of course, depend on the scheduling solution (Pinedo, 2012). The following notations are required to formulate some of the addressed objective functions. Let $C_i (i = 1, \ldots, n)$ be the latest completion time between all operations of job i, i.e., $C_i = \max_{\forall j \in J_i}\{c_j\}$. Let $d_i (i = 1, \ldots, n)$ be the due date of job J_i. Finally, for each machine k, let $B_k = \{i \in V : \kappa(i) = k\}$ be the set of operations assigned to be processed on machine k.

Given the reviews provided in (Çaliş and Bulkan, 2015; Chaudhry and Khan, 2016), the six most-used objective functions in the JS scheduling problem literature are described and formulated in Table 2.3. In (Chaudhry and Khan, 2016) it is shown that the makespan is the most used objective function; out of 197 papers, 44.67% considered the minimization of the makespan, and 23.35% considered the minimization of the makespan, maximum workload, and total workload.

The description of the third field concludes the analysis of Graham et al.'s three field scheduling notation. The next section will introduce the OPS manufacturing system and the OPS scheduling problem in more detail before attempting to contextualize the problem within the existing literature.

Table 2.3: Commonly used objective functions in the FJS scheduling problem. Symbol.

Objective	Symbol	Formulation	Interpretation
Makespan	$C_{\max}$	$\max_{\forall i \in 1,\dots,n} \{C_i\}$	The maximal completion time between all operations of all jobs
Maximum workload	W_M	$\max_{\forall k \in 1,\dots,m} \{\sum_{i=1}^{\lvert B_k \rvert} p_{ik}\}$	The machine with most working time; appropriate to balance the workload among machines
Total workload	W_T	$\sum_{i=1}^{o} p_{i\kappa(i)}$	Sum of the working time on all machines; appropriate to reduce the total amount of processing time taken to produce all operations
Total tardiness	T	$\max_{\forall i \in 1,\dots,n} \{0, C_i - d_i\}$	Positive difference between the completion time and the due date of all jobs; appropriate when early jobs bring no reward
Mean tardiness	$\bar{T}$	$\max_{\forall i \in 1,\dots,n} \{0, C_i - d_i\}$	Positive difference between the completion time and the due date of all jobs; appropriate when early jobs bring no reward
Maximum lateness	$L_{\max}$	$\max_{\forall i \in 1,\dots,n} \{C_i - d_i\}$	Maximal difference between the completion time and the due date of all jobs; appropriate when there is a positive reward for completing a job early

2.2 Online Printing Scheduling Problem

The OPS scheduling problem represents a real problem of the nowadays printing industry. OPSs receive a wide variety of online orders of diverse clients per day. Orders include the production of books, brochures, calendars, cards (business, Christmas, or greetings cards), certificates, envelopes, flyers, folded leaflets, as well as beer mats, paper cups, or napkins, among many others. Naturally, the production of the ordered items includes a printing operation. Aiming to reduce the production cost, a cutting stock problem is solved to join the printing operations of different placed orders. These merged printing operations are known as *ganging operations*. The production of the orders whose printing operations were ganged constitutes a single job. Operations of a job also include cutting, embossing (e.g., varnishing, laminating, hot foil, etc.), and folding operations. Each operation must be processed on one out of multiple machines with varying processing times. Due to their nature, the structure of the jobs, i.e., the number of operations and their precedence relations, as well as the routes of the jobs through the machines, are completely different. Multiple operations of the same type may appear in a job structure. For example, in the production of one or several copies of a book, that could be part of a job, there are commonly required multiple independent printing operations corresponding to the book cover and the book pages.

Disassembling and assembling operations are also present in a job structure, e.g., at

some point during the production of a book, cover and pages must be gathered together. A simple example of a disassembling operation is the cutting of the printed material of a ganged printing operation. Another example of a disassembling operation is present in the production of catalogs. Production of catalogs for some franchise usually presents an elaborate production plan composed of several operations (e.g., printing, cutting, folding, embossing, etc.). Once catalogs are produced, the production is branched into several independent sequences of operations, i.e., one sequence for each partner. This is due to the fact that, for each partner and its associated catalogs, a printing operation must be performed in the catalog cover to denote its address and other information. Subsequently, each catalog must be delivered to its respective partner.

Several important factors that have a direct impact on the manufacturing system and its efficiency must be taken into consideration in the OPS scheduling problem. Machines are flexible, which means they can perform a wide variety of tasks. To produce something in a flexible machine requires the machine to be configured. The configuration or setup time of a machine depends on the current configuration of the machine and the characteristics of the operation to be processed. A printing operation has characteristics related to the size of the paper, its weight, the required set of colors, and the type of varnishing, among others. The more different two consecutive operations processed on the same machine, the more time consuming the setup. Thus, setup operations are sequence-dependent.

Working days are divided into three eight-hour shifts, namely, morning, afternoon/evening, and overnight shift, in which different workers groups perform their duties. However, the presence of all three shits depends on the working load. When a shift is not present, the machines are considered unavailable. In addition to shift patterns, other situations such as machines' maintenance, pre-scheduling, and overlapping of two consecutive time planning horizons imply machines' downtimes. Operations are resumable, in the sense that the processing of an operation can be interrupted by a period of unavailability of the machine to which the operation has been assigned; the operation being resumed as soon as the machine returns to be active. On the other hand, setup operations can not be interrupted; and the end of a setup operation must be immediately followed by the beginning of its associated regular operation. This is because a setup operation might include cleaning a machine before the execution of an operation. If we assume that a period of unavailability of a machine corresponds to pre-scheduled maintenance, the machine can not be opened and half-cleaned, the maintenance operation executed, and then the cleaning operation finished after the interruption. The same situation occurs if the period of unavailability corresponds to a night shift during which the store is closed. In this case, the half-cleaned opened

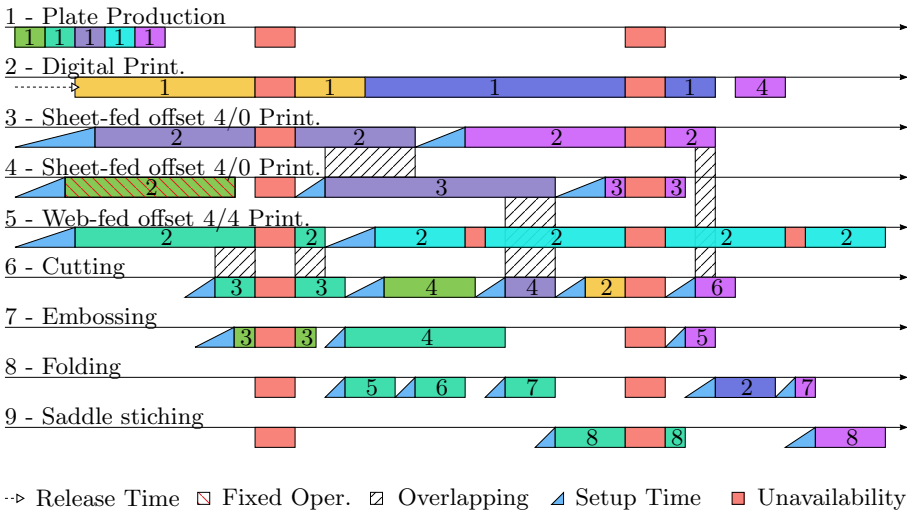


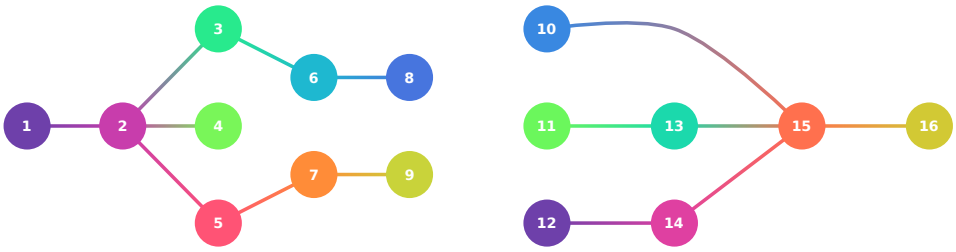
Figure 2.5: Gantt chart of a solution to a simple instance of the considered scheduling problem containing 9 machines and 7 jobs.

machine could get dirty because of dust or insects during the night.

Operations that compose a job are subject to precedence constraints. The classical conception of precedence among a pair of operations called predecessor and successor means that the predecessor must be fully processed before the successor can start to be processed. However, in the OPS scheduling problem, some operations connected by a precedence constraint may overlap to a certain predefined extent, e.g., a cutting operation preceded by a printing operation may overlap its predecessor — If the printing operation consists in printing a certain number of copies of something, already printed copies can start to be cut; while some others are still being printed.

Fixed operations (i.e., with starting time and machine established in advance) can also be present in the OPS. This is because customers may choose to visit the OPS to check the quality of the outcome product associated with that operation. This is mainly related to printing quality, so most fixed operations are printing operations. Fixed operations are also useful to assemble the scheduling being executed of a planning horizon that is ending with the scheduling of a new planning horizon that must be computed. In this way, sequence-dependent setup times can be properly considered.

This thesis addresses the problem of minimizing the makespan of an OPS scheduling problem by considering it as a FJS scheduling problem with sequencing flexibility and a wide variety of challenging features, such as non-trivial operations' precedence relations given by an arbitrary DAG, partial overlapping among operations with precedence constraints, periods of unavailability of the machines, resumable operations, sequence-dependent setup times, release times, and fixed operations. According Graham et al.'s three field scheduling notation, the OPS scheduling problem can be denoted as $FJ|r_i, s_{ijk}, \bar{s}_i, \text{prmp}, \text{overlap}, \text{fixed}, \text{resumable}, \text{dag}|C_{\max}$.



(a) Job 1 with 9 operations (b) Job 2 with 7 operations

Figure 2.6: Directed acyclic graph representing precedence constraints between operations of two different jobs with 9 and 7 operations, respectively.

The OPS scheduling problem is NP-hard since it includes, as a particular case, the job shop scheduling problems, known to be strongly NP-hard (Garey, Johnson, and Sethi, 1976). Figure 2.5 presents a Gantt chart of a toy instance of the OPS scheduling problem. The figure aims to illustrate a small set of diverse jobs in a complicated system that includes features such as periods of unavailability of the machines, release times, setup operations, overlapping between operations of a job with precedence constraints, etc.

2.2.1 Problem Notation

In the OPS scheduling problem, there are n jobs and m machines. Each job i is decomposed into o_i operations with arbitrary precedence constraints represented by a DAG. For simplicity, it is assumed that operations are numbered consecutively from 1 to $o := \sum_{i=1}^{n} o_i$; and all n disjoint DAGs are joined together into a single DAG $D(V, A)$, where $V = \{1, 2, \ldots, o\}$ and A is the set all arcs of the n individual DAGs. For each operation $i \in V$, there is a set $F(i) \subseteq \{1, \ldots, m\}$ of machines by which the operation can be processed; the processing time of executing operation i on machine $k \in F(i)$ being given by p_{ik}. Each operation i has a release time r_i. Figure 2.6 presents directed acyclic graph representing precedence constraints between operations of two different jobs with 9 and 7 operations, respectively. Nodes represent operations and arcs, directed from left to right, represent precedence constraints. Operations are numbered consecutively from 1 to 16. So, $V = \{1, 2, \ldots, 16\}$ and $A = \{(1,2), (2,3), (2,4), (2,5), (3,6), (5,7), (6,8), (7,9), (10,15), (11,13), (12,14), (13,15), (14,15), (15,16)\}$.

Machines $k = 1, \ldots, m$ have periods of unavailability given by $[\underline{u}_1^k, \bar{u}_1^k], \ldots, [\underline{u}_{q_k}^k, \bar{u}_{q_k}^k]$, where q_k is the number of unavailability periods of machine k. Although preemption is not allowed, the execution of an operation can be interrupted by periods of unavailability of the machine to which it was assigned; i.e., operations are resumable. The starting time s_i of an operation i assigned to a machine $\kappa(i)$ must be

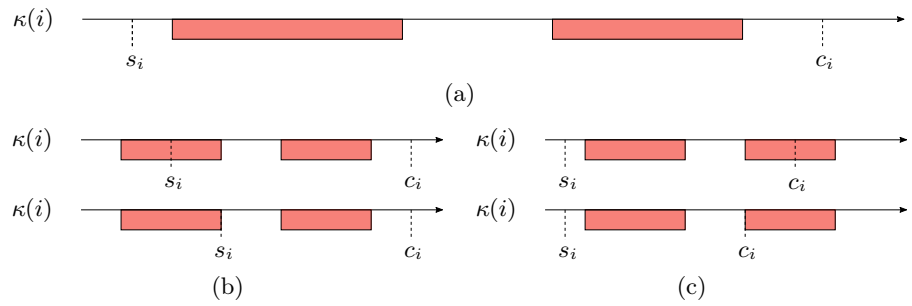


Figure 2.7: Allowed and forbidden relations between the starting time s_i, the completion time c_i and the periods of unavailability of machine $\kappa(i)$.

such that $s_i \notin [\underline{u}_\ell^{\kappa(i)}, \bar{u}_\ell^{\kappa(i)})$ for all $\ell = 1, \ldots, q_{\kappa(i)}$. This means that the starting time may coincide with the end of a period of unavailability (the possible existence of a non-null setup time is being ignored here), but it can not coincide with its beginning nor to belong to its interior, since these two situations would represent an ahead fictitious starting time. In an analogous way, the completion time c_i must be such that $c_i \notin (\underline{u}_\ell^{\kappa(i)}, \bar{u}_\ell^{\kappa(i)}]$ for all $\ell = 1, \ldots, q_{\kappa(i)}$, since violating these constraints would represent to allow a delayed fictitious completion time; it is clear that if operation i is completed at time c_i and $c_i \in (\underline{u}_\ell^{\kappa(i)}, \bar{u}_\ell^{\kappa(i)}]$ for some ℓ then it is because, in fact, the operation is completed at instant $\underline{u}_\ell^{\kappa(i)}$. Figure 2.7 illustrates the allowed and forbidden relations between the starting time s_i, the completion time c_i and the periods of unavailability of machine $\kappa(i)$. In (a), allowed positions are illustrated. For further reference, it is worth mentioning that the sum of the sizes of the two periods of unavailability in between s_i and c_i is named u_i; so the relation $s_i + p_{i,\kappa(i)} + u_i = c_i$ holds. The top picture in (b) shows the forbidden situation $s_i \in [\underline{u}_\ell^{\kappa(i)}, \bar{u}_\ell^{\kappa(i)})$ for some ℓ, that corresponds to an ahead ficticius starting time. The valid value for s_i that corresponds to the same situation is illustrated in the bottom picture in (b). The top picture in (c) shows a forbidden situation in which $c_i \in (\underline{u}_\ell^{\kappa(i)}, \bar{u}_\ell^{\kappa(i)}]$ for some ℓ, that corresponds to a delayed ficticius completion time. The valid value for c_i that corresponds to the same situation is illustrated in the bottom picture in (c).

The precedence relations $(i, j) \in A$ have a special meaning in the OPS scheduling problem. Each operation i has a constant $\theta_i \in (0, 1]$ associated with it. On the one hand, the precedence relation means that operation j can start to be processed after $\lceil \theta_i \times p_{ik} \rceil$ units of time of operation i have already been processed, where $k \in F(i)$ is the machine to which operation i has been assigned. On the other hand, the precedence relation imposes that operation j can not be completed before the completion of operation i. For a generic operation h assigned to machine $\kappa(h)$, $\bar{c}_h$ denotes the instant at which $\lceil \theta_h \times p_{h,\kappa(h)} \rceil$ units of time of operation h have already been processed. Note that $\bar{c}_h$ could be larger than $s_h + \lceil \theta_h \times p_{h,\kappa(h)} \rceil$ due to the machines' periods of unavailability. Figure 2.8 gives an example of overlapping between operations.

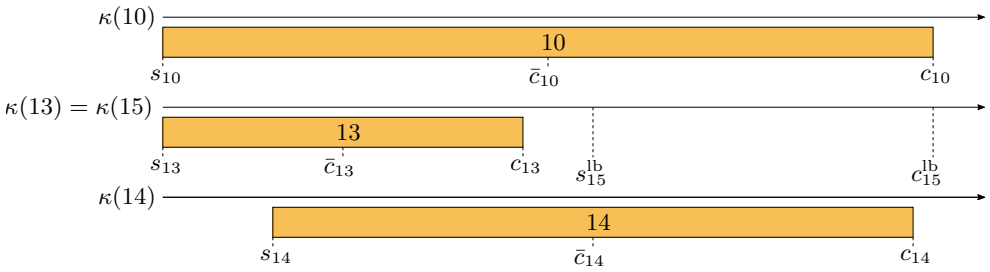


Figure 2.8: Overlapping between the operations with precedence relations. In this illustration, precedence is given by the graph in the right-hand-side of Figure 2.6.

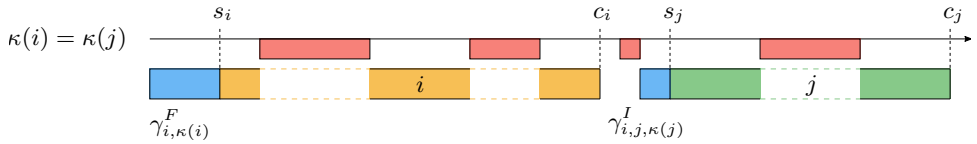


Figure 2.9: Illustration of the fact that, differently from the processing of a regular operation, a setup operation can not be interrupted by periods of unavailability of the machine to which the operation has been assigned.

According to the DAG in the right-hand-side of Figure 2.6, we have that $(10, 15)$, $(13, 15)$, and $(14, 15) \in A$. This means that $s^{\text{lb}}_{15} = \max\{\bar{c}_{10}, \bar{c}_{13}, \bar{c}_{14}\}$ is a lower bound for the starting time s_{15}; while $c^{\text{lb}}_{15} = \max\{c_{10}, c_{13}, c_{14}\}$ is a lower bound for the completion time c_{15}. If $\kappa(15) = \kappa(13)$ and operation 15 is sequenced right after operation 13, then $c_{13}+\gamma^I_{13,15,\kappa(15)}$ is another lower bound for s_i, where $\gamma^I_{13,15,\kappa(15)}$ is the sequence-dependent setup time corresponding to the processing of operation 13 right before operation 15 on machine $\kappa(15)$. In addition, s_{15} must also satisfy $s_{15} \geq r_{15}$.

Operations have a sequence-dependent setup time associated with them. If the execution of operation j on machine k is immediately preceded by the execution of operation i, then its associated setup time is given by γ^I_{ijk} (the supra-index "I" stands for *intermediate* or *in between*); while, if operation j is the first operation to be executed on machine k, the associated setup time is given by γ^F_{jk} (the supra-index "F" stands for *first*). Of course, setup times of the form γ^F_{jk} are defined if and only if $k \in F(j)$; while setup times of the form γ^I_{ijk} are defined if and only if $k \in F(i) \cap F(j)$. Differently from the execution of an operation, the execution of a setup operation can not be interrupted by periods of unavailability of the corresponding machine, i.e., setup operations are not resumable. Moreover, the completion time of the setup operation must coincide with the starting time of the associated operation. Figure 2.9 illustrates the fact that, differently from the processing of a regular operation, a setup operation can not be interrupted by periods of unavailability of the machine to which the operation has been assigned. The picture also illustrates that the completion time of the setup operation must coincide with the starting time of the operation itself. In the picture, it is assumed that operation i is the first operation to be executed on machine $\kappa(i)$; thus, the duration of its setup operation is given by $\gamma^F_{i,\kappa(i)}$.

Finally, the OPS scheduling problem may have some operations that were already assigned to a machine and for which the starting time has already been defined. These operations are known as *fixed* operations. Note that the setup time of the operations is sequence-dependent. Then, the setup time of a fixed operation is unknown and it depends on which operation (if any) will precede its execution in the machine to which it was assigned. Let $T \subseteq V$ be the set of indices of the fixed operations. Without loss of generality, we assume that $T = \{\bar{o}+1, \bar{o}+2, \ldots, o\}$, i.e., that operations $1, 2, \ldots, \bar{o}$ are the non-fixed operations. So, we assume that for $i = \bar{o}+1, \ldots, o$, s_i is given and that $F(i)$ is a singleton, i.e., $F(i) = \{k_i\}$ for some $k_i \in \{1, 2, \ldots, m\}$. Since a fixed operation i has already been assigned to a machine k_i, its processing time $p_i = p_{i,k_i}$ is known. Moreover, the instant $\bar{c}_i$ that is the instant at which $\lceil \theta_i \times p_i \rceil$ units of time of its execution has already been processed, its completion time c_i, and the value u_i such that $s_i + u_i + p_i = c_i$ can be easily computed taking into account the given starting time s_i and the periods of unavailability of machine k_i.

The problem consists of assigning each operation $i \in V$ to a machine $k \in F(i)$ and determining its starting processing time s_i to satisfy all the problem constraints. A machine can not process more than one operation at a time. The processing of operations can only be interrupted by periods of unavailability of the machine to which it was assigned. A setup operation can not be interrupted, and its completion must coincide with the beginning of the associated operation. The objective is to minimize the makespan. Table 2.4 summarizes the constants and sets that define an instance of the OPS scheduling problem.

2.2.2 Literature Review

In the classical FJS scheduling problem, each operation can be processed by a subset of machines, a situation known as routing flexibility. A wide range of techniques have been presented in the literature to deal with the FJS scheduling problem. A MILP model for this problem is given by Fattahi, Mehrabad, and Jolai (2007). An alternative model for the FJS scheduling problem is given in (Özgüven, Özbakır, and Yavuz, 2010). This model also considers process plan flexibility. A comparison among MILP models for the FJS scheduling problem is presented in (Demir and İşleyen, 2013). CP formulations for the FJS scheduling problem are presented in (Vilím, Laborie, and Shaw, 2015) and Laborie, 2018. Numerical results in these works show that the IBM ILOG CP Optimizer (Laborie et al., 2018) improves best-known results for several classical FJS scheduling instances, suggesting that this off-the-shelf solver can compete with or even outperform specialized algorithms.

Table 2.4: Summary of problem data (constants and sets) that define an instance of the problem.

Symbol	Interpretation
o	number of operations
m	number of machines
$V = \{1, 2, \ldots, o\}$	set of operations
$T \subseteq V$	set of fixed operations
$F(i) \subseteq \{1, 2, \ldots, m\}$, $i \in V$	set of machines that can process operation i
$A \subseteq V \times V$	set of operations' precedence relations
$\bar{s}_i$, $i \in T$	starting time for the fixed operation i
r_i, $i \in V$	release time of operation i
p_{ik}, $i \in V$, $k \in F(i)$	processing time of operation i on machine k
$\theta_i \in (0, 1]$, $i \in V$	fraction of operation i that must be processed before an operation j can start to be processed if $(i, j) \in A$
q_k, $k \in \{1, 2, \ldots, m\}$	number of periods of unavailability of machine k
$\underline{u}^k_\ell$ and $\bar{u}^k_\ell$, $\ell \in \{1, \ldots, q_k\}$	begin and end of the ℓth unavailability period of machine k
γ^I_{ijk}, $i, j \in V$, $i \neq j$, $k \in F(i) \cap F(j)$	setup time to process operation j right after operation i on machine k
γ^F_{ik}, $i \in V$, $k \in F(i)$	setup time to process operation i as the first operation to be processed on machine k
$B_k = \{i \in V \mid k \in F(i)\}$, $k \in \{1, 2, \ldots, m\}$	set of operations that can be processed on machine k (auxiliar set that simplifies the presentation of some constraints)

An extensive literature review of formulations and exact and heuristic methods developed in the last three decades to approach the FJS scheduling problem and some extensions is presented in (Chaudhry and Khan, 2016). On the other hand, only a few works, mostly inspired by practical applications from the glass, mold, and printing industries, deal with the FJS scheduling problem with sequencing flexibility. The literature review below, presented in chronological order, focuses on models and practical applications of the FJS scheduling problem with sequencing flexibility. It aims to show that no paper in the literature proposes a model for or describes a practical application of the FJS scheduling problem with sequencing flexibility encompassing all the challenging and complicating features of the OPS scheduling problem. Moreover, up to the author's acknowledge, no paper in the literature tackles this problem with constraint programming techniques. Table 2.5 summarizes the main features of the FJS scheduling problem with sequencing flexibility addressed in each one of the cited references.

In (Gan and Lee, 2002), an FJS scheduling problem with sequencing flexibility and process plan flexibility, is considered. The problem is based on a practical application in a mold manufacturing shop, and it is tackled with a branch and bound algorithm.

	Practical application	MILP formulation	CP formulation	Routing flexibility	Sequencing flexibility	Process plan flexibility	Sequence-dependent setups	Partial overlapping	Resumable operations	Machines' unavailabilities	Release times	Fixed operations
Gan and Lee, 2002	✓			✓	✓	✓						
Kim, Park, and Ko, 2003				✓	✓	✓						
Alvarez-Valdés et al., 2005	✓			✓	✓			✓	✓	✓		
Vilcot and Billaut, 2008	✓			✓	✓							
Lee et al., 2012b	✓	✓		✓	✓	✓					✓	
Birgin et al., 2014		✓		✓	✓							
Birgin, Ferreira, and Ronconi, 2015		✓		✓	✓							
Lunardi, Voos, and Cherri, 2019	✓			✓	✓							
Rossi and Lanzetta, 2020	✓			✓	✓	✓	✓					
Vital-Soto, Azab, and Baki, 2020	✓	✓		✓	✓							
Cao, Lin, and Zhou, to appear	✓			✓	✓		✓					
Andrade-Pineda et al., to appear	✓	✓		✓	✓							
Current work	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓
Sum	10	6	1	13	13	4	3	2	2	2	2	1

Table 2.5: Summary of the literature review considering works that deal with formulations and/or practical applications of the flexible job shop scheduling problem with sequencing flexibility.

The simultaneous optimization of the process plan and the scheduling problem is uncommon in the literature. On the other hand, no model is presented, and none of the additional features of the OPS scheduling problem appear in the mold manufacturing shop application.

In (Kim, Park, and Ko, 2003), the integrated problem considered in (Gan and Lee, 2002) is also addressed, and a symbiotic evolutionary algorithm is proposed. In comparison to the problem addressed in the present work, the problem appears not to be directly related to a practical application, no model is proposed, and none of the additional features of the OPS scheduling problem are considered.

In (Alvarez-Valdés et al., 2005), an FJS scheduling problem with sequencing flexibility issued from the glass industry, is addressed. A heuristic algorithm combining priority rules and local search is proposed. The considered problem includes extensions such as overlapping among operations, operations that can be processed simultaneously, and fixed intervals of unavailability of the machines. However, the problem does not include all the OPS scheduling problem's features, and no model is given.

In (Vilcot and Billaut, 2008), a problem issued from the printing industry, where some operations can be carried out at the same time, is addressed. In the considered

problem, operations' precedence constraints are such that each operation may have one or more predecessor, but it can not have more than one successor, i.e., ganging operations like the ones depicted in Figure 2.6a are not considered. A bi-objective genetic algorithm based on the NSGA II is proposed to tackle the problem. The problem comes from the printing industry, as the OPS scheduling problem, and, in consequence, it includes some of the features of the OPS scheduling problem. However, no model is presented, and a limited type of sequencing flexibility is considered.

In (Lee et al., 2012b), an FJS scheduling problem with AND/OR precedence constraints in the operations is considered. Thus, the problem corresponds to an FJS scheduling problem with sequencing flexibility and process plan flexibility as the one found in (Kim, Park, and Ko, 2003) and in (Gan and Lee, 2002). A MILP model and genetic and tabu search algorithms are proposed. Release times for the jobs are considered, but none of the other additional features of the OPS scheduling problem are addressed.

In (Birgin et al., 2014), a MILP model for the FJS scheduling problem in which precedence constraints between operations are described by an arbitrary directed acyclic graph is introduced, and the model for the FJS scheduling problem introduced in (Özgüven, Özbakır, and Yavuz, 2010) is extended to include sequencing flexibility. Heuristic approaches for the FJS scheduling problem with sequencing flexibility and precedence constraints given by a DAG, proposed in (Birgin et al., 2014), were presented in (Birgin, Ferreira, and Ronconi, 2015) and (Lunardi, Voos, and Cherri, 2019). In (Birgin, Ferreira, and Ronconi, 2015), a list scheduling algorithm and its extension to a beam search method were introduced, while. In (Lunardi, Voos, and Cherri, 2019), a hybrid method that combines an imperialist competitive algorithm and tabu search was proposed. None of the additional features of the OPS scheduling problem are considered in (Birgin et al., 2014; Lunardi, Voos, and Cherri, 2019).

In (Rossi and Lanzetta, 2020), an FJS scheduling problem in the context of additive/subtractive manufacturing, is tackled. Process planning and sequencing flexibility are simultaneously considered. Both features are modeled through a precedence graph with conjunctive and disjunctive arcs and nodes. Numerical experiments with an ant colony optimization procedure aiming to minimize the makespan are presented to validate the proposed approach. Among the features of the OPS scheduling problem, only the sequence-dependent setup time is considered.

In (Vital-Soto, Azab, and Baki, 2020), the minimization of the weighted tardiness and the makespan in an FJS scheduling problem with a sequencing flexibility environment is addressed. A DAG gives precedences between operations as introduced in (Birgin et al., 2014). For this problem, the authors propose a biomimicry hybrid bacterial foraging optimization algorithm hybridized with simulated annealing. The method

makes use of a local search based on the reallocation of critical operations. Numerical experiments with classical instances and a case study are presented to illustrate the performance of the proposed approach. The considered problem does not include any of the additional characteristics of the OPS scheduling problem.

The FJS scheduling problem with sequencing flexibility in which a DAG gives precedences, and that includes sequence-dependent setup times was also considered in (Cao, Lin, and Zhou, to appear). For this problem, a knowledge-based cuckoo search algorithm was introduced that exhibits a self-adaptive parameters control based on reinforcement learning. However, other features such as machines' downtimes and resumable operations are absent in the considered problem.

The scheduling of repairing orders and allocation of workers in an automobile collision repair shop is addressed in (Andrade-Pineda et al., to appear). The underlying scheduling problem is a dual-resource FJS scheduling problem with sequencing flexibility that aims to minimize a combination of makespan and mean tardiness. The authors extend the MILP formulation proposed in (Birgin et al., 2014) and propose a constructive iterated greedy heuristic.

2.3 Summary

The most important contribution of this chapter lies in the identification of a FJS-based scheduling model that adequately addresses the complex requirements of the OPS scheduling problem. Now that the problem has been suitably described, the next step is to select appropriate solution strategies. Commonly used optimization methods for the FJS scheduling problem must be analyzed to support the choice for the optimization strategies to be applied to solve the OPS scheduling problem.

Chapter 3

Optimization Methods for Scheduling

An optimization problem may be defined by $(\mathcal{S}, f)$ where $\mathcal{S}$ is the search space (i.e., the set of feasible solutions) and $f : \mathcal{S} \to \mathbb{R}$ the objective function (also denoted as cost function, evaluation function, or fitness function) to optimize. The objective function assigns to every solution $s \in \mathcal{S}$ of the search space a real number indicating its value. A solution $s^* \in \mathcal{S}$ is *global optimum* if it has a better objective function[1] than all solutions of the search space, i.e., $\forall s \in \mathcal{S}, f(s^*) \leq f(s)$. The main goal in solving an optimization problem is to find a global optimal solution s^*.

Yang (2012) elaborates optimization with an example of searching diamonds in a vast forest (search domain). Looking into the search area, inch-by-inch will consume an enormous amount of energy and time. In such a scenario, an intelligent optimization technique can be applied to focus only on the potential spots, increasing the chance to find diamonds. Sörensen et al., 2012 indicates that every optimization problem comes with few decision variables (e.g., in which order to visit the forest), a certain objective function (e.g., searching maximum number of diamonds), and some constraints (e.g., searching diamond with less time and effort). In this sense, optimization techniques are employed to obtain the values of decision variables that optimize the objective function subject to certain constraints (Hussain et al., 2019).

This chapter presents a summary of the existing optimization strategies for combinatorial optimization problems, concentrating on the most used techniques for scheduling by mainly focusing on the JS scheduling problem and the FJS scheduling problem. This detailed look into the techniques will support the choice for the optimization strategies used in this thesis to model and solve the OPS scheduling problem.

[1] Without loss of generality, a minimization problem. Maximizing an objective function f is equivalent to minimizing $-f$.

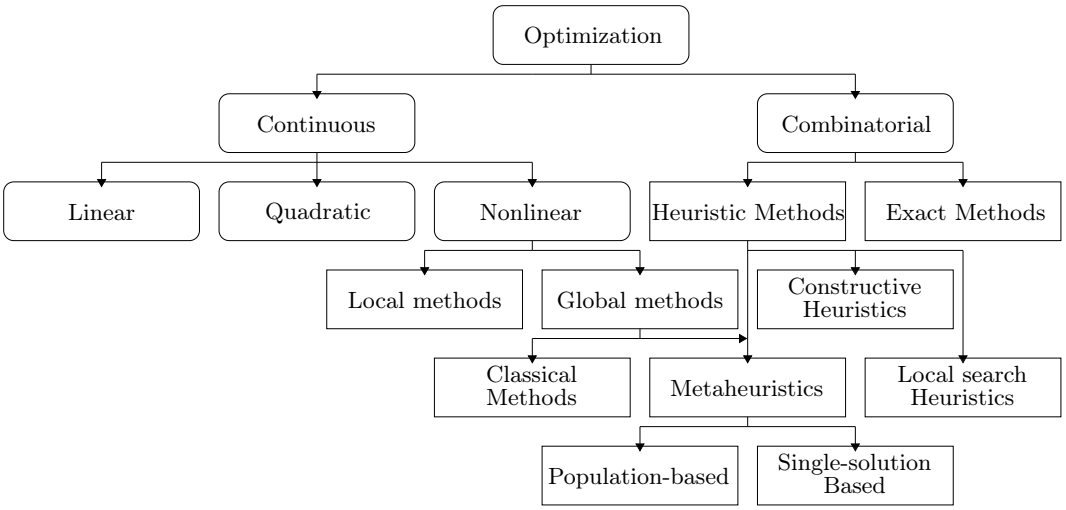


Figure 3.1: A classification diagram of optimization methods.

3.1 Classification of Optimization Methods

According to Sörensen (2015), algorithms for optimization can be roughly divided into two categories: exact algorithms and heuristics. The difference between the two categories is that exact algorithms are designed in such a way that it is guaranteed that they will find the optimal solution in a finite amount of time. Heuristics do not have this guarantee, and therefore generally return solutions that are worse than optimal. Note that there exists an intermediate category, so-called approximation algorithms, in which algorithms guarantee that the solution they find is within a certain percentage of the optimal solution. They can be thought of both as "heuristics with a guarantee" or "not-quite-exact algorithms."

Vitaliy (2006) provides a classification where optimization methods are divided into two broad classes: continuous optimization, where the search area and solutions are presumed to be situated in a continuous space; and combinatorial optimization, where a finite number of feasible solutions limits the search area. Vitaliy (2006) subdivided combinatorial methods into two categories, exact methods, and approximate methods, where the latter includes constructive heuristics, local search heuristics, and metaheuristics.

In this thesis, I use terminology comparable to that used in (Sörensen, 2015), in the sense that whenever I say heuristic methods, I am referring to problem-dependent methods such as constructive heuristics, local search heuristics, and metaheuristic methods. This is due to the fact that the term *approximate methods* may cause confusion with *approximation algorithms* (not-quite-exact algorithms). Figure 3.1 presents — adapted as previously stated — a classification scheme of optimization methods inspired by the one provided in (Vitaliy, 2006, chap. 1).

Technique	Class	# papers	Percentage
Hybrid	Mix of heuristic methods	69	35.03%
Evolutionary algorithm	Population-based	47	23.86%
Constructive algorithm	Heuristic methods	19	9.64%
Tabu search	Single-solution based	12	6.09%
Integer linear programming	Exact Methods	10	5.08%

Table 3.1: Summary of the most widespread techniques applied to solve FJS scheduling problem.

3.2 History of Optimization Methods for Scheduling

The study of optimization methods for scheduling problems dates back to the 1950s. During this time, many problems were solved by the application of crude but efficient heuristics, which formed the basis of the development of classical scheduling theory. During the 1960s, the problems encountered became more sophisticated, and researchers were unable to develop efficient algorithms for them. Most researchers tried to develop efficient Branch and Bound (BB) methods that are essentially exponential-time algorithms. Although these strategies are of tremendous theoretical value providing a significant research accomplishment, as Kan (2012) indicates, "*a natural way to attack scheduling problems is to formulate them as MP models*", the results from many of these works are incredibly disappointing (Jain and Meeran, 1999). With the advent of complexity theory, researchers began to realize that many of these problems may be inherently difficult to solve. In the 1970s, many scheduling problems were shown to be NP-hard. By the end of the 1980s, the attention was concentrated on heuristics methods. From then on, problem-specific heuristics and metaheuristics remain dominant. Nowadays, the most used techniques for optimization of complex scheduling problems are hybrid strategies combining several distinct heuristic methods such as construction heuristics, local search heuristics, and metaheuristics (Chaudhry and Khan, 2016; Amjad et al., 2018).

3.2.1 Flexible Job Shop Scheduling Problem

Chaudhry and Khan (2016) present an extensive review of various techniques that have been employed since 1990 for the FJS scheduling problem. Results show that state-of-the-art methods are hybrid methods combining global and local search heuristic methods, where 59% of the papers proposed hybrid techniques or Evolutionary Algorithms (EAs). Table 3.1 summarizes the findings of (Chaudhry and Khan, 2016)

by presenting the six most widespread techniques applied to solve the FJS scheduling problem and classifying each method accordingly with the classification scheme described in Section 3.1 (see Figure 3.1).

3.3 Mathematical Programming

A commonly used model in MP is Linear Programming (LP). In a LP optimization problem, the objective function to be optimized, and the constraints are linear functions. According to Talbi (2009), LP is one of the most satisfactory models of solving optimization problems. Indeed, for continuous linear optimization problems (i.e., problems whereas the decision variables are real values), efficient exact algorithms such as the simplex-type method or interior-point methods exist. The efficiency of the algorithms is because the feasible region of the problem is a convex set, and the objective function is convex. Then, the global optimum solution is necessarily a node of the polytope representing the feasible region (Talbi, 2009). On the other hand, Nonlinear Programming (NLP) deals with MP problems where the objective function and/or the constraints are nonlinear (Bertsekas, 1997). For NLP optimization models, specific simplex-inspired heuristics, such as the Nelder and Mead algorithm, may be used (Nelder and Mead, 1965). For quadratic and convex continuous problems, some efficient exact algorithms can be used to solve small or moderate problems (Nocedal and Wright, 2006).

Discrete optimization theory in terms of optimization algorithms is less developed than continuous optimization (Talbi, 2009). However, many real-life applications must be modeled with discrete variables, e.g., in many practical optimization problems, the resources are indivisible such as machines and people. In an Integer Programming (IP) optimization model, the decision variables are discrete. When the decision variables are both discrete and continuous, we are dealing with Mixed Integer Programming (MIP). For IP and MIP models, enumerative algorithms such as BB — addressed in Section 3.5.1 — are commonly used. A more general class of IP problems is *combinatorial* optimization problems. This class of problems is characterized by discrete decision variables and finite search space. However, the objective function and constraints may take any form.

3.4 Constraint Programming

Another common approach to model decision and optimization problems is CP, a programming paradigm that integrates more productive modeling tools than the linear expressions of MIP models. A model is composed of a set of variables. Every variable

has a finite domain of values. In the model, symbolic and mathematical constraints related to variables may be expressed. Global constraints represent constraints that refer to a set of variables. Hence, the CP paradigm models the properties of the desired solution. The declarative models in CP are flexible and, in general, more compact than in MIP models (Talbi, 2009).

For example, supose we are dealing with a assignment problem, the goal is to assign n objects $\{o_1, o_2, \ldots, o_n\}$ to n locations $\{l_1, l_2, \ldots, l_n\}$ where each object is placed on a different location. Using constraint techniques, the model will be the following:

$$\text{allDifferent}(y_1, y_2, \ldots, y_n)$$

where y_i represent the index of the location to which the object o_i is assigned. The global constraint allDifferent specifies that all variables must be different. If this problem is modeled using the IP model, one has to introduce the following decision variables:

$$x_{ij} = \begin{cases} 1, & \text{if object } o_i \text{ is assigned to location } l_j \\ 0, & \text{otherwise} \end{cases}$$

Hence, many more variables (n^2 instead of n) are declared. However, it does not mean that solving the problem will be more efficient within constraint programming than using MP. Solving the problem is another story. The efficiency depends mainly on the structure of the target problem and its associated model, and the solving strategy (optimization method) used to solve the model.

3.5 Exact Methods

Developing an efficient exact method is a non-trivial task, even for relatively straightforward problems. Precisely because of their guarantee to find the optimal solution, exact algorithms do not only have to locate this solution in the solution space; they also have to prove that it is optimal. Exact algorithms must exhaustively examine every single solution in the solution space unless they can explicitly determine that it does not need to be examined. Although many techniques exist to do this very efficiently, eliminating large numbers of solutions at each iteration, the fact remains that exact methods do not easily tackle a lot of interesting real-life combinatorial optimization problems. Enumerative techniques are examples of exact methods commonly applied to combinatorial problems, which mainly enumerates all the alternatives and then select the best. The most widely used enumerative strategy is the BB algorithm (Land and Doig, 2010), which was, for many years, the most popular technique to solve the JS scheduling problem (Jain and Meeran, 1999).

3.5.1 Branch and Bound

The BB algorithm implicitly searches a dynamically constructed tree representing the solution space of all feasible solutions. This technique formulates procedures and rules to allow large portions of the tree to be removed from the search. According to Burke and Kendall (2014), the rationale behind the BB algorithm is to reduce the number of alternatives that need to be considered by repeatedly partitioning the problem into a set of smaller subproblems and using local information in the form of bounds to eliminate those that can be shown to be sub-optima, i.e., the BB process allows to prove that some partial solutions cannot lead to optimal solutions and to cut them and their descendants from the search tree. This process is known as *pruning*.

Even though several procedures have been developed to exclude large portions of the tree to speed up the search process, the fact that the size of the tree grows explosively with problem size is still a problem. For most real-life problems, the complete enumeration is not possible, e.g., the complete enumeration of all feasible allocations of n operations to n machines would result in a tree with $n!$ terminal nodes, i.e., $\sim 2.4 \times 10^{18}$ terminal nodes for $n = 20$.

3.6 Heuristic Methods

Because of the impracticality of exhaustive search, however cleverly done, heuristics have been developed for challenging problems throughout human history, i.e., people have always been creating simple mental shortcuts to make decisions. Heuristic methods usually find reasonably "good" solutions in a reasonable amount of time. In general, heuristic methods may be classified into two families: *problem-specific heuristics* (see Section 3.7), and *metaheuristics* (Section 3.8).

3.6.1 Heuristic Principles

Early research on heuristics often focused on human intuition as a way of solving problems. As indicated in (Sörensen, 2015), although slightly optimistic about the future, Simon and Newell (1958) propose that a theory of heuristic (as opposed to algorithmic or exact) problem-solving should focus on intuition, insight, and learning. Many modern heuristic algorithms attempt to gain a deep insight into the structure of the problem that is to be solved. George Plóya's influential book "How to Solve It?" (Ploya, 1945), put forward some heuristic principles which are actively being used by heuristics researchers today to develop effective algorithms. The principle of *analogy* tells the heuristic designer to look for analogous problems for which a solution

technique is known and use this information to help solve the problem at hand. The principle of *induction* instructs to solve a problem by deriving a generalization from some examples. The *auxiliary problem* idea asks whether a subproblem exists that can help to solve the overall problem.

These principles are well-known to a designer of heuristics, who will look for similar problems in the literature and examine the best-known methods for them (analogy), who will try to solve some simple examples by hand and derive an intelligent solution strategy from this (induction), or who will try to decompose the problem into, e.g., a master problem and a subproblem and develop specialized techniques for both (auxiliary problem). The use of these techniques forces the heuristic designer to think about the optimization problem that is being faced and provides guidelines to come up with the most appropriate strategy.

3.7 Problem-Specific Heuristics

Problem-specific heuristic, as the name suggests, are methods tailored and design methods to solve a specific problem. Constructive heuristics are a perfect example of problem-specific heuristic methods. Constructive heuristics refer to the process of building an initial solution of a specific problem from scratch by joining together "pieces", or components, that are added one after the other until a solution is complete (Bianchi et al., 2009). Constructive heuristics are usually thought of as being fast because they often represent a single-pass approach.

Dispatching rules are examples of the constructive heuristics built to provide reasonable solutions to complex problems in real-time (Jones, Rabelo, and Sharawi, 2001). A dispatching rule is a rule that defines a priority to the jobs that are not sequenced. The dispatching rule inspects the not sequenced jobs and selects the job with the highest priority. Note that the priority is given by the rule being used. For example, the Shortest Processing Time First (SPT) dispatching rule prioritizes the job with the shortest processing time.

3.7.1 Local Search Heuristics

Local search heuristics also refereed to as improvement heuristics, start with a single solution — called *current solution* — which is improved at each iteration by modifying some of its components. These changes are called *moves*. Depending on the way the solution is represented, different move *types* can be defined. Each move type gives rise to a *neighborhood structure*.

Algorithm 3.7.1 Local search algorithm

```
1: function LocalSearch(s)                          ▷ s is the initial solution
2:     while exists a solution s' ∈ N(s) where f(s') < f(s) do
3:         for each s' ∈ N(s) do
4:             s ← argmin {f(s), f(s')}
5:     return s
```

Given a neighborhood structure, the set of solutions $\mathcal{N}(s)$ that can be obtained by performing a move to a given solution s is called *neighborhood* of s. At each iteration of the local search algorithm, the current solution is replaced by the best neighbor solution, where the best neighbor is defined according to a rule, called *move strategy* or *search strategy*. Let $f(s)$ be the objective function that assess the quality of solution s. Assuming that we are dealing with a minimization problem where the search strategy is to always move to the neighbor with smaller f, if $f(s) \leq f(s')$ for all $s' \in N(s)$, then s is called *local optimum* (as opposed to a *global optimum*, i.e., a best possible solution to the optimization problem). The local search procedure stops when a local optimum is found. Algorithm 3.7.1 summarizes the essential steps of a local search algorithm where the goal is to minimize $f(x)$.

According to Sörensen and Glover (2013), a typical search strategy is the steepest descent or steepest ascent strategy, in which the best move from the neighborhood is selected. Methods that use this strategy are often called hill-climbers, e.g., Algorithm 3.7.1. Other move strategies include the mildest ascent/descent, also called first improving, strategy, in which the first move is selected that improves the current solution. Selecting a random improving solution is another commonly used move strategy.

3.8 Metaheuristics

The change in perception that has put heuristics on equal footing with exact methods as a field of research coincides in time with the advent of metaheuristics. The term was first coined in (Glover, 1986), but a concrete definition has been elusive. Sörensen (2015) defined it as follows:

> "A *metaheuristic* is a high-level problem-independent algorithmic *framework* that provides a set of guidelines or strategies to develop heuristic optimization algorithms. The term is also used to refer to a problem-specific implementation of a heuristic optimization algorithm according to the guidelines expressed in such a framework." Sörensen, 2015.

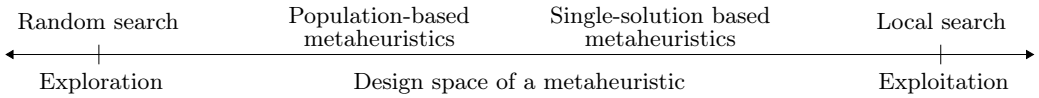


Figure 3.2: Two conflicting criteria in designing a metaheuristic: exploration (diversification) versus exploitation (intensification) (Talbi, 2009).

According to Talbi (2009), during the design of a metaheuristic, two contradictory criteria must be taken into account: *exploration* of the search space (diversification) and *exploitation* of the best solutions found (intensification). In intensification, promising regions are explored more thoroughly in the hope of finding better solutions. In diversification, non-explored regions must be visited to be sure that all regions of the search space are evenly explored, e.g., random restart is extreme in terms of exploration, while the local search is extreme in terms of exploitation. This situation is illustrated in Figure 3.2. In general, basic single-solution based metaheuristics are more exploitation oriented, whereas basic population-based metaheuristics are more exploration oriented.

3.8.1 Representation

Designing any metaheuristics require an *encoding* (representation) of a solution. The representation scheme is a problem-dependent structure that stores all the essential information (the value of the decision variables) of a solution. The encoding plays a major role in the efficiency and effectiveness of any metaheuristic. The efficiency of a representation is also related to the search operators applied on this representatio, e.g., manipulation of the components of a solution. According to Talbi (2009), one of the main characteristics of a representation is its *completness*, i.e., all solutions associated with the problem must be represented.

Many straightforward linear encodings may be applied for some traditional families of optimization problems:

- *Binary encoding*, 0/1 integer programming problem (Puchinger and Raidl, 2005), knapsak problem (Chu and Beasley, 1998), satisfiability problem (Gu et al., 1999);
- *Vector of discrete values*, assignment problem (Salman, Ahmad, and Al-Madani, 2002), facility location problem (Shen, Zhan, and Zhang, 2011), load balancing problem (Sim and Sun, 2003);
- *Vector of real values*, continuous optimization (Yang, 2013), parameter identification (Wu, Wang, and You, 2010);

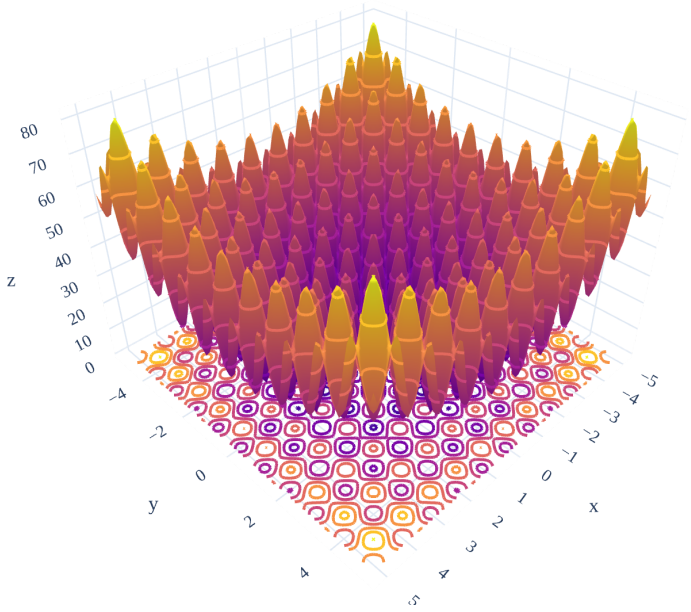


Figure 3.3: Rastrigin function with two dimensions, i.e., $x = x_1$, $y = x_2$, and $z = f(x_1, x_2)$.

- *Permutation*, sequencing problems (Pezzella, Morganti, and Ciaschetti, 2008), travelling salesman problem (Larranaga et al., 1999).

An example of problem that can be directly represented by a vector of real values is given as follows. Consider we are dealing with a continuous optimization problem where the goal is to minimize a d-dimensional non-convex function

$$f(x_1, x_2, \ldots, x_d) = Ad + \sum_{i=1}^{d} [x_i^2 - A\cos 2\pi x_i]$$

where $A = 10$ and $x_i \in [-5.12, 5.12]$. Note this function is the well-known Rastrigin function (see Figure 3.3). In this case, it would be enough to say that solutions are directly represented by a d-dimensional real vector $\vec{x} \in \mathbb{R}^d$, where $\mathbb{R}^d$ is the set of all d-tuples of real numbers in $[-5.12, 5.12]$, i.e., $\vec{x} = (x_i \in [-5.12, 5.12] : i \in \{1, 2, \ldots, d\})$.

3.8.1.1 Indirect Encodings

When using an indirect representation, the encoding is not a complete solution for the problem. A *decoder* is required to express the solution given by the encoding. Indirect encodings are popular in optimization problems dealing with many constraints, such as scheduling problems. For instance, the constraints associated with the optimization problem are handled by the decoder and will guarantee the validity of the solution that is derived.

3.8.1.2 Representation Decoding

The design questions related to the definition of the representation and the objective function may be related. In some problems, the representation (genotype) is decoded to generate the best possible solution (phenotype). In this situation, the mapping between the representation and the objective function is not straightforward; that is, a decoder function must be specified to generate from a given representation the best solution according to the objective function.

3.8.1.3 Representation Definition

In this thesis, an "universal" indirect representation is used, i.e., universal in the sense it adapts given the bounds of the decision variables associated with the optimization problem at hand. Let $f(x)$ be a d-dimensional function to be optimized, i.e., $x = (x_1, x_2, \ldots, x_d)$ such that $\forall i \in \{1, 2, \ldots, d\}, x_i \in [x_i^{\text{lb}}, x_i^{\text{ub}}]$ where x_i^{lb} and x_i^{ub} are respectively the lower and upper bound for x_i. The indirect representation is given by a d-dimensional real vector $\vec{x} \in \mathbb{R}^d$ where each component is a real number in $[0, 1)$, i.e., $\vec{x} = (x_i \in [0, 1) : i \in \{1, 2, \ldots, d\})$. Then a decoder is a linear function $d : [0, 1) \to [x_i^{\text{lb}}, x_i^{\text{ub}}]$ such that $d(0) = x_i^{\text{lb}}$ and $d(1) = x_i^{\text{ub}}$. In this way, to assess the objective value of $\vec{x}$ we have to decode it into $\vec{y}$ first, i.e., given $\vec{x}$ we have that $\vec{y} = (y_1, y_2, \ldots, y_d)$ where $\forall i \in \{1, 2, \ldots, d\}, y_i = d(x_i)$.

3.8.2 Single-Solution Based Metaheuristics

Despite their similarities with local search heuristics, single-solution metaheuristics also known as neighborhood metaheuristics, are global optimization algorithms, i.e., eventually finds the global optimum if we run it long enough. The global capability is achieved through a *escape strategy* that allows exploring neighbor solutions beyond local optimality. It is this strategy that characterizes a single-solution metaheuristic, and usually, the name of the metaheuristic is derived from it. Algorithms such as Simulated Annealing (SA), ILS, TS, and Variable Neighborhood Search (VNS) belong to this class of metaheuristics.

According to Sörensen and Glover (2013), two simple — but commonly used — escape strategies are the Random Restart (RR), which restarts the search from a new random solution (see Section 3.8.1), or the ILS, which applies a large random change (called *perturbation*) to the current solution (see Section 3.8.2.2). In contrast with memoryless designs that implement a form of sampling that heavily rely on semi-random processes (e.g., RR and ILS), TS uses adaptive memory to explore solution space beyond local optimality (see Section 3.8.2.3).

Algorithm 3.8.1 Random-restart local search algorithm

```
1: function RandomRestartLocalSearch
2:     Initialize s^best as a randomly constructed solution
3:     while stopping criterion is not satisfied do
4:         s ← a randomly constructed solution
5:         Perform local search Algorithm 3.7.1 starting from s to obtain s'
6:         s^best ← argmin {f(s^best), f(s')}
7:     return s^best
```

3.8.2.1 Random Restart Local Search

According to Lourenço, Martin, and Stützle (2003), the most straightforward escape strategy to improve upon a cost found by a local search is to repeat the search from another starting point. Although such a "random restart" approach with independent samplings is sometimes a useful strategy (in particular when all other options fail), it breaks down as the instance size grows. Algorithm 3.8.1 presents the essential steps of a RR local search algorithm, where the goal is to minimize $f(x)$.

3.8.2.2 Iterated Local Search

ILS can be seen as an extension — a more clever version — of RR local search algorithm. Instead of losing all the information obtained so far and restarting the search procedure to a random solution, ILS uses a perturbation procedure that modifies some of the components of the current local optimum. The perturbation strength $\hat{p}$ must be sufficiently strong to allow the local search to explore new search spaces, but also weak enough so that not all the good information gained in the previous search is lost. For example, considering the representation given by a vector of real values provided in Section 3.8.1.3, one possible perturbation procedure could be defined as follows. Let the perturbation strength $\hat{p} \in \{1, 2, \ldots, d\}$ denotes the number of components of $\vec{x}$ that are going to be associated with new randomly generated values. Let $\mathcal{S} \subseteq \{1, 2, \ldots, d\}$ be the set with the selected indexes of $\vec{x}$. Then, for each $i \in \mathcal{S}$ we have that $x_i = r_i$ where $r_i \in [0, 1)$ is a random number generated using uniform distribution.

In this way, starting with a local optimum as the current solution, a perturbed solution is created, and local search is performed on it. Greedily, if the new local optimum is better than the current, then the new local optimum replaces the current local optimum. Otherwise, the current local optimum is preserved (Lourenço, Martin, and Stützle, 2003). Algorithm 3.8.2 summarizes the essential steps of the ILS algorithm, where the goal is to minimize $f(x)$.

Algorithm 3.8.2 Iterated local search algorithm

1: **function** IteratedLocalSearch($\hat{p}$)
2: Initialize s^{best} and s^{pert} as a randomly constructed solutions
3: **while** stopping criterion is not satisfied **do**
4: Perform local search Algorithm 3.7.1 starting from s^{pert} to obtain s
5: $s^{\text{best}} \leftarrow \operatorname{argmin}\{f(s^{\text{best}}), f(s)\}$
6: $s^{\text{pert}} \leftarrow$ perturbates s^{best} with strength $\hat{p}$
7: **return** s^{best}

3.8.2.3 Tabu Search

TS — preceded by GA — is the second most used optimization technique for the FJS scheduling problem. Introduced by Glover (1986), TS is based on the premise that problem solving, in order to qualify as intelligent, must incorporate adaptive memory and responsive exploration. The adaptive memory feature of TS allows the implementation of procedures that are capable of searching the solution space economically and effectively. Since local choices are guided by information collected during the search, TS contrasts with memory-less designs that heavily rely on semi-random processes that implement a form of sampling. The emphasis on responsive exploration (and hence purpose) in TS, whether in a deterministic or probabilistic implementation, derives from the supposition that a wrong strategic choice can often yield more information than an excellent random choice, and therefore provides a basis for progressively improved strategies that take advantage of search history.

According to Glover and Laguna (1998), TS permits moves that deteriorate the current objective function value and selects the moves from a modified neighborhood $\mathcal{N}^*(s)$. The memory structure — so-called tabu list — is responsible for the specific composition of $\mathcal{N}^*(s)$. In other words, the modified neighborhood is the result of maintaining a selective history of the states encountered during the search. In TS, $\mathcal{N}^*(s)$ characteristically is a subset of $\mathcal{N}(s)$, and the tabu classification serves to identify elements of $\mathcal{N}(s)$ excluded from $\mathcal{N}^*(s)$. Characterized in this way, TS may be viewed as a dynamic neighborhood method. This means that the neighborhood of s is not a static set, but rather a set that can change according to the history of the search. Algorithm 3.8.3 summarizes the essential steps of the TS algorithm, where the goal is to minimize $f(x)$.

3.8.3 Population-Based Metaheuristics

Population-based methods differ from the previous methods in the sense that they keep a set of solutions — a population — rather than a single solution. From now on, the notion of population is defined as follows. Let a population $\mathcal{P} = (\vec{x}^{\,1}, \vec{x}^{\,2}, \ldots, \vec{x}^{\,n_{\text{size}}})$ be

Algorithm 3.8.3 Tabu search algorithm

```
1:  function TabuSearch
2:      Initialize solutions s and s^best as randomly constructed solutions
3:      Initialize tabu list
4:      while stopping criterion is not satisfied do
5:          if solution s' ∈ N(s) meet aspiration criterion then
6:              s ← s' and update tabu list
7:          else
8:              Filter neighbors in N(s) based on the the tabu list to obtain N*(s).
9:              Let s* be the best neighbor in N*(s)
10:             s ← s* and update tabu list
11:         s^best ← argmin {f(s^best), f(s)}
12:     return s^best
```

a sequence of solutions where $\forall i \in \{1, 2, \ldots, n_{\text{size}}\}, \vec{x}^{\,i}$ is a solution (see Section 3.8.1.3), and $\forall j \in \{1, 2, \ldots, d\}, \vec{x}^{\,i}_j$ is the j-th component.

A general framework of a population-based metaheuristic is introduced in (Talbi, 2009). First, the population is initialized. The determination of the initial population plays a crucial role in its effectiveness and efficiency. In the generation of the initial population, the main criterion to deal with is diversification. If the initial population is not well-diversified, a premature convergence can occur. Once the initial population is initialized, a new population of solutions is generated. Finally, this new population is integrated into the current one using some selection procedures. The search process is stopped when a given condition is satisfied. Algorithms such as EA (Bäck, Fogel, and Michalewicz, 1997) and Swarm Intelligence (SI) (Kennedy, 2006, chap. 6) belong to this class of metaheuristics. EAs are optimization algorithms inspired by the evolutionary metaphor. Algorithms that belongs to the SI class are inspired by the social behavior of species that compete for foods.

3.8.3.1 Genetic Algorithm

GA belongs to the EAs class, and its development was inspired through the process of natural genetic evolution (Holland, 1992). Charles Darwin contributed the original work on natural evolution published on 1859 (Darwin and Bynum, 2009), in which he claimed that natural populations evolve according to the process of natural selection based on "survival of the fittest" rule. GA mimics the natural process of genetic evolution on the principle of survival of the fittest to obtain better solutions to engineering problems. GA usually applies a crossover (recombination) operator to two solutions, plus a mutation operator that randomly modifies the individual contents to promote diversity. The replacement (survivor selection) is generational; that is, the parents are replaced systematically by the offsprings. Fixed probabilities p_c (crossover

Parents

j	1	2	3	4	5	6	7	8	9	10
$\vec{x}_j^{'}$	0	1	1	0	1	0	0	1	1	0
$\vec{x}_j^{''}$	1	0	0	0	1	1	1	0	1	1

$\rightarrow$

Offsprings

j	1	2	3	4	5	6	7	8	9	10
$\vec{x}_j^{'}$	0	1	1	0	1	0	1	0	1	1
$\vec{x}_j^{''}$	1	0	0	0	1	1	0	1	1	0

(a) 1-point crossover operator

Parents

j	1	2	3	4	5	6	7	8	9	10
r_j	0.3	0.9	0.7	0.4	0.1	0.5	0.6	0.7	0.2	0.6
$\vec{x}_j^{'}$	0	1	1	0	1	0	0	1	1	0
$\vec{x}_j^{''}$	1	0	0	0	1	1	1	0	1	1

$\rightarrow$

Offsprings

j	1	2	3	4	5	6	7	8	9	10
r_j	0.3	0.9	0.7	0.4	0.1	0.5	0.6	0.7	0.2	0.6
$\vec{x}_j^{'}$	0	0	0	0	1	0	1	0	1	1
$\vec{x}_j^{''}$	1	1	1	0	1	1	0	1	1	0

(b) Uniform crossover operator

Figure 3.4: Crossover operators applied to binary representation.

probability) and p_m (mutation probability) are applied to the crossover and mutation operators, respectively.

A selection mechanism is used to select the parents for crossover. The main principle of selection methods is "the better is an individual, the higher is its chance of being parent". *Roulette wheel* selection is the most common selection strategy. It will assign to each individual a selection probability that is proportional to its relative fitness. In the roulette wheel selection, outstanding individuals will introduce bias at the beginning of the search that may cause a premature convergence and a loss of diversity. As an alternative to this problem, *tournament* selection consists in randomly selecting k individuals, where k is the size of the tournament group. A tournament is then applied to the k members of the group to select the best one. The tournament procedure is then carried out two times to select two parents.

Once the selection of individuals to form the parents is performed, the role of the *reproduction* phase is the application of variation operators such as the crossover and mutation. The crossover procedure requires two selected individuals. The role of the crossover operator is to inherit some characteristics of the two parents to generate the offsprings. Many distinct crossover operators have been proposed. For linear representations excluding permutations, the well-known crossover operators are the *1-point* crossover, its generalization the *n-point* crossover, and the *uniform* crossover. The 1-point crossover, a position in the chromosome, is randomly selected, and two offsprings are created by interchanging the segments of the parents (see Figure 3.4.a). In the *n-point* crossover, the same procedure is performed, but instead of 1, n positions of the crossover are randomly selected. In the uniform crossover operator, each component of the offspring is selected randomly from either parent, i.e., each parent

Algorithm 3.8.4 Genetic Algorithm

function GeneticAlgorithm($n_{\text{size}}, p_c, p_{\text{mut}}$)
  Let $\mathcal{P}$ be the initial population and $\vec{x}^{\,\text{best}}$ be a randomly constructed solution
  while stopping criterion is not satisfied **do**
    Let $\mathcal{Q} \leftarrow \emptyset$ be the population for the next iteration
    for 1 to $n_{\text{size}}/2$ **do**
      Select parents $\vec{x}^{\,'}$ and $\vec{x}^{\,''}$ from $\mathcal{P}$ with probability p_c
      Perform crossover on parents to generate $\underline{\vec{x}}^{\,'}$ and $\underline{\vec{x}}^{\,''}$
      Mutate each offspring with probability p_m
      $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\underline{\vec{x}}^{\,'}, \underline{\vec{x}}^{\,''}\}$
      $s^{\text{best}} \leftarrow \text{argmin}\left\{f(s^{\text{best}}), f(\underline{\vec{x}}^{\,'}), f(\underline{\vec{x}}^{\,''})\right\}$
    Let $\mathcal{P} \leftarrow \mathcal{Q}$
  return $\vec{x}^{\,\text{best}}$

will contribute equally to generate the offsprings. This can be performed by generating d random numbers, i.e., $\forall i \in \{1, 2, \ldots, d\}$ we have that $r_i \in [0, 1)$ is a random number. Then, if $r_i \leq 0.5$, the i-th offspring's component is associated with the value from the i-th component of parent 1, otherwise, it is associated with the value of the parent 2 (see Figure 3.4.b). Additionally to the 1-point, n-point, uniform crossover, several crossover operators have been proposed for real-valued representations such as intermediate crossover, geometrical crossover, unimodal normal distribution crossover, simplex crossover, simulated binary crossover (Deb and Agrawal, 1995), and parent-centric crossover.

Unlike crossover operators, mutation operators act on a single individual. Mutations represent small changes of selected individuals of the population. The probability p_m defines the probability to mutate each component of the representation. It could also affect only one gene too. Traditional mutation operators for linear representations are: the *flip* operator for binary representations, and *swap* operators for permutations representations. For real-valued representations the most common are the *uniform*, *Gausian*, and *polynomial* mutation (Deb and Deb, 2014).

Algorithm 3.8.4 summarizes the essential steps of the genetic algorithm, where the goal is to minimize $f(x)$.

3.8.3.2 Differential Evolution

The DE algorithm (Storn and Price, 1997) was developed from the attempts to solve the Chebychev's polynomial fitting problems. DE was inspired by the idea of using vector differences for perturbing the vector population, i.e., it disturbs the current population members with a scaled difference of indiscriminately preferred and dissimilar population members. Since 1996, when DE achieved excellent results in an

Algorithm 3.8.5 Differential Evolution

function DIFFERENTIALEVOLUTION($n_{\text{size}}, \zeta, p_c$)
  for $i \leftarrow 1$ to n_{size} **do**
    $\vec{x}^{\,i} \leftarrow$ a randomly constructed solution.
  while stopping criterion is not satisfied **do**
    for $i \leftarrow 1$ to n_{size} **do**
      Compute random numbers $r_1 \neq r_2 \neq r_3 \in \{1, 2, ..., n_{\text{size}}\} \setminus \{i\}$.
      Compute $\vec{v} \leftarrow \max\left\{0, \min\left\{\vec{x}^{\,r_1} + \zeta(\vec{x}^{\,r_2} - \vec{x}^{\,r_3}), 1 - 10^{-16}\right\}\right\}$.
      Compute a random number $R(i) \in \{1, \ldots, d\}$.
      for $j \leftarrow 1$ to d **do**
        Compute a random number $\gamma \in [0, 1]$.
        if $\gamma \leq p_c$ or $j = R(i)$ **then**
          $\vec{u}^{\,i}_j \leftarrow \vec{v}^{\,i}_j$
        else
          $\vec{u}^{\,i}_j \leftarrow \vec{x}^{\,i}_j$
      if $f(\vec{u}^{\,i}) \leq f(\vec{x}^{\,i})$ **then**
        $\vec{x}^{\,i} \leftarrow \vec{u}^{\,i}$
        $\vec{x}^{\,\text{best}} \leftarrow \text{argmin}\{f(\vec{x}^{\,\text{best}}), f(\vec{u}^{\,i})\}$
  return s^{best}

international optimization contest, it has been applied to a wide variety of problems showing good results. The success of DE can be mainly attributed to the fact that it is simple, and at the same time, effective. It can also be easily adapted for running on a parallel processing system.

In the basic variant of the DE, at each iteration, for each solution $\vec{x}^{\,i}$ ($i = 1, 2, \ldots, n_{\text{size}}$), a mutant $\vec{v}^{\,i}$ is generated according to

$$\vec{v}^{\,i} = \vec{x}^{\,r_1} + \zeta(\vec{x}^{\,r_2} - \vec{x}^{\,r_3}) \tag{3.1}$$

where ζ is a parameter in $(0, 2]$, usually less than or equal to 1, and $r_1, r_2, r_3 \in \{1, 2, \ldots, n_{\text{size}}\} \setminus \{i\}$ are random indices. Note that we must have $n_{\text{size}} \geq 4$, since r_1, r_2, r_3 and i must be mutually different. The parameter ζ controls the amplifications of the differential variation. The basic DE variant with the mutation scheme given by (3.1) is named DE/rand/1. The second most used DE variant, denoted DE/best/1 (see Qin, Huang, and Suganthan, 2008), is also based in (3.1) but

$$r_1 = \underset{i=1,\ldots,n_{\text{size}}}{\text{argmin}} \left\{f(\vec{x}^{\,i})\right\},$$

i.e., $\vec{x}^{\,r_1}$ is the individual with the best fitness value in the population, and $r_2, r_3 \in \{1, 2, \ldots, n_{\text{size}}\} \setminus \{i, r_1\}$ are random indices.

Once the mutant $\vec{v}^i$ is generated, a trial $\vec{u}^i$ is formed as follows

$$\vec{u}^i_j = \begin{cases} \vec{v}^i_j & \text{if a random value in } [0,1] \text{ is less than or equal to } p_c \text{ or if } j = R(i), \\ \vec{x}^i_j & \text{otherwise.} \end{cases}$$

where $p_c \in [0,1]$ is a given parameter and $R(i)$ is a randomly chosen index in $\{1,2,\ldots,d\}$, which ensures that at least one element of $\vec{v}^i$ is passed to $\vec{u}^i$. To decide whether $\vec{u}^i$ should become a member of the next generation or not, it is compared to $\vec{x}^i$ using a greedy criterion. If $f(\vec{u}^i) \leq f(\vec{x}^i)$, then $\vec{u}^i$ substitutes $\vec{x}^i$; otherwise, $\vec{x}^i$ is retained. Algorithm 3.8.5 summarizes the essential steps of the DE/rand/1 variant of the DE algorithm, with the goal to minimize $f(x)$.

3.9 Summary

This chapter provides a summary of the widespread techniques applied to solve FJS scheduling problem, and the most successful techniques were described. DE, GA, ILS, and TS were selected as the solution strategies of choice. A brief introduction to each one of these optimization methods was provided. In the next chapter, the OPS scheduling problem formulation is presented in a precise and universal language (MILP formulation). Since the problem is NP-hard, it is well-known that only small-sized instances can be solved with a certificate of optimality when the MILP formulation is tackled with an exact commercial solver, a CP formulation is presented.

Chapter 4

Mathematical Programming and Constraint Programming Models

In this chapter the MILP formulation and CP formulation of the OPS scheduling problem are presented. Since the OPS scheduling problem is NP-hard, it is well-known that only small-sized instances can be solved with a certificate of optimality when the MILP formulation is tackled with an exact commercial solver. Numerical experiments show that this is exactly the case when the MILP formulation is tackled with an exact commercial solver (IBM ILOG CPLEX Optimization Studio, 2020). On the other hand, the commercial solver IBM ILOG CP Optimizer (Laborie et al., 2018), that applies to CP formulations, is presented in the literature (Vilím, Laborie, and Shaw, 2015) as a half-heuristic-half-exact method that, when it is not able to produce a solution with a guarantee of optimality, it produces a good quality solution in a reasonable amount of time. Experiments are performd to verify the size of the instances that IBM ILOG CP Optimizer can solve to optimality in comparison to IBM ILOG CPLEX. As a side effect, both models can be validated and analyzed comparatively; and a benchmark set with known optimal solutions can be built. Moreover, experiments are performed to analyze the capability of IBM ILOG CP Optimizer to find good quality solutions when applied to large-sized instances that are of the size of the instances of the OPS scheduling problem that appear in practice. As a whole, this chapter provide in-depth knowledge of the OPS scheduling problem that leverages the development of ad-hoc heuristic methods to be applied in practice, e.g., the heuristic methods proposed in Chapter 5.

4.1 Mixed Integer Linear Programming Formulation

The FJS scheduling problem with sequencing flexibility considered in (Birgin et al., 2014) generalizes the classical FJS scheduling problem in the sense that the linear order of the operations of a job is replaced by arbitrary precedence constraints given by a DAG. The OPS scheduling problem considered in the present work generalizes problem considered in (Birgin et al., 2014) in three relevant ways:

(a) The concept of precedence among operations of a job is redefined allowing some overlapping;

(b) Operations are resumable since their processing may be interrupted by periods of unavailability of the machines;

(c) Sequence-dependent setup times must be considered.

Other than that, operations' release times are considered; and some operations may have pre-defined machines and starting times for execution.

Note that the concept of "job" is indirectly addressed in the considered problem. It is assumed that a job is composed of a set of operations and that precedence relations may exist between operations belonging to the same job; while operations belonging to different jobs have no precedence relations among them. Once the set of precedence relations has been defined (as the union of the precedence relations of each individual job), the number n of jobs plays no explicit role in the problem definition anymore. The problem description and problem notations used in this chapter are introduced in the Section 2.2.1. Table 2.4 summarizes the constants and sets that define an instance of the OPS scheduling problem. Table 4.1 summarizes the variables used in the MILP formulations presented in this chapter.

Ignoring the three relevant generalizations listed at the beginning of the current section and named **(a)**, **(b)**, and **(c)**, the OPS scheduling problem can be modeled as a mixed integer linear programming with minor modifications to the MILP model for the FJS scheduling problem with sequencing flexibility in (Birgin et al., 2014). The model considers binary variables x_{ik} ($i \in V$, $k \in F(i)$) to indicate whether operation i is assigned to machine k ($x_{ik} = 1$) or not ($x_{ik} = 0$). It also considers binary variables y_{ij} ($i, j \in V$, $i \neq j$, $F(i) \cap F(j) \neq \emptyset$) to indicate, whenever two operations are assigned to the same machine, which one is processed first. If i is processed before j, then $y_{ij} = 1$ and $y_{ji} = 0$; while if j is processed before i, then $y_{ji} = 1$ and $y_{ij} = 0$. Finally, the model uses real variables s_i ($i \in V$) to denote the starting time of operation i and a variable $C_{\max}$ to represent the makespan. With these variables, the modified MILP

Table 4.1: Summary of the MILP model's variables of the OPS scheduling problem.

Section	Symbol	Interpretation
§4.1	$x_{ik} \in \{0,1\}$, $i \in V$, $k \in F(i)$	$x_{ik} = 1$ if and only if operation i is assigned to machine k
	$y_{ij} \in \{0,1\}$, $i,j \in V$, $i \neq j$, $F(i) \cap F(j) \neq \emptyset$	$y_{ij} = 1$ if operations i and j are assigned to the same machine and i is processed before j
	p'_i, $i \in V$	processing time of operation i (its value depends on the machine to which operation i is assigned)
	s_i, $i \in V$	starting time of operation i (it must coincide with $\bar{s}_i$ if $i \in T$)
	$C_{\max}$	makespan
§4.1.1	$\bar{p}'_i$, $i \in V$	processing time of operation i that must be completed before an operation j can start to be processed if $(i,j) \in A$ (its value coincides with $\lceil \theta_i p'_i \rceil$)
§4.1.2	c_i, $i \in V$	completion time of operation i
	$\bar{c}_i$, $i \in V$	instant at which $\bar{p}'_i$ units of time of operation i has been processed
	$v_{ik\ell}$, $w_{ik\ell}$, $\bar{w}_{ik\ell} \in \{0,1\}$, $i \in V$, $k \in F(i)$, $\ell \in \{1,\ldots,q_k\}$	auxiliar variables to model constraints on s_i, c_i, and $\bar{c}_i$ with respect to the machines' unavailability periods
	u_i, $i \in V$	amount of time the machine $k(i)$ to which operation i is assigned is unavailable between s_i and c_i
	$\bar{u}_i$, $i \in V$	amount of time the machine $k(i)$ to which operation i is assigned is unavailable between s_i and $\bar{c}_i$
§4.1.3	$y_{ijk} \in \{0,1\}$, $i,j \in V$, $i \neq j$, $k \in F(i) \cap F(j)$	$y_{ijk} = 1$ if and only if operations i and j are assigned to machine k and i is the immediate predecessor of j (these variables substitute variables y_{ij} from §2)
	$\hat{\xi}_{ik}$, $\bar{\xi}_{ik}$, $i \in V$, $k \in F(j)$	auxiliar variables to model the sequence-dependent setup time
	ξ_i, $i \in V$	sequence-dependent setup time of operation i

model can be written as follows:

$$\text{Minimize } C_{\max} \tag{4.1}$$

subject to

$$\sum_{k \in F(i)} x_{ik} = 1, \quad i \in V, \tag{4.2}$$

$$p'_i = \sum_{k \in F(i)} x_{ik} p_{ik}, \quad i \in V, \tag{4.3}$$

$$s_i + p'_i \leq C_{\max}, \quad i \in V, \tag{4.4}$$

$$s_i + p'_i \leq s_j, \quad (i,j) \in A, \tag{4.5}$$

$$y_{ij} + y_{ji} \geq x_{ik} + x_{jk} - 1, \quad i,j \in V,\ i \neq j,\ k \in F(i) \cap F(j), \tag{4.6}$$

$$s_i + p'_i - (1 - y_{ij})M \leq s_j, \quad i,j \in V,\ i \neq j,\ F(i) \cap F(j) \neq \emptyset, \tag{4.7}$$

$$s_i \geq r_i, \quad i \in V, \tag{4.8}$$

$$s_i = \bar{s}_i, \quad i \in T. \tag{4.9}$$

The objective function (4.1) minimizes the makespan $C_{\max}$. Constraints (4.2) say that each operation must be assigned to exactly one machine; and constraints (4.3) define p'_i as the processing time of operation i (that depends on the machine to which it was assigned.) Constraints (4.4) ensure that every operation is not completed later than $C_{\max}$. Constraints (4.5) represent the classical meaning of a precedence constraint, saying that, independently of the machine by which they are being processed, if operation i precedes operation j, then i must be completed before j can be started. (The meaning of "precedence" will be redefined in Section 4.1.1.) Constraints (4.6) and (4.7) deal with pairs of operations i and j that are assigned to the same machine k. On the one hand, constraints (4.6) say that if i and j were assigned to the same machine k, then i must be processed before j or j must be processed before i, i.e., an order must be established. On the other hand, constraints (4.7) say that if i and j were assigned to the same machine k and i is processed before j, then between the starting times of i and j there must be enough time to process i. Constraints (4.8) say that each operation i can not start to be processed before its release time; and constraints (4.9) fix the start time of the operations whose start times are already pre-determined at $\bar{s}_i$.

4.1.1 Modeling Partial Overlap

The simplest feature to be added to model (4.1–4.9) is the redefinition of precedence. The modification consists in substituting (4.5) by

$$\bar{p}'_i = \sum_{k \in F(i)} x_{ik} \lceil \theta_i p_{ik} \rceil, \quad (i,j) \in A, \tag{4.10}$$

$$s_i + \bar{p}'_i \leq s_j, \quad (i,j) \in A, \tag{4.11}$$

and

$$s_i + p'_i \leq s_j + p'_j, \quad (i,j) \in A. \tag{4.12}$$

Constraints (4.10) define new variables $\bar{p}'_i$ ($i \in V$) whose values coincide with $\lceil \theta_i p'_i \rceil$. Constraints (4.11) say that, if there exists a precedence constraint saying that operation i must precede operation j, it means that operation j can not start before $100\% \times \lceil \theta_i p'_i \rceil / p'_i$ of operation i is completed; while constraints (4.12) say that operation j can not be completed before operation i is completed as well.

4.1.2 Modeling Machines' Unavailabilities

Modeling the machines' unavailability starts by considering real variables c_i and $\bar{c}_i$ ($i \in V$) and binary variables $v_{ik\ell}$, $w_{ik\ell}$, and $\bar{w}_{ik\ell}$ ($i \in V$, $k \in F(i)$, $\ell = 1, \ldots, q_k$). The

meaning of variables c_i and $\bar{c}_i$ $(i \in V)$ follows:

- Variable c_i represents the completion time of each operation i.
- Variable $\bar{c}_i$ represents the completion time of the fraction of each operation i that is required to be completed before the execution of an operation j if there exists $(i, j) \in A$; i.e., while c_i represents the completion time of the execution of p'_i units of time of operation i, $\bar{c}_i$ represents the completion time of the execution of $\bar{p}'_i$ units of time of operation i.

Variables $v_{ik\ell}$, $w_{ik\ell}$, and $\bar{w}_{ik\ell}$ relate s_i, c_i, and $\bar{c}_i$ with the periods of unavailability of the machines, respectively. The relation sought between s_i and $v_{ik\ell}$ is "$v_{ik\ell} = 1$ if and only if operation i is assigned to machine k and the ℓ-th period of unavailability of machine k is to the left of s_i." The relation sought between c_i and $w_{ik\ell}$ is analogous, namely, "$w_{ik\ell} = 1$ if and only if operation i is assigned to machine k and the ℓ-th period of unavailability of machine k is to the left of c_i". Finally, the relation sought between $\bar{c}_i$ and $\bar{w}_{ik\ell}$ is "$\bar{w}_{ik\ell} = 1$ if and only if operation i is assigned to machine k and the ℓ-th period of unavailability of machine k is to the left of $\bar{c}_i$". These relations can be modeled as follows:

$$\left.\begin{array}{rcl}
v_{ik\ell} & \leq & x_{ik} \\
s_i & \leq & \underline{u}^k_\ell - 1 + M v_{ik\ell} + M(1 - x_{ik}) \\
s_i & \geq & \bar{u}^k_\ell - M(1 - v_{ik\ell}) - M(1 - x_{ik}) \\
w_{ik\ell} & \leq & x_{ik} \\
c_i & \leq & \underline{u}^k_\ell + M w_{ik\ell} + M(1 - x_{ik}) \\
c_i & \geq & \bar{u}^k_\ell + 1 - M(1 - w_{ik\ell}) - M(1 - x_{ik}) \\
\bar{w}_{ik\ell} & \leq & x_{ik} \\
\bar{c}_i & \leq & \underline{u}^k_\ell + M \bar{w}_{ik\ell} + M(1 - x_{ik}) \\
\bar{c}_i & \geq & \bar{u}^k_\ell + 1 - M(1 - \bar{w}_{ik\ell}) - M(1 - x_{ik})
\end{array}\right\} \quad \begin{array}{l} i \in V,\ k \in F(i), \\ \ell = 1, \ldots, q_k. \end{array} \tag{4.13}$$

Note that, in addition, constraints (4.13) avoid starting and completion times of an operation to belong to the interior of a period of unavailability of the machine to which the operation was assigned. In fact, constraints (4.13) also avoid a starting time to coincide with the beginning of a period of unavailability as well as avoid completion times to coincide with the end of a period of unavailability. As result

$$u_i = \sum_{k \in F(i)} \sum_{\ell=1}^{q_k} (w_{ik\ell} - v_{ik\ell})(\bar{u}^k_\ell - \underline{u}^k_\ell), \quad i \in V, \tag{4.14}$$

represents the sum of the unavailabilities between instants s_i and c_i (assuming $s_i \leq c_i$) of the machine that process operation i; and

$$\bar{u}_i = \sum_{k \in F(i)} \sum_{\ell=1}^{q_k} (\bar{w}_{ik\ell} - v_{ik\ell})(\bar{u}_\ell^k - \underline{u}_\ell^k), \quad i \in V, \tag{4.15}$$

represents the sum of the unavailabilities between instants s_i and $\bar{c}_i$ (assuming $s_i \leq \bar{c}_i$) of the machine that process operation i.

Now, it is easy to see that the relations

$$\left.\begin{aligned} s_i \leq \bar{c}_i \leq c_i \\ s_i + p'_i + u_i = c_i \\ s_i + \bar{p}'_i + \bar{u}_i = \bar{c}_i \end{aligned}\right\} i \in V, \tag{4.16}$$

must hold. The two equality constraints in (4.16) make clear that the new variables c_i and $\bar{c}_i$ are being included into the model with the only purpose of simplifying the presentation, since every appearance of c_i and $\bar{c}_i$ could be replaced by the left-hand side of the respective equality. In fact, the same remark applies to other equality constraints in the model as, e.g., constraints (4.3) that define p'_i, constraints (4.10) that define $\bar{p}'_i$, constraints (4.14) that define u_i, and constraints (4.15) that define $\bar{u}_i$. With the inclusion of the new variables c_i and $\bar{c}_i$, constraints (4.4), (4.7), (4.11), and (4.12) can be restated, respectively, as follows:

$$c_i \leq C_{\max}, \quad i \in V, \tag{4.17}$$

$$c_i - (1 - y_{ij})M \leq s_j, \quad i, j \in V,\ i \neq j,\ F(i) \cap F(j) \neq \emptyset, \tag{4.18}$$

$$\bar{c}_i \leq s_j, \quad (i, j) \in A, \tag{4.19}$$

$$c_i \leq c_j, \quad (i, j) \in A. \tag{4.20}$$

Summing up, with new real variables $\bar{p}'_i$, c_i, and $\bar{c}_i$ ($i \in V$) and new binary variables $v_{ik\ell}$, $w_{ik\ell}$, and $\bar{w}_{ik\ell}$ ($i \in V$, $k \in F(i)$, $\ell = 1, \ldots, q_k$), the model that includes the machines' unavailable periods consists in minimizing (4.1) subject to (4.2, 4.3, 4.6, 4.8, 4.9, 4.10, 4.13, 4.14, 4.15, 4.16, 4.17, 4.18, 4.19, 4.20).

4.1.3 Modeling Sequence-Dependent Setup Time

In order to model the setup, binary variables y_{ij} will be replaced by binary variables y_{ijk} ($i, j \in V$, $i \neq j$, $k \in F(i) \cap F(j)$). The idea is that $y_{ijk} = 1$ if and only if operations i and j are assigned to machine k and i is the immediate predecessor of j.

If, for each machine k, the set $B_k = \{i \in V \mid k \in F(i)\}$ is defined, the required characterization of variables y_{ijk} can be achieved with the following constraints:

$$\left.\begin{array}{rcl} y_{ijk} & \leq & x_{ik} \\ y_{ijk} & \leq & x_{jk} \end{array}\right\} \quad k = 1, \ldots, m,\ i, j \in B_k,\ i \neq j, \tag{4.21}$$

$$\sum_{i,j \in B_k, i \neq j} y_{ijk} = \sum_{i \in B_k} x_{ik} - 1, \quad k = 1, \ldots, m, \tag{4.22}$$

$$\sum_{j \in B_k, j \neq i} y_{ijk} \leq 1, \quad k = 1, \ldots, m,\ i \in B_k, \tag{4.23}$$

$$\sum_{i \in B_k, i \neq j} y_{ijk} \leq 1, \quad k = 1, \ldots, m,\ j \in B_k. \tag{4.24}$$

Constraints (4.21) say that, if operation i or operation j is not assigned to machine k, then y_{ijk} must be equal to zero. For any machine k, $\sum_{i \in B_k} x_{ik}$ represents the number of operations assigned to it. Then, constraints (4.22) say that, for every machine, the number of precedence constraints related to the operations assigned to it must be one less than the number of operations assigned to it. Finally, constraints (4.23) and (4.24) say that each operation precedes at most one operation and it is preceded by at most one operation, respectively. In fact, it is expected each operation to precede and to be preceded by exactly one operation unless the first and the last operations in the scheduling of each machine.

The substitution of variables y_{ij} $(i, j \in V,\ i \neq j,\ F(i) \cap F(j) \neq \emptyset)$ by variables y_{ijk} $(i, j \in V,\ i \neq j,\ k \in F(i) \cap F(j))$ consists in replacing constraints (4.6) with constraints (4.21, 4.22, 4.23, 4.24) and replacing constraints (4.18) with

$$c_i - \left(1 - \sum_{k \in F(i) \cap F(j)} y_{ijk}\right) M \leq s_j, \quad i, j \in V,\ i \neq j,\ F(i) \cap F(j) \neq \emptyset. \tag{4.25}$$

Up to this point, the model is an alternative to the previous one, but more suitable to include the setup feature.

Now, observe that, if operation j is assigned to machine k and

$$\sum_{i \in B_k, i \neq j} y_{ijk} = 0,$$

this means that j is the first operation to be processed by machine k. In this case, the sequence-dependent setup time required to process operation j on machine k is given by γ^F_{jk}. On the other hand, if operation j is assigned to machine k and

$$\sum_{i \in B_k, i \neq j} y_{ijk} = 1,$$

then operation j is preceded by the only operation i such that $y_{ijk} = 1$. In this case, the sequence-dependent setup time required to process operation j on machine k is given by

$$\sum_{i \in B_k, i \neq j} y_{ijk} \gamma^I_{ijk}.$$

Thus, with the definition of

$$\hat{\xi}_{jk} = \left(\sum_{i \in B_k, i \neq j} y_{ijk} \gamma^I_{ijk} \right) + \left(1 - \sum_{i \in B_k, i \neq j} y_{ijk} \right) \gamma^F_{jk}, \quad j \in V,\ k \in F(j), \tag{4.26}$$

if operation j is assigned to machine k, then $\hat{\xi}_{jk}$ corresponds to the sequence-dependent setup time required to process operation j on machine k. Note that, if operation j is *not* assigned to machine k, by (4.26), then $\hat{\xi}_{jk} = \gamma^F_{jk}$. So, with the definition of new real variables $\bar{\xi}_{jk}$ ($j \in V$, $k \in F(j)$) and constraints

$$\left.\begin{array}{rcl} 0 & \leq \bar{\xi}_{jk} \leq & M x_{jk} \\ \hat{\xi}_{jk} - M(1 - x_{jk}) & \leq \bar{\xi}_{jk} \leq & \hat{\xi}_{jk} \end{array}\right\} j \in V,\ k \in F(j), \tag{4.27}$$

if operation j is assigned to machine k, then its sequence-dependent setup time is given by $\bar{\xi}_{jk}$; while $\bar{\xi}_{jk} = 0$ if operation j is not assigned to machine k. Thus,

$$\xi_j = \sum_{k \in F(j)} \bar{\xi}_{jk}, \quad j \in V, \tag{4.28}$$

represents the sequence-dependent setup time of operation j.

For including the setup time into the model, it is required to replace constraints (4.25) with

$$c_i - \left(1 - \sum_{k \in F(i) \cap F(j)} y_{ijk} \right) M \leq s_j - \xi_j, \quad i, j \in V,\ i \neq j,\ F(i) \cap F(j) \neq \emptyset, \tag{4.29}$$

to say that between the end of an operation and the beginning of the next operation there must be enough time to process the corresponding setup. It remains to say that, in the OPS scheduling problem, a setup operation can not be interrupted and that its completion time must coincide with the starting time of the operation itself. This requirement corresponds to constraints

$$\left.\begin{array}{rcl} s_i - \xi_i & \leq & \underline{u}^k_\ell - 1 + M v_{ik\ell} + M(1 - x_{ik}) \\ s_i - \xi_i & \geq & \bar{u}^k_\ell - M(1 - v_{ik\ell}) - M(1 - x_{ik}) \end{array}\right\} i \in V,\ k \in F(i),\ \ell = 1, \ldots, q_k \tag{4.30}$$

plus

$$s_i \geq \xi_i, \quad i \in V. \tag{4.31}$$

Note that constraints (4.30), roughly speaking, say that, if s_i is in between two unavailabilities, then $s_i - \xi_i$ must be in between the same two unavailabilities. Combining (4.30) with (4.13) results in

$$\left.\begin{array}{rcl} v_{ik\ell} & \leq & x_{ik} \\ s_i & \leq & \underline{u}_\ell^k - 1 + M v_{ik\ell} + M(1 - x_{ik}) \\ s_i - \xi_i & \geq & \bar{u}_\ell^k - M(1 - v_{ik\ell}) - M(1 - x_{ik}) \\ w_{ik\ell} & \leq & x_{ik} \\ c_i & \leq & \underline{u}_\ell^k + M w_{ik\ell} + M(1 - x_{ik}) \\ c_i & \geq & \bar{u}_\ell^k + 1 - M(1 - w_{ik\ell}) - M(1 - x_{ik}) \\ \bar{w}_{ik\ell} & \leq & x_{ik} \\ \bar{c}_i & \leq & \underline{u}_\ell^k + M \bar{w}_{ik\ell} + M(1 - x_{ik}) \\ \bar{c}_i & \geq & \bar{u}_\ell^k + 1 - M(1 - \bar{w}_{ik\ell}) - M(1 - x_{ik}) \end{array}\right\} \begin{array}{l} i \in V,\ k \in F(i), \\ \ell = 1, \ldots, q_k. \end{array} \tag{4.32}$$

Summing up, with new real variables $\hat{\xi}_{jk}$, $\bar{\xi}_{jk}$ ($j \in V$, $k \in F(j)$), and ξ_j ($j \in V$) and new binary variables y_{ijk} ($i, j \in V$, $i \neq j$, $F(i) \cap F(j) \neq \emptyset$) that substitute the binary variables y_{ij} ($i, j \in V$), the updated model that includes the sequence-dependent setup time consists in minimizing (4.1) subject to (4.2, 4.3, 4.8, 4.9, 4.10, 4.14, 4.15, 4.16, 4.17, 4.19, 4.20, 4.21, 4.22, 4.23, 4.24, 4.26, 4.27, 4.28, 4.29, 4.31, 4.32). Constraints (4.27), (4.29), and (4.32) depend on a "sufficiently large" constant M whose value needs to be defined. In (4.27), a sufficiently large value for M is given by

$$M_1 = \max\left\{ \max_{j \in V,\ k \in F(j)} \left\{\gamma_{jk}^F\right\}, \max_{i,j \in V,\ i \neq j,\ F(i) \cap F(j) \neq \emptyset} \left\{\gamma_{ijk}^I\right\} \right\}.$$

In (4.29), a sufficiently large value for M is given by any upper bound for the optimal $C_{\max}$ like, e.g.,

$$M_2 = \max_{\{k=1,\ldots,m\}} \{\bar{u}_{q_k}^k\} + \sum_{j \in V} \max_{k \in F(j)} \left\{ p_{jk} + \max\left\{ \gamma_{jk}^F, \max_{\{i \in V | k \in F(i)\}} \left\{\gamma_{ijk}^I\right\} \right\} \right\}.$$

Note that M_2 is a loose upper bound for the optimal $C_{\max}$. The same value M_2 can be used in the second, the fifth, and the eighth inequalities in (4.32); while, in the third, the sixth, and the ninth inequalities in (4.32),

$$M_3 = \max_{\{k=1,\ldots,m\}} \{\bar{u}_{q_k}^k\}.$$

must be used.

4.2 Constraint Programming Formulation

Constraint Programming (Rossi, Van Beek, and Walsh, 2006) is a powerful paradigm for solving combinatorial problems; and it is particularly attractive for problems that do not have a simple formulation in terms of linear constraints, as it is the case of the OPS scheduling problem being considered in the present work. CP Optimizer (Laborie et al., 2018) is an optimization engine based on CP that extends classical CP with a few mathematical concepts that make it easier to model scheduling problems while providing an interesting problem's structure to its automatic search algorithm. The automatic search is an exact algorithm (so it produces optimality proofs) that internally uses some metaheuristics, mainly the Self-Adapting Large-Neighborhood Search (Laborie and Godard, 2007), to quickly produce good quality solutions that help to prune the search space. A formulation of the OPS scheduling problem suitable to be solved with CP Optimizer is presented below. The CP Optimizer concepts will be briefly described as soon as they appear in the formulation. For more details, please refer to (Laborie et al., 2018) and to the CP Optimizer reference manual.

A formulation using the concepts of CP Optimizer equivalent to the MILP model (4.1–4.9) can be written as follows:

$$\text{Minimize} \max_{i \in V} \text{endOf}(o_i) \tag{4.33}$$

$$\text{subject to}$$

$$\text{endBeforeStart}(o_i, o_j), \quad (i,j) \in A, \tag{4.34}$$

$$\text{alternative}(o_i, [a_{ik}]_{k \in F(i)}), \quad i \in V, \tag{4.35}$$

$$\text{noOverlap}([a_{ik}]_{i \in V : k \in F(i)}), \quad k = 1, \ldots, m, \tag{4.36}$$

$$\text{startOf}(o_i) \geq r_i, \quad i \in V, \tag{4.37}$$

$$\text{startOf}(o_i) = \bar{s}_i, \quad i \in T, \tag{4.38}$$

$$\text{interval}\, o_i, \quad i \in V, \tag{4.39}$$

$$\text{interval}\ a_{ik},\ \text{opt},\ \text{size} = p_{ik}, \quad i \in V,\ k \in F(i). \tag{4.40}$$

Decision variables of the problem are described in (4.39) and (4.40). In (4.39), an interval variable o_i for each operation i is defined. In (4.40), an "optional" interval variable a_{ik} is being defined for each possible assignment of operation i to a machine $k \in F(i)$. Optional means that the interval variable may exist or not; and the remaining of the constraint says that, in case it exists, its size must be p_{ik}. The objective function (4.33) is to minimize the makespan, given by the maximum end value of all the operations represented by the interval variables o_i. Precedence constraints between operations are posted as endBeforeStart constraints between interval variables

in constraints (4.34). Constraints (4.35) state that each operation i must be allocated to exactly one machine $k \in F(i)$ that is, one and only one interval variable a_{ik} must be present and the selected interval a_{ik} will start and end at the same values as interval o_i. Constraints (4.36) state that, for a machine k, the intervals a_{ik} representing the assignment of the operations to this machine do not overlap. (It should be noted that this noOverlap constraints actually create a hidden sequence variable on the intervals a_{ik}. More details on sequence variables will be given on Section 4.2.3.) Finally, constraints (4.37) say that each operation i can not start to be processed before its release time; and constraints (4.38) fix the starting time of the operations whose starting times are already pre-determined.

The CP Optimizer model (4.33–4.40) can be strengthened by a redundant constraint stating that, at any moment in time, there are never more than m machines being used simultaneously. This constraint (that implicitly apply to discrete instants in time) is given by

$$\sum_{i \in V} \text{pulse}(o_i, 1) \leq m. \tag{4.41}$$

4.2.1 Modeling Machines' Unavailabilities

Since the operations may be suspended by the unavailabilities of the machines, the definition of the interval variable a_{ik} must be modified by considering as intensity function a step function U_k that represents the unavailability of machine k. In CP Optimizer, step functions are constant structures of the model that are represented by a set of steps associated with a value. The value of the step function U_k is 0% when machine k is unavailable, i.e., on time windows $[\underline{u}_1^k, \bar{u}_1^k], ..., [\underline{u}_{q_k}^k, \bar{u}_{q_k}^k]$, and 100% in between these time windows. So, including the machines' unavailabilities in model (4.33–4.40) simply consists in replacing (4.40) with

$$\text{interval } a_{ik}, \text{ opt, size} = p_{ik}, \text{ intensity} = U_k, \ i \in V, k \in F(i). \tag{4.42}$$

4.2.2 Modeling Partial Overlap

To model partial overlapping between operations subjected to precedence constraints, each interval variable o_i (resp. a_{ik}) is associated with an additional interval variable ω_i (resp. α_{ik}) that represents the proportion of operation i that has to be processed before any of its successors can start. The size of the optional interval variable α_{ik} is defined as $\lceil \theta_i p_{ik} \rceil$ (see (4.43) and (4.44)); and interval variables a_{ik} and α_{ik} have the same presence status (see (4.45)). Interval ω_i is the alternative between all the interval variables α_{ik} (see (4.46)); and interval variables o_i and ω_i start at the same time (see (4.47)). This way, when operation i is allocated to machine k, both interval

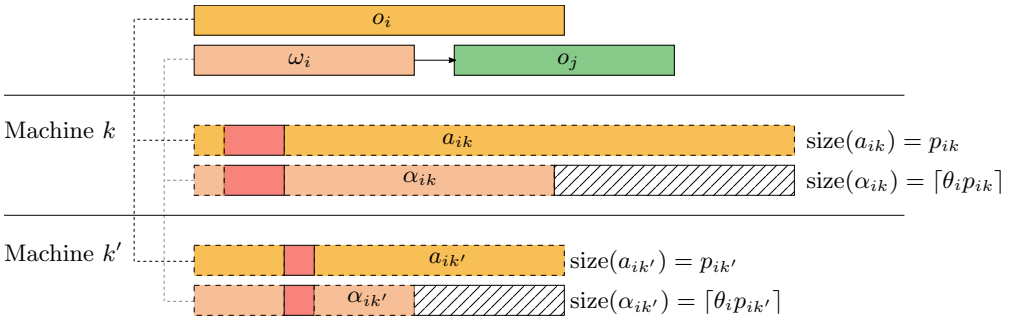


Figure 4.1: Illustration of the overlap relation between two operations i and j subjected to precedence constraints and the interval variables o_i, a_{ik}, and α_{ik}.

variables a_{ik} and α_{ik} are present, the size of α_{ik}, that represents the proportion of operation i to be executed before any of its successors, is equal to $\lceil \theta_i p_{ik} \rceil$, and interval variable ω_i is synchronized with the start and end of α_{ik}. Precedence constraints are posted between ω_i and its successors in (4.48). Constrains (4.49) say that operation j can not be completed before the completion of operation i if $(i, j) \in A$.

$$\text{interval } \omega_i, \quad i \in V, \tag{4.43}$$

$$\text{interval } \alpha_{ik}, \text{ opt, size} = \lceil \theta_i p_{ik} \rceil, \text{intensity} = U_k, \quad i \in V, k \in F(i), \tag{4.44}$$

$$\text{presenceOf}(a_{ik}) == \text{presenceOf}(\alpha_{ik}), \quad i \in V, k \in F(i), \tag{4.45}$$

$$\text{alternative}(\omega_i, [\alpha_{ik}]_{k \in F(i)}), \quad i \in V, \tag{4.46}$$

$$\text{startAtStart}(\omega_i, o_i), \quad i \in V, \tag{4.47}$$

$$\text{endBeforeStart}(\omega_i, o_j), \quad (i, j) \in A, \tag{4.48}$$

$$\text{endBeforeEnd}(o_i, o_j), \quad (i, j) \in A. \tag{4.49}$$

The model that includes the machines unavailabilities as well as the partial overlapping consists in minimizing (4.33) subject to (4.35–4.39), (4.42), and (4.43–4.49) (that substitute constraints (4.34)); constraints (4.41) being, as already mentioned, optional. Figure 4.1 illustrates the relation between the interval variables o_i, a_{ik}, and α_{ik}. In the figure, it is assumed that there is a precedence constraint between operations i and j, there are two alternative machines k and k' to process operation i; and operation i is assigned to machine k' since the size of ω_i coincides with the size of $\alpha_{ik'}$. The size of ω_i corresponds to the proportion of operation i that must be processed before operation j can start to be processed.

4.2.3 Modeling Sequence-Dependent Setup Time

Another additional feature of the OPS scheduling problem is the notion of setup time and setup activities between consecutive operations executed on a machine. The

usual formulation for sequence-dependent setup times in CP Optimizer is to use a sequence variable that permits to associate an integer type with each interval variable in the sequence and to post a no-overlapping constraint on this sequence with a transition distance matrix. This is the purpose of the sequence variable SA_k defined in (4.50). This sequence variable is defined on all the interval variables a_{ik} on machine k (optional interval variables representing the possible execution of operation i on machine k). Each interval a_{ik} is associated with a type i in the sequence variable. A non-overlapping constraint is posted in (4.51), specifying the transition distance matrix Γ^I_k defined as $\Gamma^I_k[i][j] = \gamma^I_{ijk}$. These constraints ensure that operations allocated to machine k do not overlap and that a minimal setup time of $\Gamma^I_k[i][j]$ must elapse between any two consecutive operations i and j. The definition of the sequence variable and the non-overlapping constraints are given by

$$\text{sequence } SA_k \text{ on } [a_{ik}]_{i\in V:k\in F(i)}, \text{ types } [i]_{i\in V:k\in F(i)}, \quad k = 1,\ldots,m, \tag{4.50}$$

$$\text{noOverlap}(SA_k, \Gamma^I_k), \quad k = 1,\ldots,m. \tag{4.51}$$

However, in the OPS scheduling problem, setup activities are also subject to the unavailability of the machines and, in particular, the setup cannot be interrupted by an unavailability time window. Because of these complex constraints on the setups, the setup operations need to be explicitly represented as interval variables in the model. The explicit representation of the setup operations as interval variables also allows us to model the setup of the first operation being processed by a machine, a case that is not covered by (4.50, 4.51).

For each interval a_{ik}, it is defined in (4.52) an interval variable s_{ik} that represents the setup activity just before operation a_{ik}; and it is established in (4.59) that the end of s_{ik} must coincide with the beginning of a_{ik}. For each interval a_{ik}, it is also defined in (4.53) an interval variable c_{ik} that covers both s_{ik} and a_{ik} (see (4.55)). Interval variables a_{ik}, s_{ik}, and c_{ik} have the same presence status (see (4.56) and (4.57)). A sequence variable SC_k representing the non-overlapping sequence of intervals c_{ik} on machine k is defined in (4.54), the non-overlapping being represented in (4.60). The size of the setup activity s_{ik} depends both on the type of the previous operation on sequence SC_k and the type of operation i; and its value is given in (4.58) by a matrix Γ_k. Matrix Γ_k is a matrix with row index starting from 0 and defined as the concatenation of matrices Γ^F_k and Γ^I_k, i.e., Γ_k consists in Γ^I_k with an additional 0-th row given by $\Gamma^F_k[i] = \gamma^F_{ik}$. By convention, when operation i is assigned to a machine k and it is the first activity executed by the machine, the type of the previous operation on sequence SC_k is 0 so that the size of the setup activity s_{ik} is $\Gamma_k[0][i] = \Gamma^F_k[i] = \gamma^F_{ik}$. Figure 4.2 illustrates the relation between interval variables o_i, a_{ik}, and c_{ik}. In the figure, it is shown the explicit inclusion of the setup interval variables s_{ik}. The right

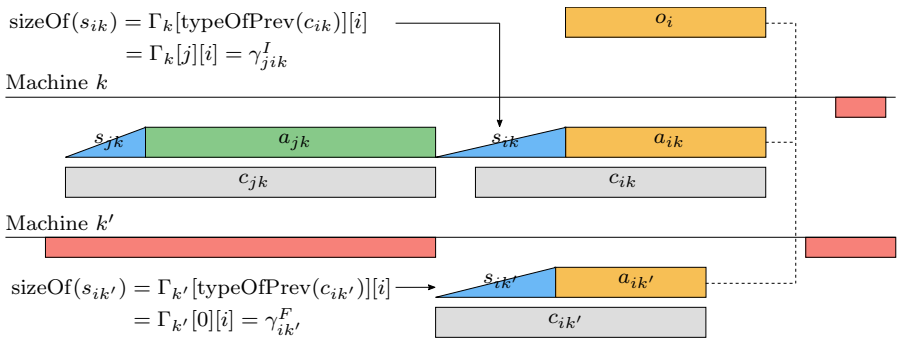


Figure 4.2: Illustration of the setup time activities for operations i and its respective interval variables o_i, a_{ik}, and c_{ik} in different machines with different precedence relations.

hand side of the graphic shows two alternatives for processing operation i on machine k or on machine k'. The left hand side illustrates the computation of the sequence-dependent setup time.

Finally, a set of constraints is added to ensure the behavior of operations and setup activities with respect to machine unavailable periods. The intensity function U_k used in the definition of the interval variables representing operations (4.42) states that operations are suspended by unavailabilities. Additionally, constraints (4.61) and (4.62) establish that operations cannot start or end during an unavailability period; whereas constraints (4.63) say that setup activities cannot overlap unavailability periods. The new set of constraints follows:

$$\text{interval } s_{ik}, \text{ opt}, \quad i \in V, k \in F(i), \tag{4.52}$$

$$\text{interval } c_{ik}, \text{ opt}, \quad i \in V, k \in F(i), \tag{4.53}$$

$$\text{sequence } SC_k \text{ on } [c_{ik}]_{i \in V : k \in F(i)}, \text{ types } [i]_{i \in V : k \in F(i)}, \quad k = 1, \ldots, m, \tag{4.54}$$

$$\text{span}(c_{ik}, [s_{ik}, a_{ik}]), \quad i \in V, k \in F(i), \tag{4.55}$$

$$\text{presenceOf}(a_{ik}) == \text{presenceOf}(c_{ik}), \quad i \in V, k \in F(i), \tag{4.56}$$

$$\text{presenceOf}(a_{ik}) == \text{presenceOf}(s_{ik}), \quad i \in V, k \in F(i), \tag{4.57}$$

$$\text{sizeOf}(s_{ik}) == \Gamma_k[\text{typeOfPrev}(SC_k, c_{ik})][i], \quad i \in V, k \in F(i), \tag{4.58}$$

$$\text{endAtStart}(s_{ik}, a_{ik}), \quad i \in V, k \in F(i), \tag{4.59}$$

$$\text{noOverlap}(SC_k), \quad k = 1, \ldots, m, \tag{4.60}$$

$$\text{forbidStart}(a_{ik}, U_k), \quad i \in V, k \in F(i), \tag{4.61}$$

$$\text{forbidEnd}(a_{ik}, U_k), \quad i \in V, k \in F(i), \tag{4.62}$$

$$\text{forbidExtent}(s_{ik}, U_k), \quad i \in V, k \in F(i). \tag{4.63}$$

Summing up, the full CP Optimizer formulation of the OPS scheduling problem is given by the minimization of (4.33) subject to constraints (4.35–4.39, 4.42, 4.43–4.49, 4.52–4.63); constraints (4.41) being, as already mentioned, optional. In the same

sense, constraints (4.50, 4.51), superseded by constraints (4.52–4.63), can be considered optional. It is expected that keeping them would result in a stronger inference in the resolution process due to the direct formulation of the minimal distance Γ_k^I between consecutive operations on each machine k.

4.3 Analysis of the MILP and CP Models

In this section, the relation between the main components of the MILP and the CP Optimizer formulations of the OPS scheduling problem are addressed. The main difference between both formulations is that, in the CP Optimizer model, the number of explicit variables and constraints is $O(o\,m)$; while, in the MILP formulation, the number of variables and constraints is, in the worst case, $O(o^2m+o\sum_{k=1}^{m} q_k)$, where o is the number of operations, m is the number of machines, and q_k is the number of periods of unavailability of machine k. A tighter bound is given by $O(|A|+\sum_{k=1}^{m}(|B_k|^2+|B_k|q_k))$, where A is the set of precedence relations and, for each machine k, B_k is the set of operations that could be processed by it. On the other hand, every constraint in the MILP model involves a constant number of variables or a number that is, in the worst case, $O(o+\sum_{k=1}^{m} q_k)$; while, in the CP Optimizer model, each non-overlap constraint on the sequence SA_k $(k = 1,\ldots,m)$ involves a dense matrix Γ_k^I of size o^2. So, as expected, the non-overlapping constraints in the CP Optimizer model involve, as a whole, dealing with the $O(o^2m)$ given setup times. (The same is true, in a similar way, for the sequence variables SC_k.) Another difference between the models is that the CP Optimizer model strongly relies on the integrality of all the constants that define an instance, i.e., it assumes that the processing times of the operation on the machines, the fixed starting times, the setup times, the beginning and the end of the machines' unavailability periods, and the release times are all integer values. If, on the one hand, this is *not* a requirement of the MILP formulation; on the other hand, this is a usual assumption that can be accomplished, in practice, by changing the constants' unit of measure. The same is not true in the required interpretation of partial overlapping. In the considered definition of overlapping, the interpretation of constant θ_i is that if p'_i is the processing time of operation i on the machine to which it was assigned, then operation i must be processed at least $\lceil \theta_i p'_i \rceil$ units of time before any successor can start to be processed. The fact of using $\lceil \cdot \rceil$ in the interpretation of overlapping is needed in the CP Optimizer formulation, due to the integrality assumption; while it is not relevant at all in the MILP formulation.

One of the most relevant components of both models is the one that represents the "duration" of each operation, where duration stands for its processing time on the machine to which it was assigned plus the duration of the unavailable periods of such

machine that interrupt its execution. In the CP Optimizer model, this object is represented by the interval variables o_i. The equivalent object in the MILP formulation is given by the starting time s_i and the completion time c_i, that correspond, respectively, to the beginning and the end of the interval variable o_i of the CP Optimizer model. In the MILP model, the time elapsed between s_i and c_i is divided into the processing time of the operation itself, represented by p'_i, and the sum of the unavailability windows in between s_i and c_i, represented by u_i. In the CP Optimizer model, this distinction is made with the help of the indicator function U_k.

Another object present in both formulations and strongly related to the duration of an operation is the duration of the fraction of an operation that must be processed before any successor of it can start to be processed. In the CP Optimizer model, this role is played by the interval variables ω_i. In the MILP model, the beginning and the end of this interval are given by variables s_i and $\bar{c}_i$, respectively. Not by coincidence, there is a constraint in the CP Optimizer model saying that the beginning of o_i and ω_i must coincide. (In the MILP model the beginning of these two intervals is represented by the same variable s_i.) Once again, the time elapsed between s_i and $\bar{c}_i$ is divided in the MILP model in the actual processing of the operation (represented by $\bar{p}'_i$) and the aggregated time of the unavailability windows that interrupt the execution of the operation, represented by $\bar{u}_i$). The same effect is, once again, obtained in the CP Optimizer model through the usage of the indicator function U_k.

The assignment of an operation to a machine is modeled in the MILP formulation with binary variables x_{ik}. Then, with the same variables, the machine-dependent processing time p_i of operation i is established. The same thing is done in the CP Optimizer formulation with the "alternative" constraint that relates the interval variables o_i with one and only one variable a_{ik} for some $k \in F(i)$, whose duration is p_{ik}. Note that, in the MILP model, the x_{ik} variables also only exist if $k \in F(i)$. Both situations correspond to the disjunction that is modeled in a classical way in the MILP model; while it corresponds to a primitive of the CP Optimizer modeling language in the other case. The non-overlapping between operations being processed by the same machine k is once again modeled with the help of binary variables y_{ijk} in the MILP model. In CP Optimizer, the "noOverlap" constraints are posted on a hidden "sequence" variable that is defined over a set of interval variables, each interval variable in the set being associated with an integer "type." The "noOverlap" constraints state that the intervals of the sequence must be ordered as a set of non-overlapping intervals (that typically represent the different operations on a machine). The "type" of the types of the intervals may, for instance, be used to index a setup time matrix that represents a minimal distance between consecutive intervals in the sequence. In the MILP model, the setup feature is achieved with the help of the y_{ijk} variables. All other elements of

both models also relate similarly.

As it was described, the MILP and the CP Optimizer model represent the OPS scheduling problem using equivalent structures; the difference lies on the fact that the CP Optimizer modeling language provides to the CP Optimizer solver a more compact model and much more structure of the problem than the one that the MILP model could give to a general-purpose MILP solver.

4.4 Experimental Verification and Analysis

The numerical experiments in the present section have three goals. Experiments with small-sized instances aim to compare the efficiency and effectiveness of the commercial solvers IBM ILOG CPLEX and IBM ILOG CP Optimizer (both included in version 12.9). The formulations were implemented using the Python API DOcplex 2.10.155 library. Experiments with medium-sized instances aim to determine the size of the instances for which optimality can be proved with the commercial solvers IBM ILOG CP Optimizer. Experiments with large-sized instances, that are of the size of real instances of the OPS scheduling problem, are also considered. The goal of these experiments is to evaluate the possibility of using feasible solutions found by the IBM ILOG CP Optimizer in practice.

The experiments, carried out using the High-Performance Computing (HPC) facilities of the University of Luxembourg (Varrette et al., 2014), were conducted on an Intel Xeon E5-2680 v4 2.4 GHz with 4GB memory (per core) running CentOS Linux 7.7 (in 64-bit mode); always using a single physical core.

By default, see (IBM ILOG CPLEX Optimization Studio, 2020, pp. 227–228), a solution to a MILP model is reported by CPLEX when

$$\text{absolute gap} = \text{incumbent solution} - \text{best lower bound} \leq \epsilon_{\text{abs}}$$

or

$$\text{relative gap} = \frac{|\text{incumbent solution} - \text{best lower bound}|}{10^{-10} + |\text{incumbent solution}|} \leq \epsilon_{\text{rel}},$$

with $\epsilon_{\text{abs}} = 10^{-6}$ and $\epsilon_{\text{rel}} = 10^{-4}$. In the considered instances, the objective function (4.1) assumes integer values at feasible points. Thus, on the one hand, a stopping criterion based on a relative error less than or equal to $\epsilon_{\text{rel}} = 10^{-4}$ may have the undesired effect of stopping the method prematurely; and, on the other hand, an absolute error strictly smaller than 1 is enough to prove the optimality of the incumbent solution. Therefore, following Birgin, Romão, and Ronconi (2020) and Andrade et al. (2014), ϵ_{abs} and ϵ_{rel} were set to $\epsilon_{\text{abs}} = 1 - 10^{-6}$ and $\epsilon_{\text{rel}} = 0$. All other parameters

of CPLEX were kept with their default values. CP Optimizer was run with all its default parameters. A CPU time limit of two hours per instance was imposed.

Section 4.4.1 describes the generation of instances; while Sections 4.4.2, 4.4.3, and 4.4.4 presents respectively numerical results with small-, medium-, and large-sized instances. The implementation of the MILP and the CP Optimizer formulations, the generator of random instances, and all the generated instances are freely available at `https://github.com/willtl/online-printing-shop`.

4.4.1 Generation of Instances

Numerical experiments with the introduced models were performed on random instances. All constants that define an instance are numbers randomly chosen with uniform distribution in a predefined interval. So, from now on, whenever "chosen", "random" or "randomly chosen" is written, it should be read "randomly chosen with uniform distribution". It should be noted that, although random, instances are generated in such a way they preserve the characteristics of the real instances of the OPS scheduling problem. Three different sets with small-, medium-, and large-sized instances will be generated, the large-sized instances being of the size of the real instances of the OPS scheduling problem.

The generation of an instance relies on six given integer parameters, namely, the number of jobs n, the minimum $o_{\min}$ and maximum $o_{\max}$ number of operations *per* job, the minimum $m_{\min}$ and the maximum $m_{\max}$ number of machines, and the maximum number q of periods of unavailability *per* machine. Precedence constraints are defined as follows; starting with $A = \emptyset$. For each job $j \in \{1, \ldots, n\}$, the generation of its associated DAG starts by choosing its number of operations $o_j \in [o_{\min}, o_{\max}]$; and it proceeds by layers. Starting by layer L_0, between 1 and 4 operations are chosen to populate L_0. Additional layers L_i are also populated with 1 to 4 operations, until the number of operations o_j is reached. Operations in L_0 have no predecessors; while operations in the last layer have no successors. For every layer L_i, unless the last one, and every $v \in L_i$, an operation $w \in L_{i+1}$ is randomly chosen and the arc (v, w) is added to A. All the other pairs (v, w) with $v \in L_i$ and $w \in L_{i+1}$ are included in A with probability 0.85. Figure 4.3 shows the random DAG of a job j with $o_j = 40$. The visual representation of the DAG was drawn by using the network simplex layering proposed in (Gansner et al., 1993). Note that the total number of operations is defined as $o = \sum_{j=1}^{n} o_j$. For each operation i such that an arc of the form $(i, \cdot) \in A$ exists, with probability 0.1, the overlapping coefficient θ_i is a real number chosen in $[0.5, 0.99]$, otherwise, θ_i is set to 1.

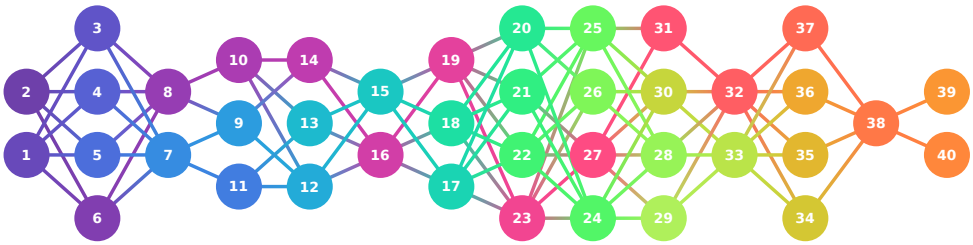


Figure 4.3: Random DAG representing the precedence constraints of a job with 40 operations. All arcs are directed from left to right.

The number of machines is given by a random number $m \in [m_{\min}, m_{\max}]$. For each operation i, the cardinality of $F(i)$ is given by a random number in $[\lceil 0.3m \rceil, \lceil 0.7m \rceil]$ and the elements of $F(i) \subseteq \{1, \ldots, m\}$ are randomly chosen. Then, a machine $\hat{k} \in F(i)$ and the associated integer processing time $p_{i\hat{k}} \in [1, 99]$ are randomly chosen. For all other machines $k \in F(i)$, $k \neq \hat{k}$, $p_{ik} \in [p_{i\hat{k}}, \min\{3p_{i\hat{k}}, 99\}]$ is randomly chosen. The number of periods of unavailability q_k of a machine k is chosen at random in $[1, q]$. Let φ_k be the mean of the processing times p_{ik} of the operations i such that $k \in F(i)$. Then, it is defined $a_k = 1 + \lceil \frac{\varphi_k}{q_k} \rceil$ as the distance between consecutive periods of unavailability. The first period of unavailability is given by $[\underline{u}_1^k, \bar{u}_1^k] = [a_k, \underline{u}_1^k + \lceil \frac{a_k}{R_1^k} \rceil + 1]$, where $R_1^k \in [2, 10]$ is a random integer number. For $\ell = 2, \ldots, q_k$, the ℓ-th period of unavailability is given by $[\underline{u}_\ell^k, \bar{u}_\ell^k] = [\bar{u}_{\ell-1}^k + a_k, \underline{u}_\ell^k + \lceil \frac{a_k}{R_\ell^k} \rceil + 1]$, where $R_\ell^k \in [2, 10]$ is a random integer number.

Each operation i has three randomly chosen integer values $size_i \in [1, \bar{S}]$, $color_i \in [1, \bar{C}]$, and $varnish_i \in [1, \bar{V}]$, with $\bar{S} = 10$, $\bar{C} = 4$, and $\bar{V} = 6$, that stand for the operation's size, color, and varnish, respectively. Consider two operations i and j that are processed consecutively, i before j, on a machine k. If $size_i < size_j$, then st'_k units of time are required to setup the machine; while st''_k units of time are required if $size_i > size_j$. (No setup time is required, regarding the size feature, if $size_i = size_j$.) If $color_i \neq color_j$, ct_k additional units of time are required; and if $varnish_i \neq varnish_j$, other additional vt_k units of time are required. Values st'_k, st''_k, ct_k, and vt_k are (machine dependent) random integer values in $[2, 6]$. The sum of the required values composes the setup time $\gamma_{ijk}^I \geq 0$. The setup time γ_{ik}^F for the case in which operation i is assigned to machine k and it is the first operation to be executed on the machine is given by $\gamma_{ik}^F = \max\{st'_k, st''_k\} + ct_k + vt_k$. It should be noted that the duration of a setup operation in between two consecutive operations i and j is not related to the processing time of the operations. A clear example of this situation corresponds to the setup operations of a cutting machine. The setup of the machine corresponds to adjusting the machine from the size of printed sheets and the cutting pattern of operation i to the size of printed sheets and the cutting pattern of operation j; and this adjustment is not related to the quantity of pieces that must be cut in the two operations.

The release time r_i of each operation i is equal to 0 with probability 0.975. When r_i is not zero, it is a random integer number within the interval $[1, 99]$. An operation i with no predecessors has probability 0.01 of belonging to T; i.e., having a fixed starting time $\bar{s}_i$ at a predefined machine $k \in F(i)$. (At most one fixed operation per machine is allowed.) If this is the case, p_{ik} is redefined as a random number in $[1, 99]$, $F(i)$ is redefined as the singleton $F(i) = \{k\}$, and $\bar{s}_i$ is randomly chosen in $[\gamma^F_{\text{ub}}, \underline{u}^k_1 - p_{ik}]$, where $\gamma^F_{\text{ub}} = \bar{S} + \bar{C} + \bar{V}$ is an upper bound of the setup time of an operation that is the first operation to be processed by a machine, as it is the case of a fixed operation generated in this way.

4.4.2 Experiments With Small-Sized Instances

In this section, a set of 30 small-sized instances is considered. The k-th instance was generated with the following parameters: $n = 1+\lceil \frac{k}{30} \times 3 \rceil$, $o_{\min} = 2$, $o_{\max} = 3+\lceil \frac{k}{30} \times 2 \rceil$, $m_{\min} = 2$, $m_{\max} = 3 + \lceil \frac{k}{30} \times 2 \rceil$, and $q = 4$. Table 4.2 shows the main characteristics of each instance; while Table 4.3 shows the value of the solutions found when solving the MILP and the CP Optimizer formulations with IBM ILOG CPLEX and IBM ILOG CP Optimizer solvers, respectively. The associated effort measurements are also shown in the table. Most of the columns in the table are self-explanatory. When optimality is not proven, column "Makespan" shows the best lower bound, the best upper bound, and the gap. The precise CP Optimizer model that is being solved and the meaning of the columns "number of branches in phases 1 and 2" will be elucidated in the next section. The CPU time is expressed in seconds. Figures in the table clearly show that the CP Optimizer solver outperformed the MILP solver. In the ten instances with $n = 2$, both solvers performed similarly. In the ten instances with $n = 3$, the CP Optimizer solver outperformed the MILP solver by one or two orders of magnitude. In the ten instances with $n = 4$, the MILP solver was not able to prove optimality of any of the instances; while the CP Optimizer solver solved all instances in a few seconds of CPU time. As an illustration, Figure 4.4 shows the solution to instance 30 found for the MILP and the CP Optimizer formulations by IBM ILOG CPLEX and IBM ILOG CP Optimizer solvers, respectively, but proven to be optimal in the latter case only.

Table 4.2: Main features of the considered thirty small-sized instances.

	Main instance characteristics						MILP formulation			CP Optimizer formulation	
Instance	m	$\sum_{k=1}^{m} q_k$	n	o	$\|A\|$	$\|T\|$	# binary variables	# continuous variables	# constraints	# integer variables	# constraints
1	3	7	2	9	10	1	248	109	889	76	209
2	4	10	2	8	8	1	220	99	784	67	187
3	3	8	2	8	7	1	238	99	843	69	191
4	4	10	2	9	9	1	263	113	928	77	217
5	2	5	2	9	8	2	200	101	752	58	162
6	2	4	2	8	7	1	184	93	685	57	159
7	2	6	2	8	6	0	234	93	832	54	154
8	4	11	2	7	6	0	240	93	836	69	192
9	4	8	2	8	7	1	212	103	766	81	214
10	4	13	2	10	8	0	425	131	1416	104	280
11	4	8	3	14	14	0	549	181	1805	135	372
12	3	9	3	11	8	0	354	131	1225	83	231
13	3	7	3	14	13	0	458	163	1546	100	281
14	3	8	3	13	12	0	514	163	1701	116	323
15	2	4	3	13	12	1	322	145	1169	83	236
16	4	8	3	13	12	0	485	169	1619	121	341
17	2	4	3	14	14	1	301	151	1131	77	224
18	4	7	3	12	11	2	339	151	1182	109	300
19	4	13	3	14	14	1	669	183	2165	134	380
20	4	11	3	12	9	1	466	153	1559	107	300
21	4	10	4	18	16	1	927	237	2936	172	490
22	2	3	4	20	18	0	628	223	2149	126	359
23	3	8	4	18	15	0	895	225	2824	144	419
24	3	6	4	18	14	0	822	227	2596	159	447
25	2	5	4	18	17	2	615	201	2097	114	326
26	2	6	4	19	21	2	622	207	2159	112	321
27	2	2	4	23	26	1	624	249	2217	127	374
28	3	7	4	19	17	1	1027	241	3192	157	459
29	3	10	4	18	18	1	1057	231	3282	153	449
30	4	12	4	19	19	1	1132	255	3516	183	530

Table 4.3: Description of the solutions found and effort measurements of the IBM ILOG CPLEX and IBM ILOG CP Optimizer applied to the thirty small-sized instances.

	IBM ILOG CPLEX				IBM ILOG CP Optimizer			
	Makespan	Effort measurement			Makespan	Effort measurement		
		# iterations	# B&B Nodes	CPU		# of branches in phase 1	# of branches in phase 2	CPU
1	274	9286	824	0.5	274	299	21	0.1
2	230	7302	705	0.4	230	215	3	0.1
3	337	2451	231	0.2	337	66	3	0.1
4	276	6410	444	0.4	276	179	21	0.1
5	495	6444	763	0.3	495	212	21	0.1
6	271	5645	665	0.3	271	193	62	0.1
7	370	7223	561	0.3	370	203	580	0.1
8	279	3263	114	0.3	279	98	17	0.1
9	274	1261	113	0.1	274	8	19	0.1
10	329	8530	357	0.7	329	1148	30	0.1
11	239	649022	21705	56.6	239	5274	3	0.2
12	273	54160	2870	3.9	273	410	3	0.1
13	266	2464645	79179	176.0	266	14835	10038	1.0
14	518	401863	20863	42.4	518	1669	29	0.1
15	551	1826080	75865	153.8	551	14824	95573	1.2

continues on next page

Table 4.3 continued

	IBM ILOG CPLEX				IBM ILOG CP Optimizer			
	Makespan	Effort measurement			Makespan	Effort measurement		
		# iterations	# B&B Nodes	CPU		# of branches in phase 1	# of branches in phase 2	CPU
16	278	21274	1340	1.4	278	361	29	0.1
17	540	108667	8321	7.7	540	14516	3	0.2
18	327	4623	441	0.3	327	12	27	0.1
19	325	248850	11251	16.5	325	1665	990	0.1
20	264	16287	826	1.1	264	1500	179	0.1
21	[263, 300] 12.3%	50935466	1465096	7200	300	21643	11865	0.7
22	[204, 671] 69.6%	73458018	1400247	7200	651	220130	3	8.0
23	[406, 467] 13.1%	56974881	1217720	7200	467	19771	3	0.6
24	[336, 572] 41.3%	80022693	1673003	7200	571	72971	3	2.6
25	[436, 672] 35.1%	62426552	2011692	7200	672	76887	110821	3.0
26	[374, 628] 40.4%	88226422	1926203	7200	627	226752	324106	12.5
27	[223, 719] 69.0%	79180995	1529405	7200	702	498600	927317	63.7
28	[297, 493] 39.8%	62733982	707445	7200	437	28733	3	1.4
29	[343, 498] 31.1%	55996388	1397887	7200	480	100627	39	4.0
30	[417, 420] 0.7%	65757996	1158447	7200	420	8148	10885	0.3
Mean	405.2 11.75%	22718889.3	490486.1	2415.44	401.43 0.0%	44398.3	49756.53	3.37

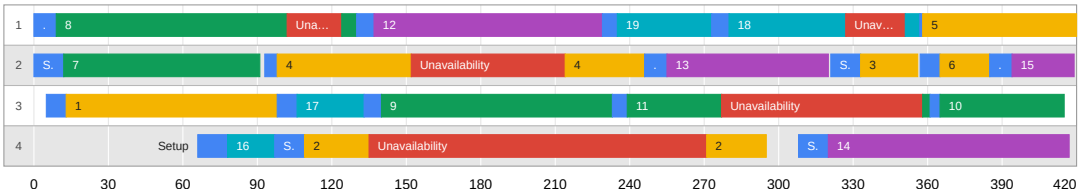


Figure 4.4: Optimal solution to instance 30 of the set of small-sized instances. Operations with the same color belong to the same job; while setups are represented in blue and machines' unavailiability periods in red.

4.4.3 Experiments With Medium-Sized Instances

In this section, a set of 20 medium-sized instances is considered. The k-th instance was generated with the following parameters: $n = 4+\lceil \frac{k}{20} \times 6 \rceil$, $o_{\min} = 6$, $o_{\max} = 7+\lceil \frac{k}{20} \times 5 \rceil$, $m_{\min} = 6$, $m_{\max} = 7 + \lceil \frac{k}{20} \times 13 \rceil$, and $q = 8$. Table 4.4 shows the main characteristics of each instance.

As described at the end of Section 4.2, the CP Optimizer formulation of the OPS scheduling problem is given by the minimization of (4.33) subject to constraints (4.35–4.39, 4.42, 4.43–4.49, 4.52–4.63); with constraints (4.41) and (4.50, 4.51) being optional. Therefore, in a first experiment, I aim to evaluate the influence of the optional constraints by comparing: (i) the plain model (named CP Model 1 from on); (ii) the plain model plus constraints (4.41) (named CP Model 2 from on); (iii) the plain model plus constraints (4.50, 4.51) (named CP Model 3 from on); and (iv) the plain model plus constraints (4.41) and constraints (4.50, 4.51) (named CP Model 4 from on). Table 4.5 shows the value of the solutions found when solving each of the four CP Optimizer models with IBM ILOG CP Optimizer solver. The associated effort measurements are also shown in the table. Figures in the table show that considering the optional constraints (4.50, 4.51) helps CP Optimizer solver to close the gap and prove optimality in 12 out of the 20 considered instances when solving CP Models 3 and 4; while, when solving CP Models 1 and 2, that do not consider the optional constraints (4.50, 4.51), gaps are closed in 10 out of the 20 considered instances. On the other hand, including or not the optional constraints (4.41) appears to have no relevant influence on the resolution process of the considered set of instances. Figure 4.5 shows the average value of the makespan, over the 20 considered instances, as a function of time, over the resolution of CP Models 1–4. The graphic confirms, as expected, that, CP Optimizer solver is able to improve the incumbent solution faster when applied to CP Models 3 and 4 compared to its application to CP Models 1 and 2.

Table 4.4: Main features of the considered twenty medium-sized instances.

Main instance characteristics							MILP formulation			CP Optimizer formulation	
Instance	m	$\sum_{k=1}^{m} q_k$	n	o	$\|A\|$	$\|T\|$	# binary variables	# continuous variables	# constraints	# integer variables	# constraints
1	8	41	5	39	58	0	5987	635	16984	584	1671
2	7	28	5	36	54	3	4341	559	12398	471	1381
3	8	46	5	43	75	1	7753	717	21670	643	1895
4	7	27	6	43	60	1	6615	693	18246	595	1749
5	6	31	6	45	61	3	6189	659	17641	512	1519
6	7	38	6	46	60	0	7292	715	20646	579	1725
7	8	31	7	64	108	1	12302	1049	33289	929	2731
8	9	51	7	53	76	1	11814	933	32185	860	2524
9	6	25	7	56	90	2	9139	839	25240	677	2011
10	8	31	7	63	110	2	12993	1057	34805	932	2777
11	13	52	8	75	118	0	26493	1615	67250	1693	4962
12	11	56	8	78	142	2	26370	1535	68106	1554	4543
13	16	75	8	68	105	0	30571	1713	77029	1949	5652
14	12	61	9	72	105	0	26915	1533	68895	1610	4685
15	17	73	9	76	107	0	37144	1965	92371	2239	6508
16	15	67	9	89	156	2	42871	2113	106337	2287	6756
17	7	34	10	109	207	1	32729	1723	86417	1460	4362
18	17	72	10	96	164	1	53876	2457	131704	2827	8190
19	14	72	10	92	142	2	44376	2095	110957	2273	6617
20	11	51	10	91	135	0	36772	1843	92819	1850	5439

Table 4.5: Description of the solutions found and effort measurements of the IBM ILOG CP Optimizer applied to the twenty medium-sized instances.

	CP Model 1			CP Model 2			CP Model 3			CP Model 4		
	Makespan	Effort measurement		Makespan	Effort measurement		Makespan	Effort measurement		Makespan	Effort measurement	
		# branches	CPU		# branches	CPU		# branches	CPU		# branches	CPU
1	344	11819616	599.7	344	4945664	499.0	344	368683	16.1	344	399038	32.5
2	357	129666	12.0	357	112211	11.3	357	35850	3.9	357	34779	3.8
3	404	45163467	4315.8	[361, 409] 11.7%	77088019	7200	[361, 406] 11.1%	88706470	7200	404	18733373	2825.1
4	458	17065512	1081.0	458	7777910	531.7	458	269785	25.4	458	548398	49.5
5	[474, 510] 7.1%	112252005	7200	[474, 515] 8.0%	86834261	7200	506	384590	40.8	506	399229	90.5
6	[329, 454] 27.5%	86652702	7200	[334, 438] 23.7%	84114310	7200	[334, 437] 23.6%	80406442	7200	[335, 442] 24.2%	47105274	7200
7	2429	1262	0.2	2429	1272	0.2	2429	1455	0.2	2429	1488	0.2
8	[360, 456] 21.1%	89059560	7200	[360, 451] 20.2%	64334715	7200	[360, 460] 21.7%	65621010	7200	[360, 452] 20.4%	59032373	7200
9	[629, 631] 0.3%	118721872	7200	[629, 636] 1.1%	34808015	7200	[629, 630] 0.2%	94392736	7200	[629, 630] 0.2%	86015687	7200
10	1184	1079	0.1	1184	1052	0.1	1184	1204	0.2	1184	1189	0.3
11	[406, 430] 5.6%	39491824	7200	[406, 431] 5.8%	53721744	7200	[406, 427] 4.9%	48553125	7200	[406, 424] 4.2%	45050481	7200
12	[457, 508] 10.0%	75583273	7200	[457, 510] 10.4%	76775244	7200	[457, 503] 9.1%	57651439	7200	[457, 505] 9.5%	48316050	7200
13	347	148874	26.5	347	144200	15.8	347	64381	7.7	347	29387	6.6
14	[302, 412] 26.7%	43670920	7200	[320, 404] 20.8%	57130518	7200	[302, 399] 24.3%	48125327	7200	[320, 403] 20.6%	33552405	7200
15	319	259273	40.6	319	275642	58.8	319	55919	12.4	319	94552	27.6
16	543	138290	18.0	543	1814	0.5	543	1343	0.4	543	1331	0.8
17	[1044, 1053] 0.9%	60769510	7200	[1044, 1059] 1.4%	48747654	7200	1052	12862072	1589.7	[1052, 1055] 0.3%	32700611	7200
18	3184	1968	0.6	3184	1968	0.6	3184	2091	0.8	3184	2091	0.7
19	[1449, 1451] 0.1%	93263868	7200	[1449, 1451] 0.1%	83339741	7200	1451	2305	0.7	1451	2371	0.7
20	[360, 544] 33.8%	71372646	7200	[417, 543] 23.2%	51052068	7200	[360, 532] 32.3%	43254587	7200	[417, 532] 21.6%	41945390	7200
Mean	800.9 6.65%	43278359	3904.72	800.6 6.32%	36560401	4015.9	798.4 6.36%	27038040	2964.91	798.45 5.05%	20698274	3031.91

Table 4.6: Description of the solutions found and effort measurements of the IBM ILOG CP Optimizer applied to the twenty medium-sized instances of CP Model 4 without using (already shown in Table 4.5) and using the solution of the Incomplete CP Model as an initial guess.

	CP Model 4			Incomplete CP Model + CP Model 4					
		Effort measurement			Effort measurement				
	Makespan	# branches	CPU	Makespan	phase 1		phase 2		Total
					# branches	CPU	# branches	CPU	CPU
1	344	399038	32.5	344	288898	15.53	280015	15.07	30.6
2	357	34779	3.8	357	22228	1.09	3	0.01	1.1
3	404	18733373	2825.1	404	132017215	4969.69	16246793	611.61	5581.3
4	458	548398	49.5	458	146895	5.59	3	0.01	5.6
5	506	399229	90.5	506	272817	31.26	294203	33.74	65.0
6	[335, 442] 24.2%	47105274	7200	[335, 441] 24.0%	142959186	4800.0	21538650	2400.0	7200
7	2429	1488	0.2	2429	1241	0.08	131	0.02	0.1
8	[360, 452] 20.4%	59032373	7200	[360, 450] 20.0%	129153428	4800.0	19362459	2400.0	7200
9	[629, 630] 0.2%	86015687	7200	629	43101316	1096.49	115	0.01	1096.5
10	1184	1189	0.3	1184	1250	0.09	3	0.01	0.1
11	[406, 424] 4.2%	45050481	7200	[406, 418] 2.9%	123558867	4800.0	18454514	2400.0	7200
12	[457, 505] 9.5%	48316050	7200	[457, 499] 8.4%	128774356	4800.0	17042168	2400.0	7200
13	347	29387	6.6	347	6914	0.39	3	0.01	0.4
14	[320, 403] 20.6%	33552405	7200	[320, 394] 18.8%	109752381	4800.0	13032611	2400.0	7200
15	319	94552	27.6	319	47107	3.19	3	0.01	3.2
16	543	1331	0.8	543	1481	0.29	3	0.01	0.3
17	[1052, 1055] 0.3%	32700611	7200	1052	572991	59.72	455352	47.48	107.2
18	3184	2091	0.7	3184	2063	0.39	3	0.01	0.4
19	1451	2371	0.7	1451	2504	0.36	187	0.04	0.4
20	[417, 532] 21.6%	41945390	7200	[417, 520] 19.8%	106764402	4800.0	12752639	2400.0	7200
Mean	798.45 5.05%	20698274	3031.91	796.45 4.7%	45872377	1749.21	5972992	755.4	2504.61

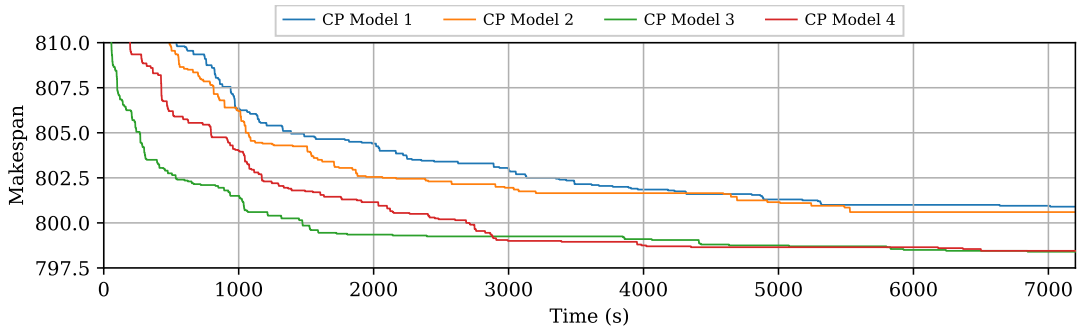


Figure 4.5: Evolution over time of the (average of the) incumbent solutions' makespan of medium-sized instances along the resolution of CP Models 1–4.

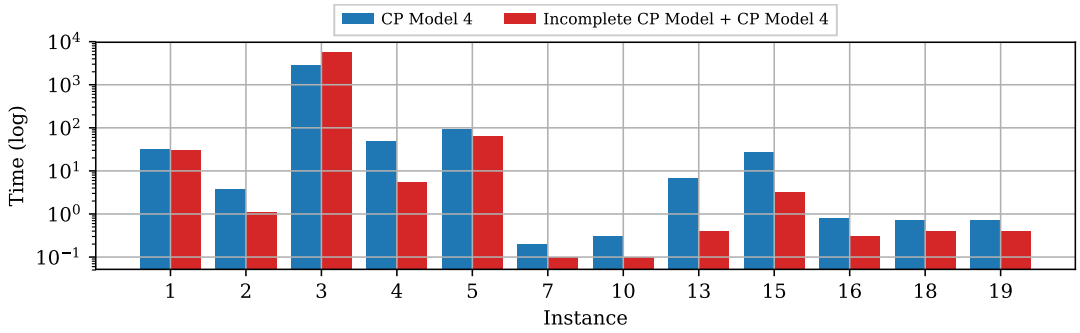


Figure 4.6: Time comparison between the resolution of the CP Model 4 and its resolution in two phases considering the solution found to the Incomplete CP Model as an initial guess.

As described in Section 4.2.3, modeling a setup in between consecutive operations is relatively easy in the CP Optimizer language. However, the simple formulation given by constraints (4.50, 4.51) does not consider the setup time of the first operation processed by each machine; as well as it does not consider that setup operations are not resumable, i.e., they can not be interrupted by periods of unavailability of the machines. This is why the model that consists in minimizing (4.33) subject to constraints (4.35–4.39, 4.42, 4.43–4.51) can be considered an incomplete model of the OPS scheduling problem, and I name it Incomplete CP Model from now on. The Incomplete CP Model is much simpler that CP Model 4 and, as a resolution strategy, it can be solved first; and its optimal (or best known feasible) solution used as an initial guess to the resolution of CP Model 4. This two-phases strategy is based on the fact that IBM ILOG CP Optimizer solver has the capability of accepting as initial guess a possible infeasible and incomplete solution. By incomplete I mean that CP Model 4 has variables that are not present in the Incomplete CP Model; and, by infeasible, I mean that, with very high probability, a solution to the Incomplete CP Model has setup operations being interrupted by machines' unavailabilities, as well as it does not consider the setup operation of the first operation being processed by each machine; thus being infeasible to CP Model 4. Nevertheless, IBM ILOG CP Optimizer solver is able to heuristically transform this infeasible and incomplete initial guess into a feasible solution that potentially helps to prune the search space

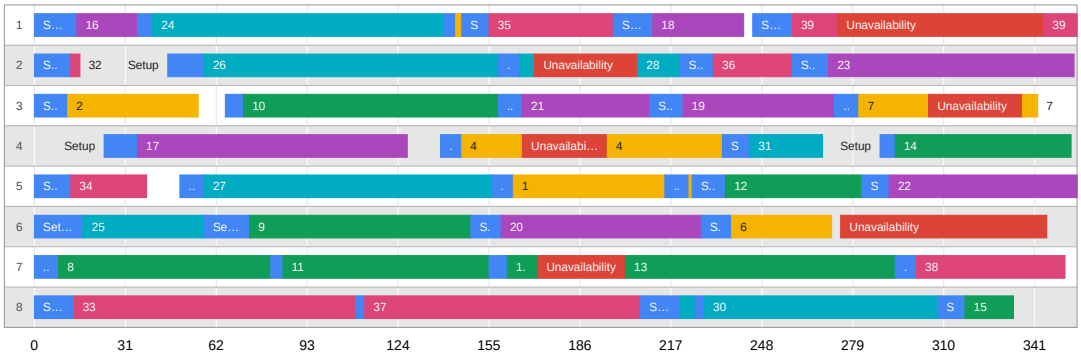


Figure 4.7: Optimal solution to instance 1 of the set of medium-sized instances. Operations with the same color belong to the same job; while setups are represented in blue and machines' unavailiability periods in red.

in the resolution process of CP Model 4. Table 4.6 shows the solutions found by IBM ILOG CP Optimizer solver when applied to the twenty medium-sized instances of CP Model 4 without using (already shown in Table 4.5) and using the solution of the Incomplete CP Model as an initial guess. In the two-phases strategy, 2/3 of the two hours budget is allocated to the resolution of the Incomplete CP Model; while the remaining 1/3 is allocated to the resolution of CP Model 4 itself. In the table, "# branches 1" corresponds to the resolution of the Incomplete CP Model; while "# branches 2" corresponds to the resolution of CP Model 4. The CPU time reported for "Incomplete CP Model + CP Model 4" corresponds to the total CPU time for solving both models. Figures in the table show that, in instance 3, the two-phases strategy took longer than the "single-phase strategy" to close the gap. On the other hand, it closed the gaps of instances 9 and 17 (that were not closed by the single-phase strategy); and it reduced the gap, by improving the incumbent solution, in the other six instances in which the single-phase strategy was unable to prove optimality (namely, instances 6, 8, 11, 12, 14, and 20). Moreover, as depicted in Figure 4.6, the two-phases strategy was able to prove optimality faster than the single-phase strategy on 11 out of the 12 instances in which both strategies proved optimality. As a whole, it can be inferred that the two-phases strategy for solving CP Model 4 is the most efficient way of solving the CP Optimizer formulation of the OPS scheduling problem. The figures reported in Table 4.3, where the performances of IBM ILOG CPLEX and IBM ILOG CP Optimizer solvers are compared when applied to the MILP and the CP optimizer formulations of the OPS scheduling problem, respectively, correspond to the two-phases strategy applied to CP Model 4. As an illustration, Figure 4.7 shows the solution to instance 1 of the set of medium-sized instances found by the two-phases strategy applied to CP Model 4.

4.4.4 Experiments With Large-Sized Instances

In this section, a set of 50 large-sized instances is considered. Instances are of the size of the real instances of the OPS scheduling problem. The k-th instance was generated with the following parameters: $n = 11 + \lceil \frac{k}{100} \times 189 \rceil$, $o_{\min} = 5$, $o_{\max} = 6 + \lceil \frac{k}{100} \times 14 \rceil$, $m_{\min} = 9 + \lceil \frac{k}{100} \times 20 \rceil$, $m_{\max} = 10 + \lceil \frac{k}{100} \times 90 \rceil$, and $q = 8$. Table 4.7 shows the main characteristics of each instance. Note that the last instance in the set has 55 machines, 250 unavailability periods, 106 jobs, 978 operations, and 1,581 precedence constraints; while its CP formulation has 85,463 integer variables and 251,649 constraints. Table 4.8 shows the solutions found by and the performance of IBM ILOG CP Optimizer applied to CP Model 4 and to "Incomplete CP Model + CP Model 4". In the table, only the number of branches is shown, since in all case the CPU time limit of two hours was reached. (Recall that, in the case in which the two phases strategy is considered, 2/3 of the time is devoted to solve the Incomplete CP Model and the remaining 1/3 of the time is used to solve CP Model 4 using as starting guess the solution to the Incomplete CP Model.) Figures in the table show that, as in the case of the medium-sized instances, the two phases strategy is more effective, in the sense that it is able to found better quality solutions in 41 out of the 50 considered instances. Figures in the table also show that in all cases a feasible solution is found and that the average gap is around 33%. Table 4.9 shows the solution found by IBM ILOG CP Optimizer applied to "Incomplete CP Model + CP Model 4" with a CPU time limit of 5 minutes, 30 minutes, 2 hours, and 10 hours. The table shows that the average gap for the different CPU time limits is 41%, 34%, 33%, and 31%, respectively. These figures suggest that "good quality" solutions are found relatively quickly, that most of the time is used to close the gap, and that, when trying to close the gap, slightly better solutions might be found. This means that IBM ILOG CP Optimizer could be used in practice to find feasible solutions to real instances of the OPS scheduling problem. On the other hand, the non-null gap leaves space for the development of ad-hoc heuristic methods that, hopefully, would be able to find better quality solutions.

Table 4.7: Main features of the considered fifty large-sized instances.

	Main instance characteristics						CP Optimizer formulation	
Instance	m	$\sum_{k=1}^{m} q_k$	n	o	$\|A\|$	$\|T\|$	# integer variables	# constraints
1	10	56	13	79	95	0	1293	3792
2	10	44	15	98	120	0	1595	4648
3	13	47	17	107	131	0	2359	6888
4	11	69	19	120	150	0	2451	7209
5	10	52	21	128	165	0	2235	6540
6	15	64	23	141	180	0	3820	11235
7	14	71	25	170	229	0	4120	12054
8	13	58	27	193	257	0	4655	13631
9	16	68	29	207	308	1	5592	16522
10	19	81	30	201	266	0	6537	19152
11	14	68	32	240	337	1	5938	17404
12	15	67	34	232	310	0	6124	18115
13	18	82	36	257	350	0	7860	23081
14	13	60	38	259	350	0	5883	17357
15	15	53	40	298	430	1	7708	22820
16	17	88	42	311	429	0	9140	26801
17	22	98	44	341	505	1	12788	37473
18	13	57	46	345	486	1	7853	23212
19	24	108	47	362	524	0	14367	42184
20	28	136	49	383	555	1	17662	52005
21	14	69	51	386	557	0	9544	28138
22	20	79	53	398	572	0	12750	37494
23	28	129	55	421	638	0	19538	57451
24	15	61	57	437	592	1	11299	33312
25	29	145	59	507	810	1	24190	70876

continues on next page

Table 4.7 continued

	Main instance characteristics						CP Optimizer formulation	
Instance	m	$\sum_{k=1}^{m} q_k$	n	o	$\|A\|$	$\|T\|$	# integer variables	# constraints
26	26	120	61	487	743	0	21193	62377
27	21	93	63	498	746	0	17571	51678
28	20	66	64	533	784	2	17039	50072
29	19	59	66	547	833	0	17555	51928
30	26	130	68	563	831	0	24305	71287
31	37	148	70	601	967	1	35633	105024
32	39	154	72	593	935	0	37521	110493
33	20	90	74	617	931	0	20027	58960
34	35	161	76	656	1011	0	37094	108969
35	17	71	78	675	1049	1	19362	57160
36	19	81	80	700	1127	1	22642	66713
37	43	197	81	789	1290	1	55235	162562
38	43	186	83	706	1143	1	49300	145476
39	32	137	85	765	1242	1	40211	118131
40	26	93	87	812	1305	0	34800	102295
41	27	135	89	834	1383	0	36985	109069
42	38	152	91	821	1282	2	50074	147194
43	26	93	93	904	1424	1	38750	114131
44	18	89	95	921	1564	0	28031	82872
45	34	164	97	965	1628	0	53196	156446
46	28	127	98	868	1367	1	39853	117804
47	19	87	100	961	1609	1	30667	90453
48	53	228	102	942	1540	0	78706	232310
49	23	101	104	959	1513	0	36885	108542
50	55	250	106	978	1581	0	85463	251649

Table 4.8: Description of the solutions found and effort measurements of the IBM ILOG CP Optimizer applied to the fifty large-sized instances.

Instance	CP Model 4			Incomplete CP Model + CP Model 4			
	Makespan	gap (%)	# branches	makespan	gap (%)	# branches 1	# branches 2
1	[387, 528]	26.7	49798163	[387, 527]	26.6	120241891	14581110
2	[494, 661]	25.3	39711504	[494, 650]	24.0	110885682	13313040
3	[452, 631]	28.4	39468386	[452, 633]	28.6	97607050	11604473
4	[562, 763]	26.3	35944769	[562, 756]	25.7	93096578	9904719
5	[625, 855]	26.9	39251314	[625, 828]	24.5	97073106	10349840
6	[490, 717]	31.7	32448937	[490, 715]	31.5	85024768	9228219
7	[659, 943]	30.1	28100210	[659, 923]	28.6	72857628	7788597
8	[739, 1055]	30.0	23528650	[739, 1039]	28.9	67355598	6681507
9	[654, 983]	33.5	24454758	[654, 980]	33.3	65804340	6537839
10	[547, 827]	33.9	25762983	[547, 790]	30.8	63801015	6740620
11	[857, 1244]	31.1	19205336	[857, 1230]	30.3	57729625	7599655
12	[838, 1225]	31.6	21224292	[838, 1180]	29.0	61503923	6541171
13	[699, 1073]	34.9	20602257	[699, 1011]	30.9	54006978	5190138
14	[1052, 1539]	31.6	18026217	[1052, 1486]	29.2	50939574	4958123
15	[975, 1478]	34.0	16007825	[975, 1445]	32.5	42382885	4677113
16	[924, 1488]	37.9	13721997	[924, 1382]	33.1	38344783	3879028
17	[763, 1150]	33.7	11518247	[763, 1117]	31.7	32621308	2787395
18	[1415, 2040]	30.6	11688840	[1415, 1897]	25.4	36621269	3282986
19	[737, 1107]	33.4	12841773	[737, 1028]	28.3	31061348	3454907
20	[671, 1093]	38.6	7883338	[671, 1053]	36.3	26628577	3368228
21	[1378, 2043]	32.6	9278464	[1378, 1956]	29.6	38050942	3605124
22	[985, 1563]	37.0	11218731	[985, 1496]	34.2	37444198	2991893
23	[762, 1185]	35.7	8102079	[762, 1146]	33.5	29827277	3268703
24	[1377, 2162]	36.3	7139492	[1377, 2010]	31.5	37578271	2841939
25	[892, 1397]	36.1	4160924	[892, 1367]	34.7	20504018	1181601

continues on next page

Table 4.8 continued

Instance	CP Model 4			Incomplete CP Model + CP Model 4			
	Makespan	gap (%)	# branches	makespan	gap (%)	# branches 1	# branches 2
26	[880, 1463]	39.8	7210391	[880, 1362]	35.4	26867887	1847124
27	[1246, 1856]	32.9	8158263	[1246, 1790]	30.4	29586547	2630271
28	[1396, 2175]	35.8	7900895	[1396, 2089]	33.2	27240976	1642209
29	[1452, 2241]	35.2	4952743	[1452, 2199]	34.0	28329290	1889844
30	[1116, 1692]	34.0	8890534	[1116, 1769]	36.9	21953739	1703439
31	[822, 1284]	36.0	4139996	[822, 1218]	32.5	21050620	1652157
32	[776, 1204]	35.5	3423918	[776, 1151]	32.6	19731639	2487825
33	[1469, 2414]	39.1	4898465	[1469, 2276]	35.5	24557450	2204832
34	[951, 1490]	36.2	5776499	[951, 1483]	35.9	16280020	1379212
35	[1932, 3161]	38.9	5417895	[1932, 3049]	36.6	25096957	1948660
36	[1788, 2876]	37.8	4898721	[1788, 2826]	36.7	23458076	379399
37	[924, 1426]	35.2	2985158	[924, 1468]	37.1	11587937	821564
38	[832, 1281]	35.1	5009442	[832, 1192]	30.2	14892180	1195237
39	[1214, 1895]	35.9	2483387	[1214, 1845]	34.2	16606096	1193648
40	[1552, 2439]	36.4	3344183	[1552, 2439]	36.4	16202692	1411237
41	[1587, 2553]	37.8	3239079	[1587, 2524]	37.1	15517269	873936
42	[1111, 1905]	41.7	1837052	[1111, 1715]	35.2	14722059	1154822
43	[1737, 2779]	37.5	4525245	[1737, 2767]	37.2	16091655	1129221
44	[2587, 4007]	35.4	3260320	[2587, 3908]	33.8	13253041	1121671
45	[1446, 2331]	38.0	5837368	[1446, 2301]	37.2	14297375	1145106
46	[1539, 2432]	36.7	3434877	[1539, 2446]	37.1	15226438	1126198
47	[2518, 3928]	35.9	2421741	[2518, 4040]	37.7	14957027	961575
48	[896, 1368]	34.5	925320	[896, 1473]	39.2	6202106	895555
49	[2108, 3187]	33.9	3352348	[2108, 3233]	34.8	17104498	979735
50	[931, 1487]	37.4	1857144	[931, 1505]	38.1	13503241	927131
Mean	1692.48	34.4	12825409	1654.26	32.8	38666188	3821191

Table 4.9: Description of the solutions found by the IBM ILOG CP Optimizer applied to the fifty large-sized instances, considering the Incomplete CP Model + CP Model 4 strategy, and with increasing CPU time limits.

	5 minutes		30 minutes		2 hours		10 hours	
	Makespan	gap (%)	Makespan	gap (%)	Makespan	gap (%)	Makespan	gap (%)
1	[387, 538]	28.1	[387, 530]	27.0	[387, 527]	26.6	[387, 521]	25.7
2	[494, 663]	25.5	[494, 654]	24.5	[494, 650]	24.0	[494, 649]	23.9
3	[452, 653]	30.8	[452, 635]	28.8	[452, 633]	28.6	[452, 623]	27.4
4	[562, 780]	27.9	[562, 755]	25.6	[562, 756]	25.7	[562, 756]	25.7
5	[625, 860]	27.3	[625, 837]	25.3	[625, 828]	24.5	[625, 825]	24.2
6	[490, 724]	32.3	[490, 718]	31.8	[490, 715]	31.5	[490, 698]	29.8
7	[659, 964]	31.6	[659, 938]	29.7	[659, 923]	28.6	[659, 902]	26.9
8	[739, 1091]	32.3	[739, 1044]	29.2	[739, 1039]	28.9	[739, 1022]	27.7
9	[654, 1019]	35.8	[654, 966]	32.3	[654, 980]	33.3	[654, 936]	30.1
10	[547, 902]	39.4	[547, 900]	39.2	[547, 790]	30.8	[547, 781]	30.0
11	[857, 1290]	33.6	[857, 1231]	30.4	[857, 1230]	30.3	[857, 1182]	27.5
12	[838, 1257]	33.3	[838, 1223]	31.5	[838, 1180]	29.0	[838, 1178]	28.9
13	[699, 1084]	35.5	[699, 1010]	30.8	[699, 1011]	30.9	[699, 998]	30.0
14	[1052, 1557]	32.4	[1052, 1514]	30.5	[1052, 1486]	29.2	[1052, 1449]	27.4
15	[975, 1542]	36.8	[975, 1476]	33.9	[975, 1445]	32.5	[975, 1427]	31.7
16	[924, 1516]	39.1	[924, 1420]	34.9	[924, 1382]	33.1	[924, 1343]	31.2
17	[763, 1131]	32.5	[763, 1080]	29.4	[763, 1117]	31.7	[763, 1037]	26.4
18	[1415, 2014]	29.7	[1415, 1918]	26.2	[1415, 1897]	25.4	[1415, 1898]	25.4
19	[737, 1236]	40.4	[737, 1046]	29.5	[737, 1028]	28.3	[737, 1008]	26.9
20	[671, 1135]	40.9	[671, 1132]	40.7	[671, 1053]	36.3	[671, 1050]	36.1
21	[1378, 2104]	34.5	[1378, 1992]	30.8	[1378, 1956]	29.6	[1378, 1939]	28.9
22	[985, 1639]	39.9	[985, 1642]	40.0	[985, 1496]	34.2	[985, 1573]	37.4
23	[762, 1336]	43.0	[762, 1128]	32.4	[762, 1146]	33.5	[762, 1082]	29.6
24	[1377, 2135]	35.5	[1377, 2073]	33.6	[1377, 2010]	31.5	[1377, 1942]	29.1
25	[892, 1524]	41.5	[892, 1442]	38.1	[892, 1367]	34.7	[892, 1241]	28.1

continues on next page

Table 4.9 continued

	5 minutes		30 minutes		2 hours		10 hours	
	Makespan	gap (%)	Makespan	gap (%)	Makespan	gap (%)	Makespan	gap (%)
26	[880, 1581]	44.3	[880, 1486]	40.8	[880, 1362]	35.4	[880, 1348]	34.7
27	[1246, 1833]	32.0	[1246, 1791]	30.4	[1246, 1790]	30.4	[1246, 1741]	28.4
28	[1396, 2185]	36.1	[1396, 2171]	35.7	[1396, 2089]	33.2	[1396, 2003]	30.3
29	[1452, 2256]	35.6	[1452, 2298]	36.8	[1452, 2199]	34.0	[1452, 2282]	36.4
30	[1116, 2473]	54.9	[1116, 1590]	29.8	[1116, 1769]	36.9	[1116, 1561]	28.5
31	[822, 1296]	36.6	[822, 1295]	36.5	[822, 1218]	32.5	[822, 1288]	36.2
32	[434, 1214]	64.3	[776, 1208]	35.8	[776, 1151]	32.6	[776, 1165]	33.4
33	[1469, 2698]	45.6	[1469, 2398]	38.7	[1469, 2276]	35.5	[1469, 2166]	32.2
34	[409, 1545]	73.5	[951, 1604]	40.7	[951, 1483]	35.9	[951, 1403]	32.2
35	[1932, 3145]	38.6	[1932, 3099]	37.7	[1932, 3049]	36.6	[1932, 2892]	33.2
36	[1788, 3167]	43.5	[1788, 2819]	36.6	[1788, 2826]	36.7	[1788, 2691]	33.6
37	[924, 1463]	36.8	[924, 1671]	44.7	[924, 1468]	37.1	[924, 1350]	31.6
38	[832, 1264]	34.2	[832, 1200]	30.7	[832, 1192]	30.2	[832, 1144]	27.3
39	[452, 1880]	76.0	[1214, 1870]	35.1	[1214, 1845]	34.2	[1214, 1786]	32.0
40	[1552, 2463]	37.0	[1552, 2509]	38.1	[1552, 2439]	36.4	[1552, 2347]	33.9
41	[1587, 2550]	37.8	[1587, 2522]	37.1	[1587, 2524]	37.1	[1587, 2542]	37.6
42	[618, 1732]	64.3	[1111, 1688]	34.2	[1111, 1715]	35.2	[1111, 1598]	30.5
43	[529, 2784]	81.0	[1737, 2779]	37.5	[1737, 2767]	37.2	[1737, 2599]	33.2
44	[2587, 4030]	35.8	[2587, 4063]	36.3	[2587, 3908]	33.8	[2587, 3770]	31.4
45	[1446, 2378]	39.2	[1446, 2466]	41.4	[1446, 2301]	37.2	[1446, 2210]	34.6
46	[1539, 2439]	36.9	[1539, 2497]	38.4	[1539, 2446]	37.1	[1539, 2293]	32.9
47	[2518, 4195]	40.0	[2518, 3757]	33.0	[2518, 4040]	37.7	[2518, 3824]	34.2
48	[544, 1377]	60.5	[896, 1506]	40.5	[896, 1473]	39.2	[896, 1290]	30.5
49	[2108, 4065]	48.1	[2108, 3012]	30.0	[2108, 3233]	34.8	[2108, 2897]	27.2
50	[931, 2244]	58.5	[931, 1570]	40.7	[931, 1505]	38.1	[931, 1478]	37.0
Mean	1779.02	41.0	1683.46	34.1	1654.26	32.8	1594.56	30.6

4.5 Summary

In this chapter, MILP and CP formulations of the OPS scheduling problem were presented and analyzed. The capacity of IBM ILOG CPLEX MILP and IBM ILOG CP Optimizer solvers for dealing with the proposed formulations was investigated. While the MILP solver is a general-purpose solver; the CP Optimizer solver was born as a solver dedicated to scheduling problems, with its own modeling language that fully explores the structure of the underlying problem. Thus, it is not a surprise the latter to outperform the former by a large extent in the numerical experiments. The obtained result is in agreement with a previous comparison of similar nature presented in (Vilím, Laborie, and Shaw, 2015) and (Laborie, 2018) for the JS and the FJS scheduling problems.

The OPS scheduling problem is an NP-hard problem and the CP Optimizer mixes exact and heuristic strategies that were not specifically devised for the problem at hand. However, numerical experiments in this chapter show that the CP Optimizer is able to find feasible solution to large-sized instances; thus being an alternative to tackle the OPS scheduling problem in practice. On the other hand, it is not expected the CP Optimizer to be competitive with ad-hoc heuristics fully exploiting the specificities of the problem. These facts suggest that the development of heuristic methods to deal with the considered problem is a promising alternative; and this will be the subject of future work.

Chapter 5

Constructive Heuristic, Local Search Heuristic, and Metaheuristics

In this chapter, heuristic methods for the OPS scheduling problem are presented. First, the LS strategy introduced by Mastrolilli and Gambardella (2000), to deal with the FJS scheduling problem, is extended. The interpretation of the LS lies on the representation of the operations' precedences as a graph in which the makespan is given by the longest path from the "source" to the "target" node. The underlying graph is extended to cope with the sequencing flexibility and, more relevant, with resumable operations and machines' downtimes. With the help of the redefined graph, the main idea in (Mastrolilli and Gambardella, 2000), which consists in defining reduced neighbor sets, is also extended. The reduction of the neighborhood, that greatly speed up the LS procedure, relies on the fact that the reduction of the makespan of the current solution requires the reallocation of an operation in a *critical path*, i.e., a path that realizes the makespan. With all these ingredients, a LS for the OPS scheduling problem is proposed. To enhance the probability of finding better quality solutions, the LS procedure is embedded in metaheuristic frameworks. A relevant ingredient of the metaheuristic is the representation of a solution with two arrays of real numbers of the size of the number of non-fixed operations. One of the arrays represents the assignment of non-fixed operations to machines; while the other represents the sequencing of the non-fixed operations within the machines. The considered representation is an indirect representation, i.e., it does not encode a complete solution. Thus, another relevant ingredient is the development of a *decoder*, i.e., a methodology to construct a feasible solution from the two arrays. One of the challenging tasks of the decoder is to sequence the fixed operations, besides constructing a feasible semi-active scheduling. The representation scheme, the decoder, and the LS strategy are evaluated in connection with four metaheuristics. Two of the metaheuristics, GA and DE

are population-based methods; while the other two, ILS and TS, are trajectory methods. Since the proposed GA and DE include a LS, they can be considered memetic algorithms.

The chapter is structured as follows. Section 5.1 introduces the way in which the two key elements of a solution (assignment of operations to machines and sequencing within the machines) are represented and how a feasible solution is constructed from them. Section 5.2 introduces the proposed LS. The considered metaheuristic approaches are given in Section 5.3. Numerical experiments are presented and analyzed in Section 5.4. Final remarks and conclusions are given in the last section.

5.1 Representation Scheme and Construction of a Feasible Solution

In this section, it is described: (a) the way the assignment of non-fixed operations to machines is represented, (b) the way the sequence of non-fixed operations assigned to each machine is represented, and (c) the way a feasible solution is constructed from these two representations. it is assumed that all numbers that define an instance of the OPS scheduling problem are integer numbers. Namely, it is assumed that the processing times p_{ik} ($i \in V$, $k \in F(i)$), the release times r_i ($i \in V$), the beginning $\underline{u}_\ell^k$ and end $\bar{u}_\ell^k$ of every period of unavailability of every machine ($k = 1, \ldots, m$, $\ell = 1, \ldots, q_k$), the setup times γ_{jk}^F ($j \in V$, $k \in F(j)$) and γ_{ijk}^I ($i, j \in V$, $k \in F(i) \cap F(j)$), and the starting times s_i of every fixed operation $i \in T$ are integer values.

5.1.1 Representation of the Assignment of Operations to Machines

Let $\{i_1, i_2, \ldots, i_{\bar{o}}\} = V \setminus T$, with $i_1 \leq i_2 \leq \cdots \leq i_{\bar{o}}$, be the set of non-fixed operations. For each i_j, let $K_{i_j} = (k_{i_j,1}, k_{i_j,2}, \ldots, k_{i_j,|F(i_j)|})$ be a permutation of $F(i_j)$. Let $\tilde{\pi} = (\tilde{\pi}_j \in [0, 1) : j \in \{1, \ldots, \bar{o}\})$ be an array of real numbers that encodes the machine k_{i_j,π_j} to which each non-fixed operation i_j is assigned, where

$$\pi_j = \lfloor \tilde{\pi}_j |F(i_j)| + 1 \rfloor , \tag{5.1}$$

for $j = 1, \ldots, \bar{o}$. For example, given $F(i_j) = \{1, 4, 7\}$, the permutation $K_{i_j} = (1, 4, 7)$, and $\tilde{\pi}_j = 0.51$, then $\pi_j = \lfloor 0.51 \times 3 + 1 \rfloor = 2$, and, thus, $k_{i_j,\pi_j} = k_{i_j,2} = 4$; implying that operation i_j is assigned to machine 4. For simplicity, let $\kappa(i_j) = \kappa_{i_j,\pi_j}$. Then, if defined $\kappa(i)$ as the only element in the singleton $F(i)$ for the fixed operations $i \in T$, it becomes clear that the array of real numbers $\tilde{\pi} = (\tilde{\pi}_1, \ldots, \tilde{\pi}_{\bar{o}})$ defines a machine assignment $i \to \kappa(i)$ for $i = 1, \ldots, o$, see Figure 5.1.

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
i_j	2	3	4	5	6	7	8	9	10	12	13	14	15	16
K_{i_j}	$(1,2)$	$(3,4)$	$(2,4)$	$(2,4)$	$(1,2)$	$(1,3)$	$(3,4)$	$(1,2)$	$(3,4)$	$(1,3)$	$(1,3)$	$(1,2)$	$(2,4)$	$(1,3)$
$\tilde{\pi}_j$	0.05	0.79	0.48	0.26	0.17	0.53	0.99	0.09	0.95	0.63	0.52	0.02	0.31	0.62
π_j	1	2	1	1	1	2	2	1	2	2	2	1	1	2
$\kappa(i_j)$	1	4	2	2	1	3	4	1	4	3	3	1	2	3

Figure 5.1: An arbitrary machine assignment array assuming that operations 1 and 11 are fixed operations with $F(1) = \{3\}$ and $F(11) = \{2\}$, so $\kappa(1) = 3$ and $\kappa(11) = 2$.

j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
i_j	2	3	4	5	6	7	8	9	10	12	13	14	15	16
$\tilde{\sigma}_j$	0.05	0.55	0.95	0.51	0.75	0.54	0.00	0.99	0.15	0.15	0.16	0.11	0.79	0.55
σ_j	2	10	12	14	13	5	7	3	6	8	15	16	4	9

Figure 5.2: An operations execution order sequence σ produced by considering the values in $\tilde{\sigma}$ and the precedence relations given by the DAG represented in Figure 2.6. Note, once again, that fixed operations 1 and 11 are unsequenced at this point.

5.1.2 Representation of a Sequencing of Operations

Let $\tilde{\sigma} = (\tilde{\sigma}_j \in [0,1) : j \in \{1,\ldots,\bar{o}\})$ be an array of real numbers that encodes the order of execution of the non-fixed operations that are assigned to the same machine. Consider two non-fixed operations i_a and i_b such that $\kappa(i_a) = \kappa(i_b)$, i.e., that were assigned to the same machine. If $\tilde{\sigma}_a < \tilde{\sigma}_b$ (or $\tilde{\sigma}_a = \tilde{\sigma}_b$ and $i_a < i_b$) and if there is no path from i_b to i_a in the DAG $D(V,A)$, then operation i_a is executed before operation i_b; otherwise i_b is executed before i_a.

Let $\sigma = (\sigma_j : j \in \{1,\ldots,\bar{o}\})$ be a permutation of the set of non-fixed operations $\{i_1,\ldots,i_{\bar{o}}\}$ such that, for every pair of non-fixed operations σ_{j_1} and σ_{j_2} with $\kappa(\sigma_{j_1}) = \kappa(\sigma_{j_2})$, $j_1 < j_2$ if and only if σ_{j_1} is processed before σ_{j_2}. The permutation σ can be computed from $\tilde{\sigma}$ and the DAG $D(V,A)$ as follows: (i) start with $\ell \leftarrow 0$; (ii) let $R \subseteq \{i_1, i_2, \ldots, i_{\bar{o}}\}$ be the set of non-fixed operations i_j such that $i_j \neq \sigma_s$ for $s = 1,\ldots,\ell$ and, in addition, for every arc $(i,i_j) \in A$ it holds that $i \in V \setminus T$ and $i = \sigma_t$ for some $t = 1,\ldots,\ell$ or $i \in T$; (iii) take the operation $i_j \in R$ with smallest $\tilde{\sigma}_j$ (in case of a tie, select the operation with the smallest index i_j), set $\sigma_{\ell+1} = i_j$, and $\ell \leftarrow \ell + 1$; and (iv) if $\ell < \bar{o}$, return back to (ii), see Figure 5.2.

For further reference, for each machine k, let $\phi_k = (\phi_{k,1},\ldots,\phi_{k,|\phi_k|})$ be the subsequence of σ composed by the operations σ_ℓ such that $\kappa(\sigma_\ell) = k$. Given the machine assignment $\tilde{\pi}$ as illustrated in Figure 5.1 and the order of execution within each machine implied by $\tilde{\sigma}$ as illustrated in Figure 5.2, it holds that $\phi_1 = (2,14,6,9), \phi_2 = (5,15,4), \phi_3 = (12,13,7,16)$, and $\phi_4 = (10,3,8)$. Please note that fixed operations are not included. Moreover, let $\Phi = (\phi_1,\ldots,\phi_m)$.

5.1.3 Construction of a Feasible Solution and Makespan Calculation

Let the machine assignment $\tilde{\pi}$ and the execution order $\tilde{\sigma}$ be given; and let π, σ, κ, and ϕ_k $(k = 1, \ldots, m)$ be computed from $\tilde{\pi}$ and $\tilde{\sigma}$ as described in Sections 5.1.1 and 5.1.2. Recall that, for all fixed operations $i \in T$, it is assumed that it is known the starting time s_i, the processing time p_i, the completion time c_i, the value u_i such that $s_i + u_i + p_i = c_i$, and the "partial completion time" $\bar{c}_i$, that is the instant at which $\lceil \theta_i \times p_i \rceil$ units of time of operation i have already been processed. With all that said, an algorithm is described to compute s_i, $\bar{c}_i$, u_i, p_i, and c_i for all $i \in V \setminus T$ and to sequence the fixed operations $i \in T$ in order to construct a feasible scheduling. The algorithm also determines for all the operations (fixed and non-fixed) the corresponding sequence-dependent setup time ξ_i and some additional quantities (d_i, s_i^{lb}, and c_i^{lb}) whose meaning will be elucidated later. The algorithm processes a non-fixed operation $i \in V \setminus T$ at a time and schedules it as soon as possible (for the given $\tilde{\pi}$ and $\tilde{\sigma}$), constructing a semi-active scheduling. This computation includes sequencing the fixed operations $i \in T$.

Let $\text{pos}(i)$ be defined as the position of operation i in the sequence $\phi_{\kappa(i)}$; i.e., for any non-fixed operation i, it holds that $1 \leq \text{pos}(i) \leq |\phi_{\kappa(i)}|$. This means that, according to $\tilde{\pi}$ and $\tilde{\sigma}$ and ignoring the fixed operations, for a non-fixed operation i, $\text{ant}(i) = \phi_{\kappa(i),\text{pos}(i)-1}$ is the operation that is processed immediately before i on machine $\kappa(i)$; and $\text{ant}(i) = 0$ if i is the first operation to be processed on the machine. For further reference, $\text{suc}(i) = \phi_{\kappa(i),\text{pos}(i)+1}$ is defined as the immediate successor of operation i on machine $\kappa(i)$, if operation i is not the last operation to be processed on the machine; and $\text{suc}(i) = o + 1$, otherwise.

For $k = 1, \ldots, m$, define the $(o + 1) \times o$ matrices Γ^k of setup times, with row index starting at 0, given by $\Gamma^k_{0j} = \gamma^F_{jk}$ for $j = 1, \ldots, o$ and $\Gamma^k_{ij} = \gamma^I_{ijk}$ for $i, j = 1, \ldots, o$. Then it holds that, according to ϕ_k (that does not include the fixed operations yet), the setup time ξ_i of operation i is given by $\xi_i = \Gamma^{\kappa(i)}_{\text{ant}(i),i}$. Moreover, if $c_0 = 0$, it is obtained $c_{\text{ant}(i)} + \xi_i$ as an Lower Bound (LB) for the starting time s_i of operation i on machine $\kappa(i)$.

The algorithm follows below. In the algorithm, $\text{size}(\cdot)$ is a function that, if applied to an interval $[a, b]$ returns it size given by $b - a$ and, if applied to a set of non-overlapping intervals, returns the sum of the sizes of the intervals.

Algorithm 5.1.3.1.

Input: σ_i, κ_i $(i \in V)$, ϕ_k $(k = 1, \ldots, m)$, s_i, u_i, p_i, $\bar{c}$, c_i $(i \in T)$.

Output: ϕ_k $(k = 1, \ldots, m)$, s_i, u_i, p_i, $\bar{c}$, c_i $(i \in V \setminus T)$, ξ_i, d_i, s_i^{lb}, c_i^{lb} $(i \in V)$, $C_{\max}$.

For each $\ell = 1, \ldots, \bar{o}$, execute Steps 1 to 6. Then execute Step 7.

Step 1: Set $i \leftarrow \sigma_\ell$, $k \leftarrow \kappa(i)$, $p_i = p_{ik}$, $\bar{p}_i = \lceil \theta_i \times p_{ik} \rceil$, and $\text{delay}_i \leftarrow 0$ and compute

$$s_i^{\text{lb}} = \max \left\{ \max_{\{j \in V | (j,i) \in A\}} \{\bar{c}_j\}, \; r_i \right\} \quad \text{and} \quad c_i^{\text{lb}} = \max_{\{j \in V | (j,i) \in A\}} \{c_j\}. \tag{5.2}$$

Step 2: Set $\xi_i = \Gamma^k_{\text{ant}(i),i}$, define

$$d_i = \max \left\{ s_i^{\text{lb}}, \; c_{\text{ant}(i)} + \xi_i \right\}, \tag{5.3}$$

and compute $s_i \geq d_i + \text{delay}_i$ as the earliest starting time such that the interval $(s_i - \xi_i, s_i]$ does not intersect any period of unavailability of machine k, i.e.,

$$\left(\cup_{\ell=1}^{q_k} [\underline{u}_\ell^k, \bar{u}_\ell^k] \right) \cap (s_i - \xi_i, s_i] = \emptyset. \tag{5.4}$$

Step 3: Compute the completion time $c_i \notin (\underline{u}_\ell^k, \bar{u}_\ell^k]$, for $\ell = 1, \ldots, q_k$, such that

$$\text{size}([s_i, c_i]) - u_i = p_i, \tag{5.5}$$

where

$$u_i = \text{size}([s_i, c_i] \cap (\cup_{\ell=1}^{q_k} [\underline{u}_\ell^k, \bar{u}_\ell^k])) \tag{5.6}$$

is the time machine k is unavailable in between s_i and c_i.

Step 4: Let $f \in T$ be an operation fixed at machine k such that

$$c_{\text{ant}(i)} \leq s_f < c_i + \Gamma^k_{if}. \tag{5.7}$$

If there is none, go to Step 5. If there is more than one, consider the one with the ;earliest starting time s_f. Insert f in ϕ_k in between operations $\text{ant}(i)$ and i and go to Step 2. (Note that this action automatically redefines $\text{ant}(i)$ as f.)

Step 5: If $c_i \not\geq c_i^{\text{lb}}$ then set

$$\text{delay}_i = \text{size}([c_i, \hat{c}_i^{\text{lb}}]) - \text{size}([c_i, \hat{c}_i^{\text{lb}}] \cap \left(\cup_{\ell=1}^{q_k} [\underline{u}_\ell^k, \bar{u}_\ell^k] \right)),$$

where

$$\hat{c}_i^{\text{lb}} = \begin{cases} c_i^{\text{lb}}, & \text{if } c_i^{\text{lb}} \notin (\underline{u}_\ell^k, \bar{u}_\ell^k] \text{ for } \ell = 1, \ldots, q_k, \\ \bar{u}_\ell^k + 1, & \text{if } c_i^{\text{lb}} \in (\underline{u}_\ell^k, \bar{u}_\ell^k] \text{ for some } \ell \in \{1, \ldots, q_k\}, \end{cases}$$

and go to Step 2.

Step 6: Compute the "partial completion time" $\bar{c}_i \notin (\underline{u}_\ell^k, \bar{u}_\ell^k]$, for $\ell = 1, \ldots, q_k$, such that

$$\text{size}([s_i, \bar{c}_i]) - \bar{u}_i = \bar{p}_i,$$

where

$$\bar{u}_i = \text{size}([s_i, \bar{c}_i] \cap (\cup_{\ell=1}^{q_k} [\underline{u}_\ell^k, \bar{u}_\ell^k])).$$

Step 7: Compute $C_{\max} = \max_{i \in V}\{c_i\}$. For each unsequenced operation $f \in T$, sequence it according to its starting time s_f, update $\phi_{\kappa(f)}$, compute s_f^{lb} and c_f^{lb} according to (5.2), $\xi_f = \Gamma_{\text{ant}(f)}^{\kappa(f)}$, and d_f according to (5.3).

At Step 1, an LB s_i^{lb} to s_i is computed based on the release time r_i and the partial completion times $\bar{c}_j$ of the operations j such that $(j, i) \in A$ exists. In an analogous way, an LB c_i^{lb} to c_i is computed, based on the completion times c_j of the operations j such that $(j, i) \in A$ exists.

At Step 2, a tentative s_i is computed. At this point, it is assumed that the operation which is executed immediately before i on machine $\kappa(i)$ is the one that appears right before it in ϕ_k (namely ant(i)); and, for this reason, it is considered that the setup time of operation i is given by $\xi_i = \Gamma_{\text{ant}(i),i}^k$. (This may not be the case if it is decided that a still unsequenced fixed operation should be sequenced in between them.) The computed s_i is required by (5.3) to be not smaller than (a) its LB s_i^{lb} computed at Step 1 and (b) the completion time $c_{\text{ant}(i)}$ of operation ant(i) plus the setup time ξ_i. Note that if operation i is the first operation to be processed on machine $\kappa(i)$ then ant$(i) = 0$ and, by definition, $c_{\text{ant}(i)} = c_0 = 0$. At this point, it is assumed that $\text{delay}_i = 0$. Its role will be elucidated soon. In addition to satisfying the LBs (a) and (b), s_i is required in (5.4) to be such that (i) it does not coincide with the beginning of a period of unavailability, (ii) there is enough time right before s_i to execute the setup operation, and (iii) the setup operation is not interrupted by periods of unavailability of the machine. The required s_i is the smallest one that satisfies all the mentioned constraints. Therefore, it becomes clear that there is only a finite and small number of possibilities for s_i related to the imposed LBs and the periods of unavailability of the machine.

Once the tentative s_i has been computed at Step 2, Step 3 is devoted to the computation of its companion completion time c_i. Basically, ignoring the possible existence

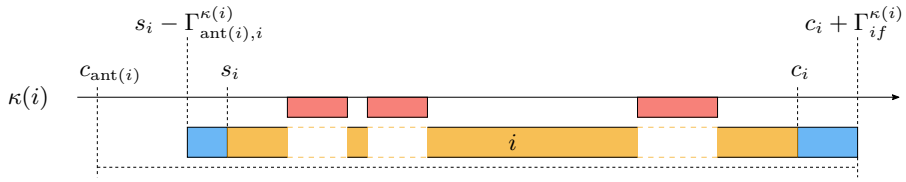


Figure 5.3: If a fixed operation f on machine $\kappa(i)$ exists such that $c_{\text{ant}(i)} \leq s_f < c_i + \Gamma^{\kappa(i)}_{if}$, it means that there is not enough space for operation i after ant(i) and before f. Thus, the unsequenced fixed operations f must be sequenced in between operations ant(i) and i.

of fixed operations on the machine, (5.5) and (5.6) indicate that c_i is such that between s_i and c_i the time during which machine $\kappa(i)$ is available is exactly the time required to process operation i. In addition, $c_i \notin (\underline{u}^k_\ell, \bar{u}^k_\ell]$, for $\ell = 1, \ldots, q_k$, says that, if the duration of the interval yields $c_i \in [\underline{u}^k_\ell, \bar{u}^k_\ell]$ for some $\ell \in \{1, \ldots, q_k\}$, it must be taken $c_i = \underline{u}^k_\ell$, since any other choice would artificially increase the completion time of the operation.

In Step 4 it is checked whether the selected interval $[s_i, c_i]$ is infeasible due to the existence of a fixed operation on the machine. If there is not a fixed operation f satisfying (5.7) then Step 4 is skipped. Note that $c_{\text{ant}(i)}$ is the completion time of the last operation scheduled on machine $\kappa(i)$. This means that if a fixed operation f exists such that $s_f \geq c_{\text{ant}(i)}$, the fixed operation f is still unsequenced. The non-existence of a fixed operation f satisfying (5.7) is related to exactly one of the following two cases:

- There are no fixed operations on machine $\kappa(i)$ or all fixed operations on machine $\kappa(i)$ have already been sequenced;
- The starting time s_f of the closest unsequenced fixed operation f on machine $\kappa(i)$ is such that operation i can be scheduled right after operation ant(i), starting at s_i, being completed at c_i and, after c_i and before s_f there is enough time to process the setup operation with duration $\Gamma^{\kappa(i)}_{if}$.

Assume now that at least one fixed operation satisfying (5.7) exists and let f be the one with smallest s_f. This means that scheduling operation i in the interval $[s_i, c_i]$ is infeasible, see Figure 5.3. Therefore, operation f must be sequenced right after ant(i), by including it in $\phi_{\kappa(i)}$ in between ant(i) and i. This operation transforms f in a sequenced fixed operation that automatically becomes ant(i), i.e., the operation sequenced on machine $\kappa(i)$ right before operation i. With the redefinition of ant(i), the task of determining the starting and the completion times of operation i must be restarted. This task restarts returning to Step 2, where a new setup time for operation i is computed and a new $c_{\text{ant}(i)}$ is considered in (5.3). Since the number of fixed operations is finite and the number of unsequenced fixed operations is reduced by one, this iterative process ends in finite time.

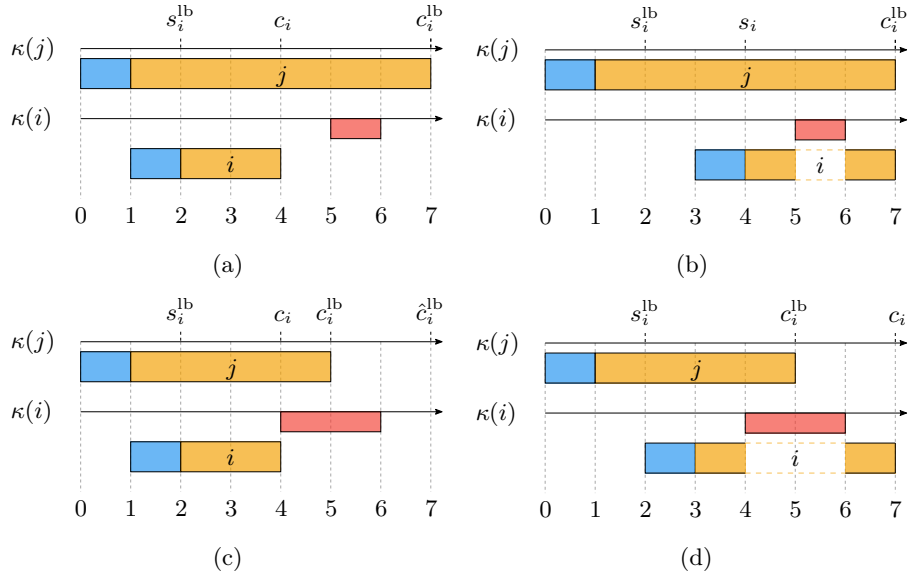


Figure 5.4: Delay computation for the case in which $c_i \not\geq c_i^{\text{lb}}$.

Step 5 is devoted to checking whether the computed completion time c_i is not smaller than its LB c_i^{lb}, computed at Step 1. If this is not the case, the algorithm proceeds to Step 6. In case $c_i < c_i^{\text{lb}}$, the starting time of operation i must be delayed. This is the role of the variable delay_i that was initialized with zero. If the extent of the delay is too short, the situation may repeat. If the extent is too long, the starting of the operation may be unnecessarily delayed. Figure 5.4 helps to realize that the time machine $\kappa(i)$ is available in between c_i and c_i^{lb} is the minimum delay that is necessary to avoid the same situation when a new tentative s_i and its associated c_i are computed. In case (a), $\hat{c}_i^{\text{lb}} = c_i^{\text{lb}}$ and machine $\kappa(i)$ has two units of available time in between c_i and c_i^{lb}. Adding this delay to the LB of s_i results in the feasible scheduling (of operation i) depicted in (b). In case (c), $c_i^{\text{lb}} \in (\underline{u}_\ell^{\kappa(i)}, \bar{u}_\ell^{\kappa(i)}]$ for some $\ell \in \{1, \ldots, q_{\kappa(i)}\}$. Thus, $\hat{c}_i^{\text{lb}} = \bar{u}_\ell^{\kappa(i)} + 1$. Machine $\kappa(i)$ has one unit of available time in between c_i and $\hat{c}_i^{\text{lb}}$. Adding this delay to the LB of s_i results in the feasible scheduling (of operation i) depicted in (d). Once the delay is computed, a new attempt is done returning to Step 2; this time with a non-null delay_i.

When the algorithm arrives at Step 6, feasible values for s_i and c_i have been computed, then it is computed the partial completion time $\bar{c}_i$ that will be used for computing the starting and completion times of the forthcoming operations.

While executing Steps 1–6 for $\ell = 1, \ldots, \bar{o}$, i.e., while scheduling the unfixed operations, some fixed operations have to be sequenced as well. However, when the last unfixed operation is scheduled, it may be the case that some fixed operations, that were scheduled "far after" the largest completion time of the unfixed operations, played no role in the scheduling process and thus remain unsequenced, i.e., not in ϕ_k for any k. These unsequenced fixed operations are sequenced at Step 7.

5.2 Local Search

Given an initial solution, a LS procedure is an iterative process that constructs a sequence of solutions in such a way that each solution in the sequence is in the *neighborhood* of its predecessor in the sequence. The neighborhood of a solution is given by all solutions obtained by applying a *movement* to the solution. A movement is a simple modification of a solution. In addition, the LS described in the current section is such that each solution in the sequence improves the objective function value of its predecessor. In the remaining of the current section, the neighbourhood and the movement concepts introduced in (Mastrolilli and Gambardella, 2000) for the FJS are extended for the OPS scheduling problem.

The definition of the proposed movement is based on the representation of a solution by a digraph. Let $\tilde{\pi}$, encoding the machine assignment of the non-fixed operations, and $\tilde{\sigma}$, encoding the order of execution of the non-fixed operations within each machine, be given. Moreover, assume that, using Algorithm 5.1.3.1, ξ_i, d_i, s_i, u_i, p_i, $\bar{c}_i$, c_i, s_i^{lb}, c_i^{lb}, and d_i have been computed for all $i = 1, \ldots, o$. From now on, $\varsigma(\tilde{\pi}, \tilde{\sigma}) = (\tilde{\pi}, \tilde{\sigma}, \pi, \sigma, \kappa, \Phi, \xi, d, s, u, p, \bar{c}, c, s^{\text{lb}}, c^{\text{lb}})$ represents a feasible solution. (Recall that π is computed from $\tilde{\pi}$ as defined in (5.1); σ and Φ are computed from $\tilde{\sigma}$ as described in Section 5.1.2; and $\kappa(i) = \kappa_{i,\pi_i}$.) Let $\text{suc}(i) = \phi_{\kappa(i),\text{pos}(i)+1}$ be the successor of operation i on machine $\kappa(i)$, if operation i is not the last operation to be processed on the machine; and $\text{suc}(i) = o + 1$, otherwise. Recall that it is already defined $\text{ant}(i) = \phi_{\kappa(i),\text{pos}(i)-1}$, if i is *not* the first operation to be processed on machine $\kappa(i)$, while $\text{ant}(i) = 0$, otherwise. This means that, for any $i \in V$, i.e., including non-fixed and fixed operations, $\text{ant}(i)$ and $\text{suc}(i)$ represent, respectively, the operations that are processed right before i (antecedent) and right after i (sucessor) on machine $\kappa(i)$.

The weighted augmented digraph that represents the feasible solution ς is given by

$$D(\varsigma) = (V \cup \{0, o+1\}, A \cup W \cup U),$$

where

$$W = \{(\phi_{k,\ell-1}, \phi_{k,\ell}) \mid k \in \{1, \ldots, m\} \text{ and } \ell \in \{2, ..., |\phi_k|\}\}$$

and U is the set of arcs of the form $(0, i)$ for every $i \in V$ such that $\text{ant}(i) = 0$ plus arcs of the form $(i, o+1)$ for every $i \in V$ such that $\text{suc}(i) = o+1$, see Figure 5.5. The weights of the nodes and arcs of $D(\varsigma)$ are defined as follows:

- arcs $(j, i) \in A$ have weight $\bar{c}_j - c_j$;
- arcs $(\text{ant}(i), i) \in W$ have weight ξ_i;
- arcs $(0, i) \in U$ have weight $\max\{r_i, \xi_i\}$;

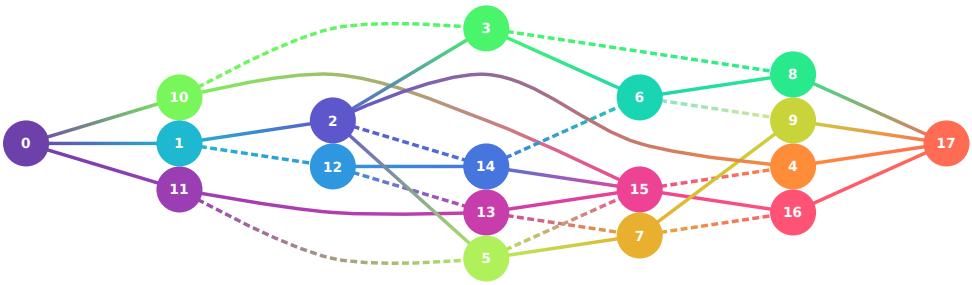


Figure 5.5: Directed acyclic graph $D(\varsigma) = (V \cup \{0, o+1\}, A \cup W \cup U)$ associated with the original precedence relations (in solid lines) illustrated in Figure 2.6 plus the precedence relations implied by the machine assignment $\tilde{\pi}$ in Figure 5.1 and the order of execution within each machine implied by $\tilde{\sigma}$ in Figure 5.2 (dashed lines). Arcs are directed from left to right.

- arcs $(i, o+1) \in U$ have null weight;
- each node $i \in V$ has weight $s_i - d_i + u_i + p_i$;
- nodes 0 and $o+1$ have null weight.

Weights of nodes and arcs are defined in such a way that, if the weight of a path $i_1, i_2, \ldots, i_q$ is defined as the sum of the weights of nodes $i_2, i_3, \ldots, i_q$ plus the sum of the weights of arcs $(i_1, i_2), \ldots, (i_{q-1}, i_q)$, then the value of the completion time c_i of operation i is given by some longest path from node 0 to node i. (If in between two nodes a and b there is more than one arc then the arc with the largest weight must be considered. This avoids naming the arcs explicitly when mentioning a path.) It follows that the weight of some longest path from 0 to $o+1$ equals $C_{\max}$ and the nodes on this path are called critical nodes or *critical operations*. Let t_i be denoted as the weight of a longest path from node i to node $o+1$. The value t_i (so-called tail time) gives an LB on the time elapsed between c_i and $C_{\max}$. It is worth noticing that (a) if an operation i is critical then $c_i + t_i = C_{\max}$ and that (b) if there is a path from i to j then $t_i \geq t_j$.

Assume that σ^{ifo} ("ifo" stands for "including fixed operations") is a permutation of $\{1, 2, \ldots, o\}$ that represents the order in which operations (non-fixed and fixed) where scheduled by Algorithm 5.1.3.1, i.e., non-fixed operations preserve in σ^{ifo} the same relative order they have in σ and σ^{ifo} corresponds to σ with the fixed operations inserted in the appropriate places. Note that σ^{ifo} can be easily obtained with a simple modification of Algorithm 5.1.3.1: start with σ^{ifo} as an empty list and every time an operation (non-fixed or fixed) is scheduled, add i to the end of the list. In the following, a description of a simple way to compute t_i for all $i \in V \cup \{0, o+1\}$ is given. Define $c_{o+1} = C_{\max}$ and $t_{o+1} = 0$ and for $\ell = o, \ldots, 1$, i.e., in decreasing order,

define $i = \sigma_\ell^{\text{ifo}}$ and

$$t_i = \max\left\{t_{\text{suc}(i)} + \omega(\text{suc}(i)) + \omega(i, \text{suc}(i)), \max_{\{j \in V \mid (i,j) \in A\}} \{t_j + \omega(j) + \omega(i,j)\}\right\}, \tag{5.8}$$

where $\omega(\cdot)$ and $\omega(\cdot,\cdot)$ represent the weight of a node or an arc, respectively. Finish defining

$$t_0 = \max_{\{j \in V \mid (0,j) \in U\}} \{t_j + \omega(j) + \omega(0,j)\}. \tag{5.9}$$

In addition to the tail times, the local search strategy also requires identifying a longest (critical) path from node 0 to node $o+1$, since operations on that path are the critical operations whose reallocation will be attempted. A critical path can be obtained as follows. Together with the computation of (5.8), define $\text{next}(i)$ as the index in $\{\text{suc}(i)\} \cup \{j \mid (i,j) \in A\}$ such that $t_i = t_{\text{next}(i)} + \omega(\text{next}(i)) + \omega(i, \text{next}(i))$, i.e., the one that realises the maximum. Analogously, together with (5.9) define $\text{next}(0) = \text{argmax}_{\{j \in V \mid (0,j) \in A\}} \{t_j + \omega(0,j)\}$. A longest path is then given by

$$0,\ \text{next}(0),\ \text{next}(\text{next}(0)),\ \text{next}(\text{next}(\text{next}(0))),\ \ldots,\ o+1.$$

5.2.1 Movement: Reallocating Operations

Let i be a (non-fixed) operation to be removed and reallocated. It can be reallocated in the same machine $\kappa(i)$, but in a different position in the sequence, or in a different machine $k \in F(i)$, $k \neq \kappa(i)$. Removing i from $\kappa(i)$ implies removing arcs $(\phi_{\kappa(i),\text{pos}(i)-1}, i)$ and $(i, \phi_{\kappa(i),\text{pos}(i)+1})$ from $W \cup U$ and including the arc $(\phi_{\kappa(i),\text{pos}(i)-1}, \phi_{\kappa(i),\text{pos}(i)+1})$ in W or U. (Whether the arcs to be removed and/or inserted belong to W or U depends on whether $\text{pos}(i) - 1 = 0$, $\text{pos}(i) + 1 = o + 1$, or none of the cases occur.) In the same sense, reallocating i implies creating two new arcs and deleting an arc. Let $D(\varsigma)^{-i}$ be the digraph after the removal of the critical operation i; and let $D(\varsigma)^{+i}$ be the digraph after its reallocation.

The relevant fact in the reallocation of operation i is avoiding the creation of a cycle in $D(\varsigma)^{+i}$, i.e., the construction of a feasible solution. Thus, for each $k \in F(i)$, the following sets of operations are defined

$$R_k = \{j \in \phi_k \mid \bar{c}_j > s_i^{\text{lb}}\}$$

and

$$L_k = \{j \in \phi_k \mid t_j + u_j + p_j > C_{\max} - \bar{c}_i^{\text{ub}}\}.$$

where

$$\bar{c}_i^{\text{ub}} = \min_{(i,j) \in A} \{s_j\}$$

is an Upper Bound (UB) for $\bar{c}_i$ and, thus, $C_{\max} - \bar{c}_i^{\text{ub}}$ is an LB for the time between $\bar{c}_i$ and $C_{\max}$. Properties of R_k and L_k follow:

R1 If $j \in R_k$ then $\bar{c}_j > s_i^{\text{lb}}$. Assume that there is a path from j to i in $D(\varsigma)^{-i}$. By the definition of s_i^{lb}, $\bar{c}_j > s_i^{\text{lb}}$ implies that $(j,i) \notin A$. Then, in the path from j to i, the immediate predecessor of i must be an operation $j' \notin R_k$ and such that $(j',i) \in A$, i.e., such that $\bar{c}_{j'} \leq s_i^{\text{lb}}$. Therefore, it must hold $\bar{c}_j \leq s_{j'} < \bar{c}_{j'} \leq s_i^{\text{lb}}$. Thus, if $j \in R_k$ then there is no path from j to i in $D(\varsigma)^{-i}$.

R2 If $j \in \phi_k \setminus R_k$ then $s_j < \bar{c}_j \leq s_i^{\text{lb}} \leq s_i < \bar{c}_i$. Therefore, there is no path from i to j in $D(\varsigma)^{-i}$.

L1 If $j \in L_k$ then $t_j + u_j + p_j > C_{\max} - \bar{c}_i^{\text{ub}}$. If there were a path from i to j in $D(\varsigma)^{-i}$ then $\bar{c}_i \leq s_j$ and, therefore, the LB on the distance between $\bar{c}_i$ and $C_{\max}$, given by $C_{\max} - \bar{c}_i^{\text{ub}}$, should be greater than or equal to the LB of the distance between s_j and $C_{\max}$, given by $t_j + u_j + p_j$. Therefore, if $j \in L_k$ then there is no path from i to j in $D(\varsigma)^{-i}$.

L2 If $j \in \phi_k \setminus L_k$ then $C_{\max} - \bar{c}_i^{\text{ub}} \geq t_j + u_j + p_j$. Assume that there is a path from j to i in $D(\varsigma)^{-i}$. Then, it must hold $s_j < s_i$ and, since $\theta_i > 0$ and, thus, $s_i < \bar{c}_i$, it follows that $s_j < \bar{c}_i$. This means that the distance between s_j and $C_{\max}$ is greater than the distance between $\bar{c}_i$ and $C_{\max}$ that, by definition, is bounded from below by $C_{\max} - \bar{c}_i^{\text{ub}}$, i.e., $t_j + u_j + p_j > C_{\max} - \bar{c}_i^{\text{ub}}$. Thus, if $j \in \phi_k \setminus L_k$ then there is no path from j to i in $D(\varsigma)^{-i}$.

Properties R1, R2, L1, and L2 imply that if operation i is reallocated in the sequence of a machine $k \in F(i)$ in a position such that all operations in $L_k \setminus R_k$ are to the left of i and all operations in $R_k \setminus L_k$ are to the right of i, then this insertion defines a feasible solution, i.e., $D(\varsigma)^{+i}$ has no cycles.

5.2.2 Neighborhood

It is well known in the scheduling literature that removing and reallocating a non-critical operation does not reduce the makespan of the current solution. Therefore, in the present work, it is defined as neighborhood of a solution ς the set of (feasible) solutions that are obtained when each critical operation i is removed and reallocated in all possible position of the sequence of every machine $k \in F(i)$, as described in the previous section. This means that, for each critical operation i, it is proceeded as follows: (i) operation i is removed from machine $\kappa(i)$; (ii) for each $k \in F(i)$, (iia) the sets R_k and L_k are determined and (iib) operation i is reallocated in the sequence of machine k in every possible position such that all operations in $L_k \setminus R_k$ are to the left

of i and all operations in $R_k \setminus L_k$ are to the right of i. For further reference, the set of neighbours of ς is named $\mathcal{N}(\varsigma)$.

5.2.3 Estimation of the Makespan of Neighbor Solutions

Given the sequences $\tilde{\pi}$ and $\tilde{\sigma}$ of the current solution ς, computing the sequences $\tilde{\pi}'$ and $\tilde{\sigma}'$ (as well as π', σ', and κ') associated with a neighbour solution $\varsigma' \in \mathcal{N}(\varsigma)$ is a trivial task. Computing the makespan (together with the quantities ξ', s', u', p', $\bar{c}'$, c', $s^{\mathrm{lb}'}$, $c^{\mathrm{lb}'}$) associated with ς' is also simple, but it requires executing Algorithm 5.1.3.1, which might be considered an expensive task in this context. Therefore, the selection of a neighbor is based on the computation of an *estimation* of its associated makespan. In fact, following (Mastrolilli and Gambardella, 2000), what is used as an estimation of the makespan is an estimation of the length of a longest path from node 0 to node $o+1$ in $D(\varsigma')$ containing the operation that was reallocated to construct ς' from ς. The exact length of this path is an LB on the makespan associated with ς'.

The estimation of the makespan of a neighbour solution $\varsigma' \in \mathcal{N}(\varsigma)$ obtained by removing and reallocating operation i somewhere in the sequence of machine k is determined as follows. If $L_k \cap R_k = \emptyset$ then the estimation of the makespan is given by $s_i^{\mathrm{lb}} + p_{ik} + C_{\max} - \bar{c}_i^{\mathrm{ub}}$. If $L_k \cap R_k \neq \emptyset$, consider the elements (operations) in $L_k \cap R_k$ sorted in increasing order of their starting times; and let $\tau : \{1, \ldots, |L_k \cap R_k|\} \to L_k \cap R_k$ be such that $s_{\tau(1)} < s_{\tau(2)} < \cdots < s_{\tau(|L_k \cap R_k|)}$ and, in consequence, $t_{\tau(1)} > t_{\tau(2)} > \cdots > t_{\tau(|L_k \cap R_k|)}$. Let j be such that $j = 0$ if operation i is being inserted before operation $\tau(1)$ and $1 \leq j \leq |L_k \cap R_k|$ if operation i is being inserted right after operation $\tau(j)$. In this case, the estimation of the makespan is given by

$$p_{ik} + \begin{cases} s_i^{\mathrm{lb}} + p_{\tau(1)} + u_{\tau(1)} + t_{\tau(1)}, & \text{if } j = 0, \\ s_{\tau(j)} + p_{\tau(j)} + u_{\tau(j)} + p_{\tau(j+1)} + u_{\tau(j+1)} + t_{\tau(j+1)}, & \text{if } 1 \leq j < |L_k \cap R_k|, \\ s_{\tau(j)} + p_{\tau(j)} + u_{\tau(j)} + C_{\max} - \bar{c}_i^{\mathrm{ub}}, & \text{if } j = |L_k \cap R_k|. \end{cases} \tag{5.10}$$

These estimations follow very closely those introduced by Mastrolilli and Gambardella (2000) for the FJS, see (Mastrolilli and Gambardella, 2000, §5) for details.

5.2.4 Local Search Procedure

The LS procedure starts at a given solution. It identifies all critical operations (operations in the longest path from node 0 to node $o+1$) and for each critical operation i and each $k \in F(i)$ it computes the estimation of the makespan associated with removing and reallocation operation i in every possible position of the sequence of machine k

(as described in the previous sections). The neighbor with the smallest estimation of the makespan is selected and its actual makespan is computed by applying Algorithm 5.1.3.1. In case this neighbor solution improves the makespan of the current solution, the neighbor solution is accepted as the new current solution and the iterative process continues. Otherwise, the LS stops.

5.3 Metaheuristics

In this section, the four proposed metaheuristics are described. Two of the metaheuristics, namely GA and DE are population-based methods, while the other two, ILS and TS, are trajectory methods. GA and TS were chosen because they are the two most popular metaheuristics applied to the FJS scheduling problem (Chaudhry and Khan, 2016, Table 4). On the other hand, in the last decade DE has been successfully applied to a wide range of complex real-world problems (Damak et al., 2009; Wang et al., 2010; Ali, Siarry, and Pant, 2012; Tsai, Fang, and Chou, 2013; Yuan and Xu, 2013), but its performance while dealing with the FJS scheduling problem with sequencing flexibility was not tested yet. Another reason that reinforces the choice of DE is that preliminary experiments involving other well-known metaheuristics such as Artificial Bee Colony (ABC) (Karaboga and Akay, 2009), Firefly Algorithm (FA) (Yang, 2010), Grey Wolf Optimizer (GWO) (Mirjalili, Mirjalili, and Lewis, 2014), Imperialist Competitive Algorithm (ICA) (Atashpaz-Gargari and Lucas, 2007), Particle Swarm Optimization (PSO) (Kennedy and Eberhart, 1995), and Whale Optimization Algorithm (WOA) (Mirjalili and Lewis, 2016) showed that DE achieved outstanding results among the considered methods. Finally, ILS is being considered due to its simplicity of implementation and usage. All metaheuristics are based on the same representation scheme (described in Section 5.1) and use the same definition of the neighborhood (described in Section 5.2).

In the current section, $\vec{x} \in \mathbb{R}^{2\bar{o}}$ is denoted as the concatenation of a machine assignment $\tilde{\pi}$ and an execution order $\tilde{\sigma}$. This means that $\vec{x}_1, \ldots, \vec{x}_{\bar{o}}$ correspond to $\tilde{\pi}_1, \ldots, \tilde{\pi}_{\bar{o}}$; while $\vec{x}_{\bar{o}+1}, \ldots, \vec{x}_{2\bar{o}}$ correspond to $\tilde{\sigma}_1, \ldots, \tilde{\sigma}_{\bar{o}}$. Given $\vec{x}$ (and the instance constants s_i, u_i, p_i, $\bar{c}_i$, and c_i for $i \in T$), it is easy to compute π_i, σ_i $(i \in V \setminus T)$, κ_i $(i \in V)$, and ϕ_k $(k = 1, \ldots, m)$ as described in Sections 5.1.1 and 5.1.2; and then the associated makespan $C_{\max}$ using Algorithm 5.1.3.1. In this section, $f(\vec{x}) = C_{\max}$ is denoted as the makespan of the associated solution given by $\vec{x}$. Additionally, in the algorithms, the short terms "chosen", "random" or "randomly chosen" should be interpreted as abbreviations of "randomly chosen with uniform distribution".

Initial solutions of all methods are constructed in the same way. For each operation $i \in V \setminus T$, the machine $k \in F(i)$ with the lowest processing time is chosen. (For operations

$i \in T$, the machine that processes operation i is fixed by definition.) Then, a Cost-based Breadth-First Search (CBFS) algorithm is used to sequence the operations. The *costs* of each operation are given by a random number in $[0, 1]$. At each iteration of the CBFS algorithm, a set of eligible operations $\mathcal{E}$ is defined. Operations in $\mathcal{E}$ are those for which their immediate predecessors have already been sequenced. If $|\mathcal{E}| > 1$, operations in $\mathcal{E}$ are sequenced in increasing order of their costs; if $|\mathcal{E}| = 1$ then the single operation in $\mathcal{E}$ is sequenced. The procedure ends when $\mathcal{E} = \emptyset$ which implies that all operations have been sequenced. In the following, the main principles of each adapted metaheuristic framework for the OPS scheduling problem are schematically describe, see Section 3.8 for a more in-depth description of each metaheuristic framework.

5.3.1 Differential Evolution

The proposed DE follows very closely the Algorithm 3.8.5. The major difference is that the proposed DE contains an additional parameter named "variant", which denotes the mutation scheme that will be used to generate $\vec{v}_i$. If parameter variant = DE/rand/1 then the indexes $r_1 \neq r_2 \neq r_3$ are randomly chosen in $\{1, 2, \ldots, n_{\text{size}}\} \backslash \{i\}$. If variant = DE/best/1 then the index r_1 is the index of the individual with best fitness value in the population, while $r_2 \neq r_3$ remain random numbers in $\{1, 2, \ldots, n_{\text{size}}\} \setminus \{i, r_1\}$. Algorithm 5.3.1 shows the essential steps of the proposed DE algorithm.

5.3.2 Genetic Algorithm

In the proposed GA, tournament selection is used to select the individuals (solutions) that are recombined (crossover) to generate the offspring. During tournament selection, two pairs of individuals are randomly chosen from the population and the fittest individual of each pair takes part of the recombination using uniform crossover. Preliminary experiments with uniform crossover, two-point crossover and simulated binary crossover (Deb and Agrawal, 1995), showed that uniform crossover achieves the best results. Therefore, during uniform crossover of two solutions $\vec{x}^{\,i_1}$ and $\vec{x}^{\,i_2}$, two new solutions $\vec{x}^{\,j_1}$ and $\vec{x}^{\,j_2}$ are generated as follows. For each $k \in \{1, 2, \ldots, 2\bar{o}\}$, with probability $\frac{1}{2}$, $\vec{x}_k^{\,j_1} \leftarrow \vec{x}_k^{\,i_1}$ and $\vec{x}_k^{\,j_2} \leftarrow \vec{x}_k^{\,i_2}$; otherwise, $\vec{x}_k^{\,j_1} \leftarrow \vec{x}_k^{\,i_2}$ and $\vec{x}_k^{\,j_2} \leftarrow \vec{x}_k^{\,i_1}$. Preliminary experiments with uniform mutation, Gaussian mutation and polynomial mutation (Deb and Agrawal, 1999; Deb and Deb, 2014), showed that uniform mutation achieves the best results. Therefore, following uniform crossover, each offspring solution $\vec{x}$ is mutated with probability $p_{\text{mut}} \in [0, 1]$. During mutation, a random integer value $j \in \{1, 2, \ldots, 2\bar{o}\}$ is chosen; and the value $\vec{x}_j$ is set to a random number in $[0, 1)$. Once the new population is finally built, an elitist strategy is used. If the

Algorithm 5.3.1 Differential Evolution algorithm

Input parameters: n_{size}, ζ, p_{cro}, variant, and t
$\mathcal{P} \leftarrow \emptyset$
for $i \leftarrow 1$ to n_{size} **do**
    Compute a random array of costs $c \in [0,1]^o$
    Using CBFS, construct an initial solution $\vec{x}^{\,i}$
    Let $\mathcal{P} \leftarrow \mathcal{P} \cup \{\vec{x}^{\,i}\}$
while time limit t not reached **do**
    for $i \leftarrow 1$ to n_{size} **do**
        if variant = DE/rand/1 **then**
            Compute random numbers $r_1 \neq r_2 \neq r_3 \in \{1, 2, \ldots, n_{\text{size}}\} \setminus \{i\}$
        else if variant = DE/best/1 **then**
            Let $r_1 \leftarrow \text{argmin}_{\ell=1,\ldots,n_{\text{size}}} \{f(\vec{x}^{\,\ell})\}$
            Compute random numbers $r_2 \neq r_3 \in \{1, 2, \ldots, n_{\text{size}}\} \setminus \{i, r_1\}$
        Compute $\vec{v} \leftarrow \max\{0, \min\{\vec{x}^{\,r_1} + \zeta(\vec{x}^{\,r_2} - \vec{x}^{\,r_3}), 1 - 10^{-16}\}\}$
        Compute a random number $R(i) \in \{1, \ldots, 2\bar{o}\}$
        for $j \leftarrow 1$ to $2\bar{o}$ **do**
            Compute a random number $\gamma \in [0, 1]$
            if $\gamma \leq p_{\text{cro}}$ or $j = R(i)$ **then**
                $\vec{u}^{\,i}_j \leftarrow \vec{v}^{\,i}_j$
            else
                $\vec{u}^{\,i}_j \leftarrow \vec{x}^{\,i}_j$
        Perform a local search starting from $\vec{u}^{\,i}$ to obtain $\vec{w}^{\,i}$ and compute $f(\vec{w}^{\,i})$
        if $f(\vec{w}^{\,i}) < f(\vec{x}^{\,i})$ **then** $\mathcal{P} \leftarrow \mathcal{P} \setminus \{\vec{x}^{\,i}\} \cup \{\vec{w}^{\,i}\}$
$\vec{x}^{\,\text{best}} \leftarrow \text{argmin}_{\vec{x} \in \mathcal{P}} \{f(\vec{x})\}$
Return $\vec{x}^{\,\text{best}}$

best individual $\vec{x}^{\,\text{best}}_{\text{new}}$ of the new population is less fit than the best individual $\vec{x}^{\,\text{best}}$ of the current population, i.e., if $f(\vec{x}^{\,\text{best}}_{\text{new}}) > f(\vec{x}^{\,\text{best}})$, then the worst individual of the new population is replaced with $\vec{x}^{\,\text{best}}$. Algorithm 5.3.2 shows the essential steps of the proposed GA.

5.3.3 Iterated Local Search

The proposed ILS also follows very closely the Algorithm 3.8.2. In the proposed ILS algorithm, the perturbation of the current solution $\vec{x}$ is governed by a perturbation strength $\hat{p} \in \{1, 2, \ldots, 2\bar{o}\}$ that determines how many randomly chosen positions of a local solution must be perturbed. The perturbation of a position simply consists in attributing to it a random value in $[0, 1)$. Algorithm 5.3.3 shows the essential steps of the considered ILS algorithm.

Algorithm 5.3.2 Genetic Algorithm

Input parameters: n_{size}, p_{mut}, and t.
$\mathcal{P} \leftarrow \emptyset$.
for $i \leftarrow 1$ to n_{size} **do**
    Compute a random array of costs $c \in [0,1]^o$ and, using CBFS, construct an initial solution $\vec{x}^{\,i}$.
    Let $\mathcal{P} \leftarrow \mathcal{P} \cup \{\vec{x}^{\,i}\}$.
while time limit t not reached **do**
    Let $\mathcal{Q} \leftarrow \emptyset$.
    for 1 to $n_{\text{size}}/2$ **do**
        Compute random numbers $r_1 \neq r_2 \neq r_3 \neq r_4 \in \{1, 2, \ldots, n_{\text{size}}\}$ and
        let $\vec{x}^{\,i_1} \leftarrow \operatorname{argmin}\{f(\vec{x}^{\,r_1}), f(\vec{x}^{\,r_2})\}$ and $\vec{x}^{\,i_2} \leftarrow \operatorname{argmin}\{f(\vec{x}^{\,r_3}), f(\vec{x}^{\,r_4})\}$.
        for $\ell \leftarrow 1$ to $2\bar{o}$ **do**
            Compute a random number $\gamma \in [0,1]$.
            if $\gamma \leq \frac{1}{2}$ **then** $\vec{x}_\ell^{\,j_1} \leftarrow \vec{x}_\ell^{\,i_1}$ and $\vec{x}_\ell^{\,j_2} \leftarrow \vec{x}_\ell^{\,i_2}$
            else $\vec{x}_\ell^{\,j_1} \leftarrow \vec{x}_\ell^{\,i_2}$ and $\vec{x}_\ell^{\,j_2} \leftarrow \vec{x}_\ell^{\,i_1}$
        for $j \in \{j_1, j_2\}$ **do**
            Compute a random numbers $\gamma \in [0,1]$.
            if $\gamma \leq p_{\text{mut}}$ **then**
                Compute random numbers $r \in \{1, 2, \ldots, 2\bar{o}\}$ and $\xi \in [0,1)$ and let $\vec{x}_r^{\,j} \leftarrow \xi$.
            Perform a local search starting from $\vec{x}^{\,j}$ to generate $\vec{x}^{\,k}$.
            Let $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{\vec{x}^{\,k}\}$.
    Let $\vec{x}^{\,\text{best}} \leftarrow \operatorname{argmin}_{\vec{x} \in \mathcal{P}} \{f(\vec{x})\}$ and $\vec{x}_{\text{new}}^{\,\text{best}} \leftarrow \operatorname{argmin}_{\vec{x} \in \mathcal{Q}} \{f(\vec{x})\}$.
    if $f(\vec{x}_{\text{new}}^{\,\text{best}}) > f(\vec{x}^{\,\text{best}})$ **then**
        $\vec{x}_{\text{new}}^{\,\text{worst}} \leftarrow \operatorname{argmax}_{\vec{x} \in \mathcal{Q}} \{f(\vec{x})\}$ and $\mathcal{Q} \leftarrow \mathcal{Q} \setminus \{\vec{x}_{\text{new}}^{\,\text{worst}}\} \cup \{\vec{x}^{\,\text{best}}\}$.
    Let $\mathcal{P} \leftarrow \mathcal{Q}$.
$\vec{x}^{\,\text{best}} \leftarrow \operatorname{argmin}_{\vec{x} \in \mathcal{P}} \{f(\vec{x})\}$.
Return $\vec{x}^{\,\text{best}}$.

5.3.4 Tabu Search

In the proposed TS, an action is composed of a couple (i,k), where i is an operation being moved and k is the machine to which i was assigned before the move. Actions are tracked by means of a matrix $\tau = (\tau_{ik})$ with $i = 1, \ldots, \bar{o}$ and $k = 1, \ldots, m$. In this way, τ_{ik} is set to $iter + T$ whenever action (i,k) is performed at iteration $iter$, i.e. $\tau_{ik} = iter + T$ whenever a move is performed from the current solution $\vec{x}$ to another solution $\vec{x}\,' \in N(\vec{x})$ by assigning to machine k' an operation i currently assigned to machine k. An action (i,k) is tabu if $\tau_{ik} > iter$. The tabu tenure T is crucial to the success of the tabu search procedure and it is defined as $T = T(\lambda) = \lceil \lambda \log_e(\bar{o})^2 \rceil$, where λ is a parameter in $[0,2]$. During the search, the next solution is randomly chosen among the two neighbors with the smallest estimated makespan (see Section 5.2.3) that are non-tabu. Note that the neighborhood is defined as in the LS described in Section 5.2.2. If all neighbors are tabu, a neighbor whose associated action (i,k) has the smallest τ_{ik} is chosen. With some abuse of notation, "a neighbor

Algorithm 5.3.3 Iterated local search

Input parameters: $\hat{p}$ and t.
Compute a random array of costs $c \in [0,1]^o$ and, using CBFS, construct an initial solution $\vec{x}$.
Let $\vec{x}^{\text{pert}} \leftarrow \vec{x}$.
while time limit t not reached **do**
    Perform a local search starting from $\vec{x}^{\text{pert}}$ to obtain $\vec{v}$.
    if $f(\vec{v}) \leq f(\vec{x})$ **then**
        $\vec{x} \leftarrow \vec{v}$
    Let $\mathcal{R} \subseteq \{1, 2, \ldots, 2\bar{o}\}$ be a a set with $\hat{p}$ mutually exclusive random numbers
    for $i \leftarrow 1$ to $2\bar{o}$ **do**
        if $i \in \mathcal{R}$ **then** compute a random number $\gamma \in [0,1]$ and let $\vec{x}_i^{\text{pert}} \leftarrow \gamma$
        else let $\vec{x}_i^{\text{pert}} \leftarrow \vec{x}_i$.
Return $\vec{x}$.

is a tabu or not" depending on whether the action that transforms the current solution into the neighbor is tabu or not. Specifically, let *iter* be the current iteration $\vec{x}$ be the current solution. Let $\mathcal{N}(\vec{x})$ be its neighborhood and let $\vec{y} \in \mathcal{N}(\vec{x})$ be a neighbor. Moreover, assume that in $\vec{x}$ there is an operation i assigned to machine k and that the action that transforms $\vec{x}$ into $\vec{y}$ includes to remove i from k and to assign it to another machine k'. Then, $\vec{y}$ is considered a tabu neighbor of $\vec{x}$ if (i,k) is tabu, i.e., if $\tau_{ik} > iter$. Otherwise, $\vec{y}$ is a non-tabu neighbor. Algorithm 5.3.4 shows the essential steps of the considered TS algorithm.

5.4 Experimental Verification and Analysis

In this section, extensive numerical experiments with the proposed metaheuristics for the OPS scheduling problem are presented. In a first set of experiments, parameters of the proposed metaheuristics are calibrated with a reduced set of OPS instances. In a second set of experiments, additionally to the whole set of OPS instances used in Chapter 4, another 50 even larger OPS instances are generated. The calibrated methods are compared to each other and against the IBM ILOG Constraint Programming Optimizer (CPO), also considered in Chapter 4. As a result of the analysis of the performance of the proposed methods, a combined metaheuristic approach is introduced. In a last set of experiments, the best performing approach is evaluated when applied to instances of the FJS scheduling problem with sequencing flexibility and instances of the classical FJS scheduling problems considering well-known benchmark sets from the literature.

Metaheuristics were implemented in C++. Numerical experiments were conducted using a single physical core on an Intel Xeon E5-2680 v4 2.4 GHz with 4GB memory (per core) running CentOS Linux 7.7 (in 64-bit mode), at the High-Performance

Algorithm 5.3.4 Tabu search

Input parameters: $T(\lambda)$ and t.
iter $\leftarrow 0$ and $\tau_{ik} \leftarrow 0$ $(i = 1, \ldots, \bar{o},\ k = 1, \ldots, m)$.
Compute a random array of costs $c \in [0, 1]^o$ and, using CBFS, construct an initial solution $\vec{x}$.
Initialize $\vec{x}^{\,\text{best}} \leftarrow \vec{x}$.
while time limit t not reached **do**
    iter $\leftarrow$ iter $+ 1$.
    if there are at least two non-tabu neighbors in $\mathcal{N}(\vec{x})$ **then**
        Let $\vec{v}, \vec{w} \in \mathcal{N}(\vec{x})$ the two non-tabu neighbour solutions with smallest estimated makespan
        Let $\vec{y} \in \{\vec{v}, \vec{w}\}$ be randomly chosen. Let (i, k) be the action that transforms $\vec{x}$ into $\vec{y}$.
    else if there is a single non-tabu neighbor in $\mathcal{N}(\vec{x})$ **then**
        Let $\vec{y} \in \mathcal{N}(\vec{x})$ be the single non-tabu neighbour and let (i, k) be the action that transforms $\vec{x}$ into $\vec{y}$.
    else
        Let $\vec{y} \in \mathcal{N}(\vec{x})$ be a (tabu) neighbour whose associated action (i, k) has minimum τ_{ik}.
    Let $\tau_{ik} \leftarrow$ iter $+ T(\lambda)$.
    Let $\vec{x} \leftarrow \vec{y}$.
    if $f(\vec{x}) < f(\vec{x}^{\,\text{best}})$ **then** $\vec{x}^{\,\text{best}} \leftarrow \vec{x}$
Return $\vec{x}^{\,\text{best}}$.

Computing (HPC) facilities of the University of Luxembourg (Varrette et al., 2014).

5.4.1 Considered Sets of Instances

As a whole, 20 medium-sized and 100 large-sized instances of the OPS scheduling problem were considered. The set of medium-sized instances, named MOPS from now on, corresponds to the instances described in Section 4.4.3 and presented in Table 4.4. The set of large-sized instances corresponds to the set with 50 instances described in Section 4.4.4 and presented in Table 4.7, named LOPS1 from now one, plus a set with 50 additional even larger instances, named LOPS2 from now on, generated with the random instances generator procedure described in Section 4.4.1. The instances generator relies on six integer parameters, namely, the number of jobs n, the minimum $o_{\min}$ and maximum $o_{\max}$ number of operations *per* job, the minimum $m_{\min}$ and the maximum $m_{\max}$ number of machines, and the maximum number q of periods of unavailability *per* machine. The LOPS2 set contains 50 instances numbered from 51 to 100, the k-th instance being generated with the following parameters: $n = 11 + \lceil \frac{k}{100} \times 189 \rceil$, $o_{\min} = 5$, $o_{\max} = 6 + \lceil \frac{k}{100} \times 14 \rceil$, $m_{\min} = 9 + \lceil \frac{k}{100} \times 20 \rceil$, $m_{\max} = 10 + \lceil \frac{k}{100} \times 90 \rceil$, and $q = 8$. The instances generator and all considered instances are freely available at https://github.com/willtl/online-printing-shop. Table 5.1 describes the main features of the 50 instances in the set LOPS2. The union

of LOPS1 and LOPS2 will be named LOPS from now on. It is worth noticing that, although random, the OPS instances preserve the characteristics of real instances of the OPS scheduling problem. Moreover, large-sized instances are of the size of the instances that occur in practice.

In addition to the OPS instances, instances of the FJS scheduling problem with sequencing flexibility as proposed in (Birgin et al., 2014) and instances of the FJS scheduling problem as proposed in (Brandimarte, 1993; Hurink, Jurisch, and Thole, 1994; Barnes and Chambers, 1996; Dauzère-Pérès and Paulli, 1997) were considered. The instances in (Birgin et al., 2014) are divided into two sets named YFJS and DAFJS. The first set corresponds to instances with "Y-jobs" while the second set corresponds to instances in which the jobs' precedence constraints are given by certain types of directed acyclic graphs, see (Birgin et al., 2014) for more details. The sets of instances of the FJS scheduling problem were named BR, HK, BC, and DP, respectively. The HK set consists of the well-known EData, RData, and Vdata sets, with varying degrees of routing flexibility.

Table 5.2 shows the main figures of each instance set. The first two columns of the table ("Set name" and "#inst.") identify the set and the number of instances in each set. In the remaining columns, characteristics of the instances in each set are given. Column m refers to the number of machines, $\hat{q}$ refers to the number of periods of unavailability *per* machine, n is the number of jobs, $\hat{o}$ refers to the number of operations *per* job, $|V|$ is the total number of operations (i.e., $|V| = o$), $|A|$ is the total number of precedence constraints, $|T|$ is the number of fixed operations, "#overlap" is the number of operations whose processing may overlap with the processing of a successor (i.e., $|\{i \in V \mid \theta_i < 1\}|$), and "#release" is the number of operations with an *actual* release time (i.e., $|\{i \in V \mid r_i > 0\}|$). For each of these quantities, the table shows the minimum (min), the average (ave), and the maximum (max), in the form min|ave|max, over the whole considered set. It is worth noticing that, as a whole, 348 instances of different sources and nature are being considered.

Table 5.1: Main features of the fifty large-sized OPS instances in the LOPS2 set.

	Main instance characteristics						CP Optimizer formulation	
Instance	m	$\sum_{k=1}^{m} q_k$	n	o	$\|A\|$	$\|T\|$	# integer variables	# constraints
51	49	219	108	1067	1812	1	85148	249492
52	41	170	110	1076	1772	0	71169	209792
53	20	88	112	1105	1807	4	35763	105440
54	54	262	114	1137	1917	0	98745	291090
55	40	203	115	1065	1720	1	67611	199468
56	31	143	117	1098	1745	1	55292	162385
57	46	177	119	1217	2013	2	88912	261270
58	51	233	121	1274	2122	3	103766	304923
59	26	124	123	1271	2181	0	54918	162302
60	48	212	125	1346	2339	0	103373	304605
61	50	228	127	1358	2381	4	106864	314187
62	32	130	129	1290	2133	0	68060	199865
63	41	144	131	1370	2297	1	90142	265633
64	54	257	132	1421	2442	3	122801	361440
65	55	264	134	1427	2384	1	125843	370335
66	63	281	136	1523	2627	0	152642	449811
67	64	304	138	1499	2621	2	153859	452214
68	38	158	140	1579	2750	2	97716	288436
69	40	171	142	1577	2739	1	99887	294099
70	37	147	144	1588	2755	4	95755	281382
71	53	247	146	1590	2734	0	136147	400311
72	70	354	148	1701	2952	4	185238	544518
73	32	132	149	1778	3174	3	92225	271844
74	29	125	151	1726	3000	1	82365	242813
75	33	167	153	1744	3077	0	94906	278965

continues on next page

Table 4.7 continued

	Main instance characteristics						CP Optimizer formulation	
Instance	m	$\sum_{k=1}^{m} q_k$	n	o	$\|A\|$	$\|T\|$	# integer variables	# constraints
76	58	247	155	1757	3054	0	161749	475363
77	71	292	157	1793	3137	1	201915	594077
78	79	367	159	1789	3105	0	223365	655201
79	74	324	161	1798	3121	2	212354	626118
80	80	355	163	1850	3202	1	231248	680674
81	49	207	165	2080	3785	2	164905	485323
82	29	139	166	2063	3763	0	98767	291587
83	49	207	168	2044	3565	2	158986	468564
84	78	357	170	2082	3753	0	256059	753223
85	61	251	172	2047	3700	4	202088	593823
86	67	301	174	2133	3827	6	226909	666635
87	56	273	176	2215	4006	0	198539	584436
88	27	130	178	2141	3953	1	95622	282029
89	45	188	180	2299	4187	3	166199	488842
90	51	255	182	2213	4020	1	181658	534436
91	72	341	183	2340	4276	1	266059	782641
92	56	246	185	2400	4418	0	215525	634439
93	85	374	187	2399	4386	0	320129	941218
94	38	153	189	2447	4431	0	150904	444416
95	73	337	191	2568	4721	1	299347	879356
96	60	310	193	2508	4565	2	237394	698041
97	70	324	195	2443	4530	1	268046	788064
98	32	173	197	2579	4667	2	134682	397669
99	97	433	199	2548	4649	2	390037	1148630
100	58	247	200	2661	5032	3	246960	727868

Table 5.2: Main features of the considered sets of instances.

Set name	#inst.	m	$\hat{q}$	n	$\hat{o}$	$\|V\|$	$\|A\|$	$\|T\|$	#overlap	#release
MOPS	20	6|10|17	25|48|75	5|8|10	6|9|14	36|67|109	54|106|207	0|1|3	0|7|16	0|1|6
LOPS	100	10|37|97	44|168|433	13|106|200	5|10|22	79|1153|2661	95|1985|5032	0|1|6	7|115|270	0|28|79
YFJS	20	7|14|26	0	4|10|17	4|10|17	24|115|289	18|105|272	0	0	0
DAFJS	30	5|7|10	0	4|7|12	4|9|23	25|71|120	23|66|117	0	0	0
BR	10	4|8|15	0	10|15|20	3|9|15	55|141|240	45|125|220	0	0	0
HK	129	5|8|15	0	6|16|30	5|8|15	36|145|300	30|128|270	0	0	0
BC	21	11|13|18	0	10|13|15	10|11|15	100|158|225	90|145|210	0	0	0
DP	18	5|7|10	0	10|15|20	15|19|25	196|292|387	186|277|367	0	0	0

5.4.2 Parameters Tuning

In this section, the performance of the proposed metaheuristics are evaluated under variations of their parameters. In this way, thirty OPS instances were used to fine-tune each parameter of each metaheuristic. The set of instances was composed of the five most difficult instances from the MOPS set according to the numerical results presented in Table 4.6 plus twenty-five representative instances from the LOPS set, namely, instances $1, 5, 9, 13, \ldots, 97$. Since methods whose parameters are being calibrated have a random component, for each desired combination of parameters, each method was applied to each instance ten times. For each run, a CPU time limit of 1200 seconds was imposed.

Assume that the combinations of parameters $c_1, c_2, \ldots, c_A$ for method M applied to the set of instances $\{p_1, p_2, \ldots, p_B\}$ are to be evaluated. Let $f(M(c_\alpha), p_\beta)$ be the average makespan over the ten runs of method M with the combination of parameters c_α applied to instance p_β for $\alpha = 1, \ldots, A$ and $\beta = 1, \ldots, B$. Let

$$f_{\text{best}}(M, p_\beta) = \min_{\{\alpha=1,\ldots,A\}} \{f(M(c_\alpha), p_\beta)\}, \text{ for } \beta = 1, \ldots, B,$$

$$f_{\text{worst}}(M, p_\beta) = \max_{\{\alpha=1,\ldots,A\}} \{f(M(c_\alpha), p_\beta)\}, \text{ for } \beta = 1, \ldots, B,$$

and

$$\text{RDI}(M(c_\alpha), p_\beta) = \frac{f(M(c_\alpha), p_\beta) - f_{\text{best}}(M, p_\beta)}{f_{\text{worst}}(M, p_\beta) - f_{\text{best}}(M, p_\beta)}, \text{ for } \alpha = 1, \ldots, A \text{ and } \beta = 1, \ldots, B.$$

Thus, for every α and β, $\text{RDI}(M(c_\alpha), p_\beta) \in [0, 1]$ indicates the performance of method M with the combination of parameters c_α applied to instance p_β with respect to the performance of the same method with other combinations of parameters. The smaller the $\text{RDI}(M(c_\alpha), p_\beta)$, the better the performance. In particular, $\text{RDI}(M(c_\alpha), p_\beta) = 0$ if and only if $f(M(c_\alpha), p_\beta) = f_{\text{best}}(M, p_\beta)$ and $\text{RDI}(M(c_\alpha), p_\beta) = 1$ if and only if $f(M(c_\alpha), p_\beta) = f_{\text{worst}}(M, p_\beta)$. If defined

$$\text{RDI}(M(c_\alpha)) = \frac{1}{|B|} \sum_{\beta=1}^{B} \text{RDI}(M(c_\alpha), p_\beta), \text{ for } \alpha = 1, \ldots, A,$$

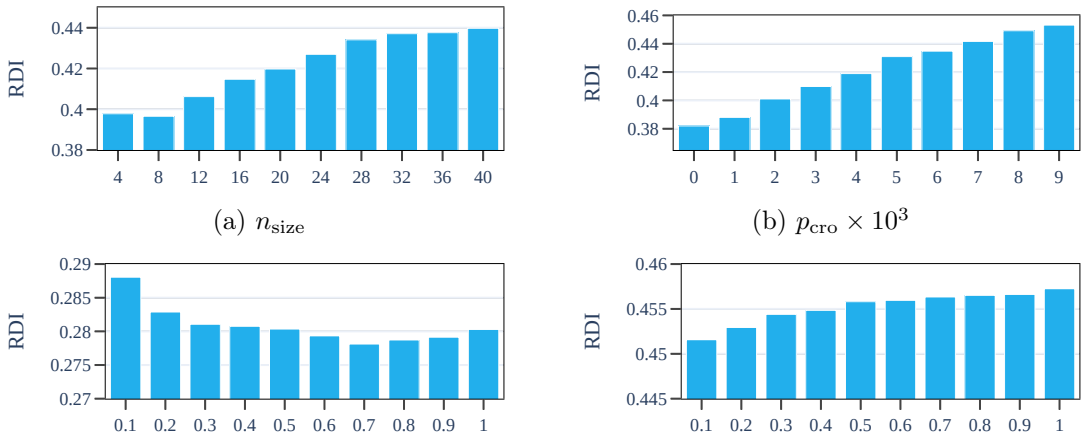
then the combination of parameters c_α with the smallest $\text{RDI}(M(c_\alpha))$ is the one for which method M performed best.

5.4.2.1 Differential Evolution

In DE there are four parameters to be calibrated, namely, n_{size}, p_{cro}, ζ, and *variant*. Preliminary experiments indicated that varying these parameters within the ranges $n_{\text{size}} \in [4, 40]$, $p_{\text{cro}} \in [0, 0.01]$, $\zeta \in [0, 1]$, and *variant* $\in \{\text{DE/rand/1}, \text{DE/best/1}\}$ would provide acceptable results. Since testing all combinations in a grid would be very time consuming, it was arbitrarily proceeded as follows. First, $n_{\text{size}} \in \{4, 8, 12, \ldots, 40\}$ was varied with the following parameters set to a fixed value, $p_{\text{cro}} = 0.005$, $\zeta = 0.5$, and variant = DE/rand/1. Figure 5.6a shows the RDI for the different values of n_{size}. The figure shows that the method achieved its best performance at $n_{\text{size}} = 8$. In a second experiment, $p_{\text{cro}} \in \{0, 10^{-3}, 2 \times 10^{-3}, \ldots, 9 \times 10^{-3}\}$ was varied with the following fixed parameters, $n_{\text{size}} = 8$, $\zeta = 0.5$, *variant* = DE/rand/1. Figure 5.6b shows that the best performance was obtained with $p_{\text{cro}} = 0$. In a third experiment, $\zeta \in \{0.1, 0.2, \ldots, 1\}$ with the following fixed parameters, $n_{\text{size}} = 8$, $p_{\text{cro}} = 0$, *variant* = DE/rand/1. Figures 5.6c and 5.6d show the results for the five problems in the MOPS set and the twenty five instances in the LOPS set,; respectively. The results demonstrate that the best performance is obtained for $\zeta = 0.7$ and $\zeta = 0.1$, respectively. It is worth noticing that the performance of the method varies smoothly as a function of its parameters as indicated by Figures 5.6a–5.6d. Finally, Figures 5.7a and 5.7b show the performance of the algorithm with $n_{\text{size}} = 8$, $p_{\text{cro}} = 0$, and $\zeta = 0.7$ applied to the five instances from the MOPS set and with $n_{\text{size}} = 8$, $p_{\text{cro}} = 0$, and $\zeta = 0.1$ applied to the twenty five instances from the LOPS set. In both cases, the figures compare the performance for variations of *variant* $\in \{\text{DE/rand/1}, \text{DE/best/1}\}$. The considered mutation variants are the two most widely adopted ones in the literature. The main difference between both of them is that the former emphasizes exploration while the latter emphasizes exploitation. In this experiment, the time limit was extended to 1 hour. Figures 5.7a and 5.7b show the average makespan over the considered subsets of instances as a function of time. Both graphics show that a choice of *variant* = DE/rand/1 is more efficient.

5.4.2.2 Genetic Algorithm

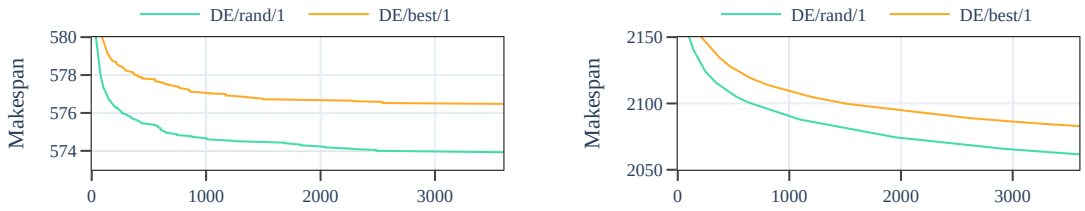
In the proposed GA there are two parameters to be calibrated, namely, n_{size} and p_{mut}. Preliminary experiments indicated that varying these parameters within the ranges $n_{\text{size}} \in [4, 40]$ and $p_{\text{mut}} \in [0.01, 0.5]$ would provide acceptable results. In a first experiment, $n_{\text{size}} \in \{4, 8, \ldots, 40\}$ was varied with fixed $p_{\text{mut}} = 0.25$. Figure 5.8a shows that the best performance is obtained with $n_{\text{size}} = 8$. In a second experiment, $n_{\text{size}} = 8$ was fixed and $p_{\text{mut}} \in \{0.01, 0.06, \ldots, 0.46\}$ was varied. Figures 5.8b and 5.8c show that the best performance is obtained with $p_{\text{mut}} = 0.36$ when the method is applied to the



(a) n_{size} (b) $p_{\text{cro}} \times 10^3$

(c) ζ (five MOPS instances) (d) ζ (twenty five LOPS instances)

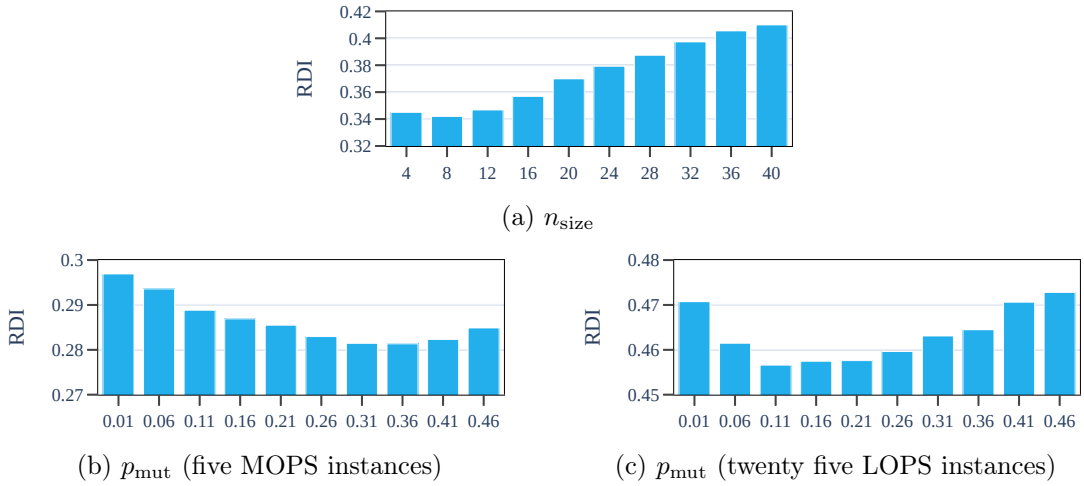
Figure 5.6: DE performance for different parameters' settings.



(a) five MOPS instances (b) twenty five LOPS instances

Figure 5.7: Evolution of the average makespan as a function of time obtained with DE (a) with $n_{\text{size}} = 8$, $p_{\text{cro}} = 0$, and $\zeta = 0.7$ applied to the five selected instances from the MOPS set and (b) with $n_{\text{size}} = 8$, $p_{\text{cro}} = 0$, and $\zeta = 0.1$ applied to the twenty five selected instances from the LOPS set.

five selected instances from the MOPS set; while its best performance is obtained with $p_{\text{mut}} = 0.11$ when applied to the twenty five selected instances from the LOPS set. It can be observed that, as is happened with DE, the best population size is $n_{\text{size}} = 8$ and it does not depend on the size of the instances. On the other hand, the same behavior is not observed for the mutation probability parameter p_{mut}. Similar to the parameter ζ of DE that appears in its mutation scheme, a different behavior is observed when the method is applied to instances from the MOPS and the LOPS sets. At this point, it is important to stress that this should not be considered problematic. The main goal is to develop an efficient and effective method to be applied to practical instances of the OPS scheduling problem, i.e., to a real-world problem; and these instances are very similar to the instances in the LOPS set. Numerical experimentation with the MOPS instances is carried out for assessment purpose, comparing the obtained results with the ones presented in Chapter 4, which include numerical experiments with instances of the MOPS set.



(a) n_{size}

(b) p_{mut} (five MOPS instances)

(c) p_{mut} (twenty five LOPS instances)

Figure 5.8: Genetic Algorithm performance for different parameters' settings.

5.4.2.3 Iterated Local Search and Tabu Search

ILS and TS have a single parameter to calibrate, namely $\hat{p}$ and λ, respectively. Preliminary experiments indicated that varying these parameters within the ranges $\hat{p} \in \{1, 2, \ldots, 10\}$ and $\lambda \in [0.6, 1.5]$ would provide acceptable results. Figures 5.9 and 5.10 show the results varying $\hat{p} \in \{1, 2, \ldots, 10\}$ and $\lambda \in \{0.6, 0.7, \ldots, 1.5\}$, respectively. They show that ILS performed best with $\hat{p} = 2$; while TS obtained the best results with $\lambda = 1.2$. It is worth noticing that, in both cases, the performance varies smoothly as a function of the parameters; thus similar performances are obtained for small variations of the parameters.

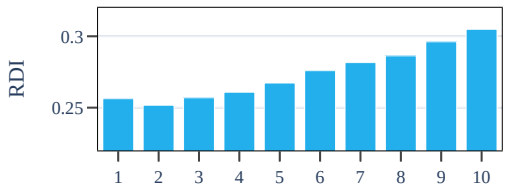


Figure 5.9: Iterated Local Search performance as a function of its solely parameter $\hat{p}$.

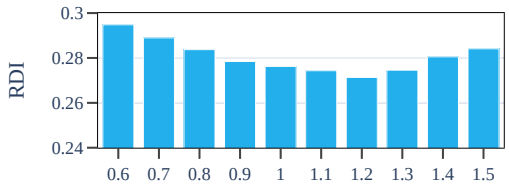


Figure 5.10: Tabu Search performance as a function of its solely parameter λ.

5.4.3 Experiments With OPS Instances

This section presents numerical experiments with the four calibrated metaheuristics DE, GA, ILS, and TS. In addition, the performance of the IBM ILOG CPO (Laborie et al., 2018), version 12.9, is presented. In this experiments, the two-phases strategy "Incomplete model + CP Model 4" described in Chapter 4 is considered. This approach consists of first solving a simplified model and, in a second phase, using the solution obtained in the first phase as the initial solution to the full and more complex model. This is the approach that performed best among several alternative CP models and solution strategies considered in Chapter 4.

Numerical experiments consider the 20 instances in the MOPS set and the 100 instances in the LOPS set. Each metaheuristic was run 50 times in each instance of the MOPS set and 12 in each instance of the LOPS set. As described in Section 5.4.2, the *average* over all runs is considered for comparison purposes. For each run, a CPU time limit of 2 hours was imposed. The metaheuristics being evaluated start from a feasible solution and generate a sequence of feasible solutions. Thus, it is possible to observe the evolution of the makespan over time. This is not the case of the strategy of the CPO being considered. In the two-phases strategy, 2/3 of the time budget is allocated to the solution of a relaxed or incomplete OPS formulation in which setup operations can be preempted and the setup of the first operation to be processed in each machine is considered to be null; while the remaining 1/3 of the time budget is allocated to the solution of the actual CP formulation of the OPS scheduling problem. Due to the two-phases strategy, it is not possible to track the evolution of the makespan over time, since in the first 2/3 of the time budget the incumbent solution is, with high probability, infeasible. Therefore, to compare the performance of the CPO against the proposed methods, CPO was run several times with increasing time budgets given by 5 minutes, 30 minutes, and 2 hours per instance.

Figure 5.11 shows the evolution of the average makespan (over the 50 runs and over all instances) when the five methods are applied to the instances in the MOPS set. Table 5.3 presents the *best* makespan and Table 5.4 presents *average* makespan obtained by each metaheuristic method in each instance. The last line named "Mean" in each table presents the average results. The last line in Table 5.4 exhibits the Pooled Standard Deviation (PSD). For each instance, figures in bold represent the best result obtained by the methods under consideration. Average makespans and PSDs are graphically represented in Figure 5.12. Method TS+DE that appears in the figures and the table should be ignored at this time. The motivation for its definition as well as its presence in the experiments will be elucidated later in the current section. Table 5.5 shows the results of applying CPO to instances in the MOPS set. In the table, UB corresponds to the best solution found (UB to the optimal solution); while

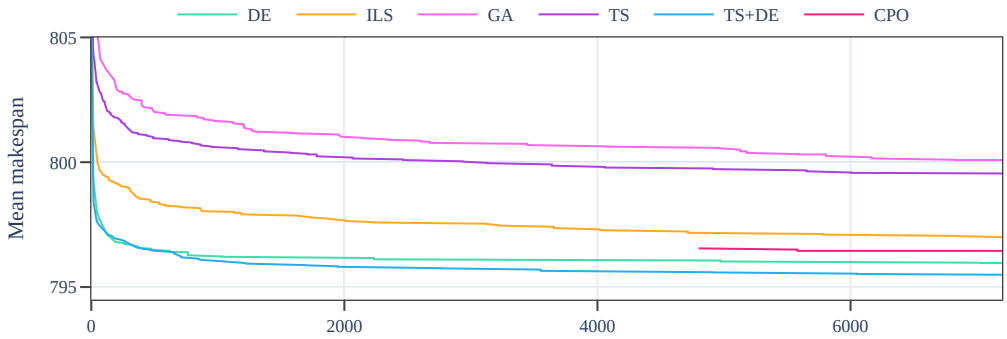


Figure 5.11: Evolution of the average makespan over time of each proposed method and CPO applied to the MOPS set instances.

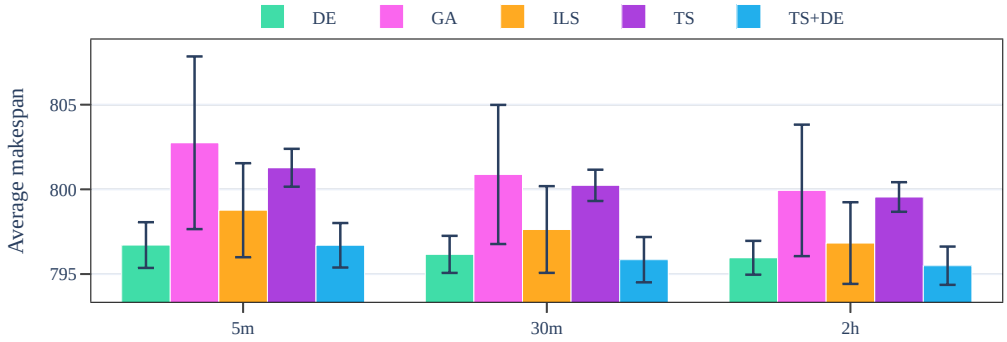


Figure 5.12: Average makespans and pooled standard deviations of applying the proposed metaheuristic approaches fifty times to instances in the MOPS set with CPU time limits of 5 minutes, 30 minutes, and 2 hours.

LB corresponds to the computed LB when the CPU time limit is equal to two hours. A comparison between the lower and the UB shows that the optimal solution was found for instances 1–5, 7, 9, 10, 13, and 15–19; while a non-null gap is reported for instances 6, 8, 11, 12, 14, and 20.

The results presented in Figure 5.11 show that DE outperforms any other method at any instant in time if the average makespan is considered. Recalling that CPO does not produce feasible solutions in the first 2/3 of the time budget, the comparison of DE with CPO requires the analysis of the results in Tables 5.3, 5.4, and 5.5. The results in the tables show that DE outperforms CPO when the CPU time limit is 5 minutes, 30 minutes, or 2 hours. Results in the tables show that DE outperforms CPO also when the performance measure is the best makespan instead of the average makespan. The method that ranks in second place depends on the time limit and the performance measure (average or best makespan). Depending on the choice, CPO or ILS achieve second best result. The second place belongs to CPO when the average makespan is considered or when the CPU time limit is 2 hours. If the performance measure is the best makespan and the CPU time limit is 5 minutes or 30 minutes, the second place belongs to ILS. Concerning the best makespan and considering a CPU

time limit of 2 hours, DE, GA, ILS, TS, and CPO obtained the best makespan 17, 10, 12, 12, and 13 times, respectively. Note that these numbers are slightly influenced by the presence of the method TS+DE that should be ignored. This is because TS+DE was the only method to find the best makespan in instance 6; so this instance is not computed for TS, that was the only method that found the second-best makespan for this instance. In any case, considering the average makespan, it is worth noting that, depending on whether the CPU time limit is 5 minutes, 30 minutes, or 2 hours, the difference between the methods that rank in first and last places is not larger than 0.8%, 0.7%, or 0.6%, respectively.

Table 5.3: Results of applying the metaheuristic approaches to instances in the MOPS set.

	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
1	**344**	351	346	**344**	**344**	**344**	350	346	**344**	**344**	**344**	350	346	**344**	**344**
2	**357**	358	358	**357**	**357**	**357**	358	**357**	**357**	**357**	**357**	**357**	**357**	**357**	**357**
3	**405**	409	407	409	**405**	405	409	407	409	**404**	405	409	405	408	**404**
4	**458**	**458**	**458**	**458**	**458**	**458**	**458**	**458**	**458**	**458**	**458**	**458**	**458**	**458**	**458**
5	**507**	516	510	**507**	**507**	**507**	516	510	**507**	**507**	**507**	516	509	**507**	**507**
6	435	447	436	437	**432**	435	446	436	434	**432**	435	442	436	433	**432**
7	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**
8	**447**	459	453	461	448	**446**	451	453	456	447	**445**	451	451	456	447
9	**629**	632	630	633	**629**	**629**	631	630	631	**629**	**629**	631	**629**	630	**629**
10	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**
11	**413**	427	419	433	414	**413**	426	414	433	**413**	**413**	423	414	430	**413**
12	**491**	500	496	511	492	**489**	492	492	511	**489**	**489**	492	492	507	**489**
13	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**
14	392	404	396	412	**389**	391	404	393	408	**389**	**389**	400	391	408	**389**
15	320	320	**319**	**319**	**319**	320	**319**	**319**	**319**	**319**	320	**319**	**319**	**319**	**319**
16	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**
17	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**
18	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**
19	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**
20	**507**	519	521	538	511	**507**	518	514	534	**507**	**507**	514	514	534	**507**
Mean	**794.75**	799.5	796.95	800.45	**794.75**	794.55	798.4	795.95	799.55	**794.25**	794.4	797.6	795.55	799.05	**794.25**

Table 5.4: Results of the average makespan applying the metaheuristic approaches to instances in the MOPS set.

	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
1	346.25	361.2	349.2	**344**	344.12	346	361	348	**344**	344.12	346	360.8	347.6	**344**	344.12
2	357.75	361.6	361.2	**357.25**	357.88	357.75	360.8	359.2	**357**	357.88	357.75	359	357.4	**357**	357.88
3	408.25	417.4	408.4	409.5	**407.62**	407	416	408.4	409	**406.5**	406.25	414.8	407.6	408.25	**406.12**
4	**458**	461.6	**458**	**458**	**458**	**458**	460	**458**	**458**	**458**	**458**	459	**458**	**458**	**458**
5	511	521.2	511.8	510	**509.12**	509.5	518.6	511.6	**508**	508.5	**508**	518.6	510.2	**508**	508.5
6	436.5	457.2	441.6	438	**436.12**	436.25	449	441	**435.5**	435.62	436.25	447.8	441	**433.75**	435.62
7	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**	**2429**
8	**451.25**	463	459.2	462	452.5	**450.5**	461	456.2	458.75	451.12	**450**	460.6	453.6	456.5	450.62
9	**630.5**	638	630.8	637	**630.5**	630	632.6	630	633.5	**629.88**	629.75	631.8	629.6	632.25	**629.5**
10	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**	**1184**
11	421.5	428.4	421.6	434.25	**419.62**	420	426.2	416.8	433.5	**416.5**	420	423.6	416.6	430.75	**416**
12	**495**	504.2	501.2	512	496.75	494.75	497.6	497.8	511.25	**493.5**	494	494.6	495.8	508.5	**493**
13	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**	**347**
14	396.5	408.2	401.4	414.25	**395.5**	394.25	404.4	397.6	410.5	**393.88**	394	400.8	394.8	409	**393.12**
15	320	320	319.6	**319**	319.5	320	319.6	**319**	**319**	319.5	320	319.2	**319**	**319**	319.5
16	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**	**543**
17	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**	**1052**
18	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**	**3184**
19	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**	**1451**
20	**511.75**	523	521.4	540.25	516.75	**509.25**	520.8	519	536.75	512	509.25	518.2	515.4	536	**507.88**
Mean	796.71	802.75	798.77	801.27	**796.7**	796.16	800.88	797.63	800.24	**795.85**	795.96	799.94	796.83	799.55	**795.49**
PSD	1.35	5.10	2.77	1.12	1.31	1.09	4.11	2.56	0.92	1.34	1.00	3.88	2.41	0.87	1.13

Table 5.5: Results of applying CPO to instances in the MOPS set.

Instance	5m	30m	2h		Instance	5m	30m	2h		Instance	5m	30m	2h		Instance	5m	30m	2h	
	UB	UB	LB	UB		UB	UB	LB	UB		UB	UB	LB	UB		UB	UB	LB	UB
1	344	344	344	344	6	441	441	335	441	11	418	418	406	418	16	543	543	543	543
2	357	357	357	357	7	2429	2429	2429	2429	12	506	497	457	499	17	1080	1052	1052	1052
3	404	404	404	404	8	456	450	360	450	13	347	347	347	347	18	3184	3184	3184	3184
4	458	458	458	458	9	632	629	629	629	14	402	402	320	394	19	1451	1451	1451	1451
5	506	506	506	506	10	1184	1184	1184	1184	15	319	319	319	319	20	522	522	417	520
															Mean	799.1	796.9		796.5

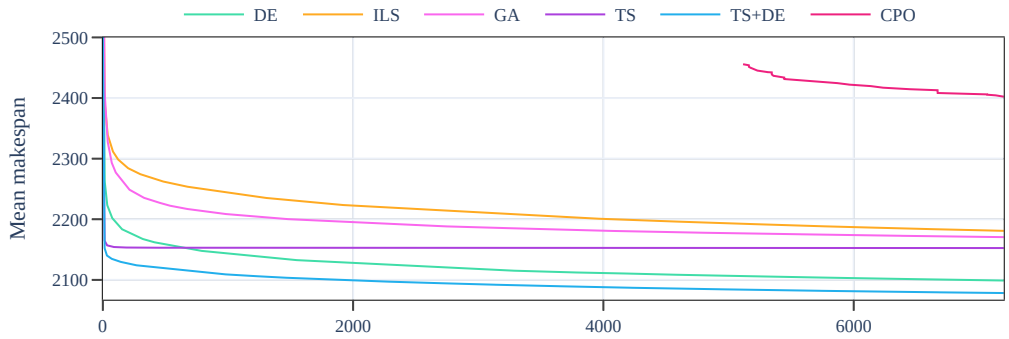


Figure 5.13: Evolution of the average makespan over time of each proposed method and CPO applied to the LOPS set instances.

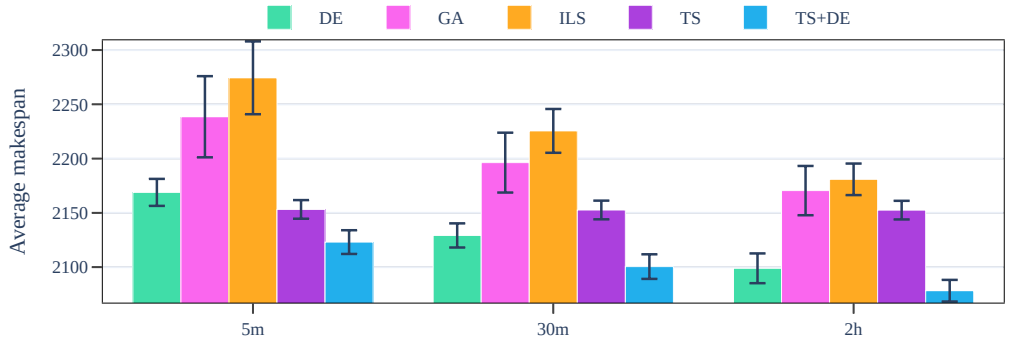


Figure 5.14: Average makespans and pooled standard deviations of applying the proposed metaheuristic approaches twelve times to instances in the LOPS set with CPU time limits of 5 minutes, 30 minutes, and 2 hours.

Figure 5.13 shows the evolution of the average makespan (over the 12 runs and over all instances) when the five methods are applied to the instances in the LOPS set. Tables 5.6–5.7 present respectively the best makespan and the average makespan obtained by each metaheuristic method in each instance when the CPU time limit is 5 minutes, 30 minutes, or 2 hours. For each instance, numbers in bold represent the best results obtained by the methods under consideration. Method TS+DE should still be ignored. The last line named "Mean" in each table presents the average results. In Table 5.7, and additional line exhibits the PSD. Average makespans and PSDs are graphically represented in Figure 5.14. Table 5.8 shows the results of applying CPO to the instances in the LOPS set. The symbol " — " means that CPO was not able to find a feasible solution within the time budget.

The results in Figure 5.13 show that, differently from the previous experiments with the medium-sized OPS instances, in the large-sized instances no method obtains the smallest average makespan regardless of the considered time instant. TS outperforms all the other methods for any instant $t \leq 650$ seconds while DE outperforms all the other methods for any instant $t \geq 650$ seconds. Another difference concerning the medium-sized instances is that CPO was outperformed by all introduced metaheuristic

approaches. The numerical values in Tables 5.6 and 5.7 reflect the results already observed in Figure 5.13. TS found the best results for small-time limits while DE found the best results for large time limits. The average results show that the methods rank (a) TS, DE, GA, ILS, and CPO; (b) DE, TS, GA, ILS, and CPO; and (c) DE, TS, GA, ILS, and CPO, when the CPU time limit is 5 minutes, 30 minutes, and 2 hours, respectively, independent of considering the *best* or the *average* makespan as a performance measure.

The observations described in the paragraph above led us to consider a combined approach, named TS+DE, that uses TS to construct an initial population for DE. The combined approach has three phases. In the first phase, TS is used to obtain a solution. Instead of running the method until it reaches the CPU time limit, the search is stopped if the incumbent solution is not updated during a period of $\log_{10}(o)$ seconds o CPU time, recalling that o is the number of operation of an instance. In the second phase, a population is constructed by running the LS procedure starting from $n_{\text{size}} - 1$ perturbations of the TS solution. The perturbation procedure is the one described for the ILS algorithm in Section 5.3. The solution of the TS plus the $n_{\text{size}} - 1$ solutions found with the LS constitute the initial population of DE. Running DE with this initial population is the third phase of the strategy. The three-phases strategy is interrupted at any time if the CPU time limit is reached. In this strategy, parameters of TS, DE, and the perturbation procedure of ILS were set as already calibrated for each individual method.

Figure 5.11 and Tables 5.3–5.4 show the performance of the combined approach when applied to the MOPS set, while Figure 5.13 and Tables 5.6–5.7 show the performance of the combined approach when applied to the LOPS set. Specifically for the instances in the MOPS set, TS+DE (with a CPU time limit of at least 30 minutes) finds the optimal solutions in the 14 instances with the known optimal solution and improves the solutions found by CPO in the 6 instances with a non-null gap. Figures and tables show that TS+DE is the most successful approach. It always found the lowest average makespan in the MOPS and LOPS sets independent of the CPU time limit imposed. It found the lowest best and average makespans and it found the largest number of best solutions among all considered methods, outperforming CPO to a large extent. It is worth noting that, in the LOPS set, considering the average makespan, the difference between the metaheuristics that rank in first and last places is not larger than 7%, 6%, or 5%, depending on whether the CPU time limit is 5 minutes, 30 minutes, or 2 hours, respectively. This result is not surprising since the four metaheuristic approaches share the representation scheme and the definition of the neighborhood in the LS strategy. On the other hand, the difference between TS+DE and CPO, with a CPU time limit of 2 hours, is 16%.

Table 5.6: Best makespan of applying the metaheuristics to the first-half of the instances in the LOPS set.

Instance	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
1	**516**	525	**516**	527	**516**	**516**	522	**516**	526	**516**	**513**	520	516	524	514
2	**641**	655	647	660	642	**641**	651	647	659	**641**	**638**	648	642	658	639
3	620	623	622	644	**617**	615	622	616	644	**614**	613	618	**612**	640	613
4	741	750	**736**	769	742	**736**	743	**736**	767	738	**736**	741	**736**	767	737
5	826	836	829	857	**824**	**820**	831	821	856	821	**820**	824	**820**	854	**820**
6	689	691	690	726	**683**	681	683	679	724	**678**	676	679	**674**	724	675
7	**896**	903	**896**	935	897	890	895	892	935	**888**	**886**	894	890	935	**886**
8	**1007**	1014	1013	1049	1010	1001	1005	1004	1049	**1000**	999	1002	999	1049	**997**
9	919	921	920	969	**915**	**905**	914	911	969	906	902	910	906	969	**900**
10	765	775	768	829	**762**	**748**	764	754	829	752	745	755	749	829	**744**
11	1182	1191	1182	1217	**1174**	1165	1183	1167	1217	**1162**	1160	1164	1161	1217	**1153**
12	1168	1183	1170	1227	**1164**	**1146**	1162	1150	1227	1149	**1138**	1155	**1138**	1227	1140
13	**988**	1001	994	1055	990	**971**	986	976	1055	974	**961**	980	965	1055	965
14	1443	1450	1443	1498	**1436**	1430	1443	1428	1498	**1427**	1421	1431	**1419**	1498	**1419**
15	1386	1398	1384	1454	**1380**	1366	1384	**1360**	1454	1362	1355	1373	1356	1454	**1352**
16	1311	1327	**1306**	1366	1312	1293	1308	**1288**	1366	1293	1284	1301	**1280**	1366	1282
17	**1041**	1061	**1041**	1085	1045	**1028**	1046	1030	1085	1029	1019	1030	**1016**	1085	1021
18	1885	1898	**1880**	1956	1885	1862	1875	**1855**	1956	1859	1848	1858	**1840**	1956	1843
19	990	1007	997	1025	**989**	978	992	980	1025	**974**	**962**	985	964	1025	968
20	**965**	988	971	1013	967	**948**	969	952	1013	949	**932**	955	934	1013	934
21	1879	1894	1881	1948	**1878**	**1852**	1866	1853	1948	1854	1837	1850	**1834**	1948	1835
22	1417	1424	1442	1477	**1402**	1380	1404	1406	1477	**1359**	1361	1381	1383	1477	**1349**
23	**1070**	1083	1074	1105	1074	**1050**	1062	1056	1105	1059	1038	1050	**1037**	1105	1040
24	**1914**	1935	1921	1974	1919	1884	1905	1894	1974	**1882**	1859	1887	1870	1974	**1857**
25	1227	1245	1238	1272	**1222**	**1204**	1223	1209	1272	1205	**1189**	1208	1194	1272	1191

continues on next page

Table 5.6 continued

Instance	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
26	1281	1306	1293	1309	**1279**	**1256**	1284	1259	1309	1261	**1237**	1264	1238	1309	1243
27	1698	1718	1715	1753	**1696**	**1670**	1683	1677	1753	**1670**	1652	1674	1651	1753	**1648**
28	1929	1944	1953	1988	**1926**	1885	1925	1901	1988	**1877**	1858	1892	1879	1988	**1853**
29	2011	2072	2097	2098	**1977**	1950	2017	2009	2098	**1943**	1909	1986	1958	2098	**1905**
30	1557	1571	1576	1586	**1548**	1521	1546	1535	1586	**1516**	1496	1527	1510	1586	**1487**
31	1164	1185	1221	1179	**1133**	1128	1155	1167	1179	**1103**	1100	1136	1127	1179	**1089**
32	1062	1079	1094	1086	**1058**	1050	1062	1057	1086	**1043**	1034	1053	1039	1086	**1030**
33	2095	2114	2145	2151	**2092**	2058	2094	2086	2151	**2052**	2033	2075	2053	2151	**2025**
34	1438	1429	1465	1437	**1390**	1391	1405	1419	1437	**1361**	1356	1390	1388	1437	**1342**
35	**2772**	2835	2877	2895	2795	**2732**	2789	2795	2895	2740	**2689**	2754	2726	2895	2694
36	2482	2504	2549	2544	**2478**	2446	2482	2492	2544	**2445**	2419	2463	2443	2544	**2417**
37	1275	1299	1307	1287	**1271**	1253	1278	1281	1287	**1247**	**1236**	1263	1263	1287	1238
38	1159	1164	1182	1169	**1145**	1145	1142	1154	1169	**1135**	1125	1134	1133	1169	**1119**
39	1756	1754	1787	1760	**1733**	1721	1739	1758	1760	**1706**	1692	1714	1728	1760	**1688**
40	2204	2220	2261	2226	**2181**	2174	2200	2213	2226	**2154**	2142	2186	2163	2226	**2131**
41	2316	2344	2345	2304	**2251**	2268	2307	2281	2304	**2229**	2231	2275	2246	2304	**2194**
42	1582	1605	1655	1559	**1539**	1546	1565	1591	1559	**1521**	1520	1551	1551	1559	**1502**
43	2523	2540	2574	2548	**2507**	2490	2535	2526	2548	**2486**	2457	2500	2496	2548	**2449**
44	3678	3776	3839	3695	**3638**	3645	3715	3771	3695	**3571**	3578	3663	3702	3695	**3516**
45	2060	2065	2143	2069	**2051**	2043	2054	2095	2069	**2033**	2021	2044	2055	2069	**2014**
46	2185	2199	2236	2220	**2180**	2153	2185	2187	2220	**2150**	**2122**	2163	2150	2220	2123
47	3413	3539	3689	3438	**3363**	3356	3491	3602	3438	**3330**	3297	3454	3515	3438	**3273**
48	1272	1287	1313	1260	**1242**	1250	1274	1276	1260	**1231**	1231	1255	1256	1260	**1218**
49	2862	2893	2933	2889	**2851**	2836	2876	2886	2889	**2829**	2816	2844	2844	2889	**2804**
50	1374	1382	1403	1369	**1347**	1349	1369	1380	1369	**1333**	1331	1356	1359	1369	**1321**

continues on next page

Table 5.6 continued

Instance	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
51	1567	1590	1649	1595	**1557**	1549	1573	1608	1595	**1543**	1530	1554	1573	1595	**1527**
52	1956	2034	2099	1952	**1913**	1920	1984	2040	1952	**1888**	1886	1952	1985	1952	**1868**
53	3629	3639	3678	3668	**3623**	3590	3614	3643	3668	**3588**	3555	3585	3599	3668	**3549**
54	1553	1578	1596	1542	**1535**	1535	1554	1563	1542	**1525**	1526	1542	1545	1542	**1514**
55	2001	2021	2082	1998	**1986**	1975	1999	2036	1998	**1970**	1953	1984	1993	1998	**1944**
56	2582	2584	2635	2577	**2554**	2552	2562	2602	2577	**2535**	2528	2537	2567	2577	**2509**
57	1917	1947	1989	1892	**1878**	1888	1936	1960	1892	**1865**	1873	1892	1907	1892	**1857**
58	1814	1841	1890	1807	**1779**	1790	1822	1853	1807	**1764**	1769	1795	1802	1807	**1739**
59	3415	3440	3485	3432	**3411**	3393	3418	3448	3432	**3388**	3356	3394	3408	3432	**3355**
60	1994	2044	2056	2004	**1981**	1970	2015	2031	2004	**1966**	1948	1998	2001	2004	**1945**
61	1988	2049	2127	2014	**1980**	**1950**	2034	2078	2014	1953	**1930**	2003	2032	2014	1934
62	3224	3289	3346	3138	**3113**	3137	3238	3284	3138	**3108**	3107	3200	3226	3138	**3075**
63	2512	2623	2727	2522	**2495**	2451	2574	2657	2522	**2448**	**2412**	2561	2571	2522	2416
64	1875	1933	1973	1911	**1866**	1858	1914	1948	1911	**1857**	**1842**	1897	1915	1911	1844
65	1909	1964	2047	1935	**1893**	1880	1951	2030	1935	**1874**	1860	1939	1994	1935	**1852**
66	1773	1813	1892	1761	**1748**	1750	1802	1878	1761	**1747**	1743	1794	1837	1761	**1734**
67	1687	1734	1755	1678	**1671**	1670	1714	1741	1678	**1666**	1658	1682	1712	1678	**1653**
68	3259	3385	3472	3074	**3023**	3081	3315	3425	3068	**2967**	2967	3272	3337	3068	**2922**
69	3037	3406	3479	2921	**2891**	2902	3296	3346	2921	**2863**	2844	3223	3220	2921	**2831**
70	3240	3269	3508	3161	**3123**	3121	3248	3461	3161	**3053**	2997	3227	3376	3161	**2973**
71	2179	2247	2280	2189	**2173**	2167	2228	2252	2189	**2158**	2145	2217	2227	2189	**2142**
72	1957	2031	2073	1802	**1790**	1880	1986	2024	1802	**1787**	1802	1964	1961	1802	**1763**
73	3967	3992	4086	3940	**3915**	3907	3981	4062	3940	**3902**	**3867**	3964	4013	3940	3869
74	4342	4355	4389	4260	**4238**	4228	4341	4354	4260	**4199**	4171	4290	4313	4260	**4157**
75	3635	3667	3712	3655	**3632**	**3607**	3642	3684	3655	3612	**3583**	3629	3653	3655	3585

continues on next page

Table 5.6 continued

Instance	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
76	2444	2570	2638	**2264**	2266	2362	2519	2585	2252	**2221**	2250	2481	2522	2252	**2184**
77	1799	1864	1904	1792	**1777**	1770	1846	1879	1792	**1764**	1751	1829	1837	1792	**1750**
78	1671	1694	1748	1669	**1659**	**1650**	1685	1716	1669	**1650**	**1634**	1671	1686	1669	**1634**
79	1750	1799	1841	1751	**1738**	1736	1786	1803	1751	**1733**	1726	1759	1777	1751	**1722**
80	1788	1891	1893	1739	**1732**	1745	1832	1864	1739	**1723**	1711	1804	1819	1739	**1697**
81	3253	3260	3375	3145	**3140**	3171	3189	3354	3145	**3122**	3140	3171	3327	3145	**3086**
82	4691	4742	4784	4693	**4683**	4665	4706	4757	4693	**4659**	4635	4687	4732	4693	**4634**
83	3122	3175	3192	3125	**3091**	3088	3160	3174	3125	**3072**	3062	3136	3144	3125	**3050**
84	2020	2056	2121	1960	**1951**	1961	2026	2076	1960	**1942**	1940	2008	2042	1960	**1931**
85	2400	3132	3196	2379	**2367**	2369	2919	2972	2379	**2344**	2344	2819	2756	2379	**2332**
86	2330	2786	2966	2296	**2267**	2282	2507	2760	2296	**2248**	2246	2425	2521	2296	**2237**
87	3230	3315	3455	2962	**2938**	3137	3243	3390	2962	**2909**	3055	3203	3311	2962	**2876**
88	5401	5481	5523	5405	**5382**	5358	5399	5490	5405	**5351**	5316	5372	5455	5405	**5315**
89	3863	3943	3942	3760	**3737**	3760	3858	3893	3760	**3720**	3716	3844	3864	3760	**3681**
90	3350	3438	3491	3328	**3310**	3344	3389	3476	3328	**3280**	3308	3380	3439	3328	**3257**
91	2456	2598	2637	2418	**2401**	2421	2523	2585	2418	**2376**	2380	2506	2550	2418	**2357**
92	3579	3659	3724	**3212**	3248	3540	3622	3662	**3167**	3212	3423	3560	3610	**3167**	3205
93	2194	2260	2372	2144	**2127**	2155	2213	2311	2144	**2116**	2137	2201	2263	2144	**2111**
94	4458	4543	4616	4430	**4407**	4422	4503	4581	4430	**4393**	4378	4471	4557	4430	**4370**
95	2647	2763	2827	2607	**2580**	2601	2675	2767	2607	**2570**	2571	2648	2746	2607	**2548**
96	3437	3588	3622	**3112**	3120	3384	3533	3600	3112	**3087**	3296	3505	3559	3112	**3046**
97	2538	2837	3204	2547	**2518**	2517	2708	2981	2547	**2506**	2491	2646	2763	2547	**2487**
98	5566	5637	5692	5534	**5511**	5506	5601	5664	5534	**5484**	5458	5562	5632	5534	**5455**
99	2146	2236	2227	**1972**	1989	2105	2192	2196	**1972**	1983	2065	2169	2179	1972	**1969**
100	3340	3399	3426	3306	**3294**	3310	3356	3397	3306	**3291**	3270	3343	3384	3306	**3265**
Mean	2151.2	2207.4	2246.6	2144.4	**2111.2**	2115.4	2173.4	2205.2	2143.7	**2090.4**	2086.3	2151.1	2166.5	2143.6	**2069.5**

Table 5.7: Average makespan of applying the metaheuristics to the first-half of the instances in the LOPS set.

Instance	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
1	**519**	528.5	520.5	528.5	521	**517**	524.2	518.8	526.5	518.9	**515.8**	523.8	518.5	524.8	518.1
2	650	657.2	650.8	661.5	**647.6**	647.5	653	648	659.5	**644**	644.2	651.5	644.2	659.2	**641.4**
3	**621**	625.8	623.5	647.5	621.2	618	622.8	621.5	645.5	**617.6**	616.5	620.8	618.5	642.8	**615.4**
4	**743.8**	752.2	745.5	771.2	745.6	**738**	745	741	769.2	741.6	**737.5**	743	739.8	768.5	739.2
5	**827.8**	839.5	833.5	861.5	828.2	**824.5**	832	827.8	857.2	824.6	**821.2**	828.5	822.8	856.5	822
6	692.8	702.8	695.5	728.8	**690.5**	683	690	685	726.5	**681.4**	**676.8**	685.8	679.5	724.8	677.6
7	**897.2**	910.8	902	942	898.4	893.2	899.8	895	940.5	**892.4**	889.5	896	892.8	939.2	**888.8**
8	**1012**	1020.2	1019	1052	1013.5	**1004.8**	1008.5	1006.5	1052	1005	**999.8**	1005.2	1003.2	1052	1000.8
9	922	935	925.2	977.8	**921.8**	911.2	925.2	912.2	977.8	**910.2**	905.2	912.5	907.8	977.8	**904.9**
10	**766.8**	785.5	773.8	831.5	768.6	**751.8**	768.2	761	831.5	754.9	**746.8**	760.5	750.5	831.2	748.6
11	1188.2	1201.8	1190.2	1227.5	**1179.1**	1174.5	1185.5	1171.8	1227.5	**1166.4**	1163.2	1174.8	1162.5	1227.5	**1158.4**
12	1172	1191.8	1184	1229	**1170.9**	1156.8	1177.5	**1153.8**	1228.2	1154.4	1146.8	1164.8	**1141.5**	1228.2	1145
13	997.5	1006.5	1004.8	1064.5	**997**	**976**	994	983	1064.5	980.9	**967**	983	970.8	1064.5	968.4
14	**1446.5**	1458.8	1447.5	1504.5	1447.4	1433.5	1449.2	1433.8	1504.5	**1432.6**	1426.2	1436.2	**1423.8**	1504.5	1424.8
15	1394.8	1407.8	1394.8	1468.2	**1386.9**	1370.5	1390.2	1369	1466.2	**1367.6**	1358.5	1376	1359.5	1465.2	**1355.2**
16	1317.5	1331.5	1320.5	1371	**1315.8**	**1296.8**	1314	1301.2	1371	1297	**1286.8**	1305	**1286.8**	1371	1287.4
17	**1047**	1067.8	**1047**	1087.5	1049.5	**1032.8**	1047.2	1034.5	1087.5	1033.8	1024	1032.5	**1020**	1087.5	1024.2
18	1889.2	1901	**1886**	1966.2	1892.4	1866.2	1878.8	**1863**	1966.2	1863.4	1855.2	1863.2	**1844.8**	1966.2	1850.4
19	995.2	1010.2	1000.8	1028.2	**994.2**	981.5	995.8	984.5	1028.2	**980.6**	970.8	988	970.8	1028.2	**970.6**
20	**977.5**	990.5	977.8	1019.5	977.6	955.8	980.5	**954.8**	1019.5	956.8	941.8	964.8	**935.2**	1019.5	939.6
21	1889.2	1911.8	1894.5	1957	**1888.8**	1859	1874.5	1860.2	1957	**1858.6**	1845.8	1859.8	**1841.5**	1957	1841.8
22	1421.5	1435.5	1469.2	1487.2	**1408**	1385.2	1410.2	1414	1487.2	**1375.6**	1364	1393	1392.8	1487.2	**1357.9**
23	**1076.8**	1090	1087	1109.5	1081.8	**1055.5**	1076.8	1067	1109.5	1062.1	**1043**	1061.5	1046.2	1109.5	1048.5
24	1923.8	1946	1938.5	1978	**1922.9**	1889.5	1918	1906	1978	**1886.4**	1870.8	1902	1879.2	1978	**1866.2**
25	1232.5	1254.2	1244.8	1281	**1228.8**	1208.5	1229.8	1215.5	1281	**1208**	**1193.5**	1212.5	1199.5	1281	1194.1

continues on next page

Table 5.7 continued

Instance	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
26	**1286.8**	1310.2	1307.8	1326.2	1287.6	**1263.8**	1292.5	1271.2	1326.2	1270.2	**1242**	1279.2	1244	1326.2	1249.2
27	**1704.5**	1727.8	1732.8	1761.8	1705.1	1676	1707.5	1690.8	1761.8	**1675.6**	**1653**	1686.2	1663.8	1761.8	1654.9
28	1935.2	1956.2	1993	1999.2	**1931.9**	1898.2	1931	1931.5	1999.2	**1884.6**	1872.5	1899.2	1894	1999.2	**1861.8**
29	2019.5	2075.8	2113.2	2108.2	**1999.8**	1968.8	2032.5	2029.5	2108.2	**1953.2**	1930.2	1989.5	1983	2108.2	**1918.4**
30	1565	1573.2	1588.5	1595.8	**1557.5**	1529.2	1551.2	1550.5	1595.8	**1521.5**	1503.5	1534	1516	1595.8	**1496.8**
31	1170	1207	1231.8	1182.8	**1141.5**	1131	1176	1178.2	1182.8	**1111.4**	1105.5	1151.5	1137	1182.8	**1092.2**
32	1074.2	1084	1102.8	1091	**1061.4**	1054.2	1067.5	1066.5	1091	**1046.2**	1039.2	1056.8	1044.5	1091	**1034**
33	2100.2	2123	2161.2	2164	**2099.1**	2068.8	2101.5	2096	2164	**2064.2**	2036.2	2087	2062.5	2164	**2034.5**
34	1454.5	1451.5	1493.8	1443.5	**1404.4**	1406.5	1427	1435.5	1443.5	**1368.9**	1369.2	1399	1401	1443.5	**1345.2**
35	2818.8	2867	2922	2907.2	**2816**	**2744.8**	2817.5	2827.8	2907.2	2748.5	**2700.5**	2786.8	2752.8	2907.2	2705.8
36	2494.8	2522.2	2604.8	2548.8	**2483.1**	2456.5	2486.8	2524.2	2548.8	**2451**	2425	2468.8	2467	2548.8	**2420.1**
37	1281.8	1304.5	1316.2	1290.8	**1274.4**	1263.2	1287.5	1290.2	1290.8	**1256.1**	1241.2	1270.2	1267	1290.8	**1241.1**
38	1173.5	1177.8	1192.8	1173.8	**1150.4**	1151.8	1156	1159.8	1173.8	**1137.1**	1131.5	1143.8	1137.2	1173.8	**1122.8**
39	1768.5	1794.2	1800	1767	**1739.1**	1734.8	1762	1764	1767	**1716**	1706	1738.8	1736.8	1767	**1693.4**
40	2217.2	2235.2	2289.5	2236.2	**2189.6**	2180.8	2210.5	2234.2	2236.2	**2162.9**	2148.8	2192.8	2184.2	2236.2	**2135.9**
41	2332	2386.5	2355.8	2323	**2278.5**	2278.5	2342.5	2294.8	2323	**2247.8**	2243	2296.8	2261.2	2323	**2209.9**
42	1601.2	1618	1668.2	1570.2	**1542.9**	1559	1585.2	1602.2	1570.2	**1527.6**	1530	1561.8	1558.2	1570.2	**1506.1**
43	2530.8	2567.5	2601.8	2552	**2522.4**	2498.8	2545.5	2557.8	2552	**2491.1**	2460.2	2529.2	2515.2	2552	**2458.6**
44	3748	3835.2	3886.8	3716	**3673.6**	3680.8	3774.5	3812.5	3716	**3614.1**	3629.5	3746.8	3744.8	3716	**3545**
45	2082.5	2116	2151.8	2077	**2054.1**	2053.2	2086.8	2100.5	2077	**2038**	2028	2065.8	2059.2	2077	**2020**
46	2190.2	2215	2258	2225.5	**2186**	2160.2	2192	2198.8	2225.5	**2155.6**	2133.8	2168.8	2154.8	2225.5	**2128.6**
47	3434.5	3753.5	3822.2	3461.8	**3389.8**	3369	3671.5	3720.2	3461.8	**3336.8**	3313	3586.2	3618.5	3461.8	**3285.6**
48	1280.8	1293.8	1342.5	1263.2	**1247.9**	1259	1285	1296	1263.2	**1236.1**	1236	1264	1264.8	1263.2	**1222.9**
49	2885.5	2927	2947.2	2896.8	**2864.5**	2847.8	2898.5	2895.2	2896.8	**2838.6**	2822.8	2866.2	2853.5	2896.8	**2811.4**
50	1378.5	1389.8	1417.5	1377.8	**1353.8**	1354.2	1373.5	1394.8	1377.8	**1339**	1335	1360	1366.8	1377.8	**1326.1**

continues on next page

Table 5.7 continued

Instance	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
51	1585.8	1611.5	1668.5	1598.2	**1564.5**	1561	1591.2	1621.8	1598.2	**1546.8**	1539.5	1571	1584.5	1598.2	**1530.9**
52	1979	2060.2	2159	1964	**1920.4**	1933.8	2007.5	2078	1964	**1895.2**	1896.5	1971	2001.8	1964	**1873.9**
53	3637.5	3645.8	3683.5	3672	**3628.6**	3598.5	3618	3645	3672	**3592.4**	3563.2	3590	3601	3672	**3556.4**
54	1566	1578.8	1606	1555.5	**1538.4**	1546.2	1561.2	1575	1555.5	**1529.9**	1530.2	1547.2	1553.5	1555.5	**1516.6**
55	2009	2058	2096	2013	**1993.1**	1982.8	2022.8	2048.8	2013	**1972.4**	1962	1997.5	1999.5	2013	**1951.2**
56	2597	2610.8	2647.8	2586.8	**2565.5**	2563.5	2589	2612	2586.8	**2542.6**	2535.8	2569.2	2574.2	2586.8	**2515.9**
57	1955.2	1957.2	2055	1906.5	**1884.6**	1928.2	1948.5	1999	1906.5	**1872.8**	1883.5	1920.5	1925.5	1906.5	**1859**
58	1822.2	1847.5	1897	1812.2	**1786.8**	1799	1830.2	1858.5	1812.2	**1773.5**	1780.8	1811	1814.5	1812.2	**1754.1**
59	3429	3454.8	3494.2	3442.2	**3416.9**	3400.2	3430	3459	3442.2	**3393.4**	3365.2	3408	3421.8	3442.2	**3363.4**
60	1998.8	2050.2	2076	2006.5	**1987.9**	1976.2	2024	2039.8	2006.5	**1970.9**	**1953.2**	2005	2007.2	2006.5	**1953.2**
61	2001	2070.5	2139.2	2025.5	**1989.8**	**1962.8**	2046.5	2081.8	2025.5	1963	**1932.2**	2014.5	2035.2	2025.5	1937.9
62	3280.5	3350.8	3366.2	3158	**3121.2**	3187.8	3273.5	3311.8	3158	**3111.5**	3116.5	3232.5	3246.8	3158	**3104.6**
63	2538.5	2648.2	2742.8	2540.8	**2506.1**	2465.5	2605.5	2686.2	2540.8	**2459.6**	2424.8	2569	2596	2540.8	**2422.6**
64	1883.2	1952.5	2005.5	1914.8	**1879.6**	1863	1925	1981	1914.8	**1860.5**	1845.8	1903.8	1937	1914.8	**1845.1**
65	1945.2	1992.5	2100.2	1939.5	**1906**	1892.2	1976	2049.2	1939.5	**1884.2**	1866	1957	2003	1939.5	**1862.6**
66	1791.5	1850.5	1927.8	1767.2	**1752.4**	1755.8	1827.5	1891	1767.2	**1749**	1746.5	1808	1846	1767.2	**1743.8**
67	1698	1766.2	1794.2	1685.2	**1677.2**	1678.5	1728.5	1765.2	1685.2	**1670.2**	1662.5	1704	1727.8	1685.2	**1658.4**
68	3386.5	3444.8	3488.8	3082	**3073.4**	3234	3358	3440.2	3080	**2985.2**	3052.8	3324	3364.5	3080	**2930.1**
69	3087.5	3443.5	3522	2945.2	**2904**	2924	3309.5	3388.8	2945.2	**2871.9**	2859	3239.2	3261.8	2945.2	**2835.9**
70	3246.5	3342.2	3544	3178	**3141.5**	3158.5	3308.5	3474.8	3178	**3078.6**	3069.5	3274	3395.2	3178	**2989.8**
71	2211.5	2270.2	2315	2196.5	**2181.6**	2179.2	2251	2284.2	2196.5	**2166.4**	2151.2	2229.2	2245	2196.5	**2147.5**
72	1979.5	2070.5	2094.8	1815.5	**1801.1**	1891.8	2023	2039.5	1815.5	**1793**	1811.8	1987.8	1970.2	1815.5	**1774.5**
73	3982	4048.5	4114.8	3951.5	**3925.2**	3929.5	4006.2	4078.8	3951.5	**3911.1**	3890.2	3983	4033.8	3951.5	**3879.8**
74	4398	4416.2	4418.8	4286.2	**4265.5**	4291.5	4348.5	4373.2	4286.2	**4232.4**	4229.8	4306.2	4329.8	4286.2	**4176.5**
75	3639.5	3688.5	3717.5	3662.2	**3635.5**	**3610.8**	3663.5	3684.8	3662.2	3615.1	**3588.2**	3642.5	3657.5	3662.2	3589

continues on next page

Table 5.7 continued

Instance	CPU time limit: 5 minutes					CPU time limit: 30 minutes					CPU time limit: 2 hours				
	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE	DE	GA	ILS	TS	TS+DE
76	2514.2	2619.2	2664	**2267**	2343.6	2391	2560	2597.5	**2261.2**	2280	2300.2	2518.5	2535.8	2261.2	**2221.2**
77	1806.2	1889.2	1919.2	1798.2	**1781.2**	1774.8	1854.2	1887.8	1798.2	**1770.9**	1760.2	1834.2	1843.8	1798.2	**1760**
78	1681.8	1724.5	1773.2	1676.2	**1664**	**1653.8**	1699.2	1734	1676.2	1654.8	**1636**	1685.5	1695.8	1676.2	1636.9
79	1759.8	1829	1844.5	1756.5	**1745.6**	1743.2	1793.8	1811.8	1756.5	**1738.6**	1727.5	1770.5	1781	1756.5	**1726.2**
80	1801.2	1900	1907.5	1746.2	**1741.8**	1760.2	1842	1875.2	1746.2	**1731.6**	1731.8	1821	1827.8	1746.2	**1705.6**
81	3260	3309.5	3387.8	3149.5	**3144.2**	3206	3243	3363	3149.5	**3136.8**	3150.8	3216.8	3330.2	3149.5	**3107.5**
82	4712.5	4761.8	4797.8	4705	**4689.5**	4682.2	4724.2	4773	4705	**4666.6**	4646.5	4703.2	4749.5	4705	**4638.6**
83	3141.8	3200.8	3224.5	3134.8	**3102.4**	3097.2	3169	3198.5	3134.8	**3081.4**	3063.2	3151.8	3170.2	3134.8	**3056.9**
84	2038	2103	2163	1961.2	**1960.6**	1974.5	2067.2	2108.2	1961.2	**1951.9**	1957.8	2041.5	2061.2	1961.2	**1937.6**
85	2473.8	3222	3236.2	2395.5	**2375.5**	2398.2	3010	3027.2	2395.5	**2357.5**	2359.2	2885	2791.8	2395.5	**2340.2**
86	2345.8	2875.8	3005.5	2310.8	**2282.8**	2303	2629	2774	2310.8	**2265.2**	2258	2530.5	2565.8	2310.8	**2242.2**
87	3253.8	3375.5	3535	**2975.2**	2990.2	3157.8	3292.5	3424.5	2975.2	**2949.8**	3097.8	3246.8	3330	2975.2	**2908.5**
88	5413	5495.2	5534.5	5415.5	**5395.2**	5369.8	5432.2	5504	5415.5	**5362.2**	5324.5	5400.2	5464.5	5415.5	**5323**
89	3884	3998.5	4004.8	3765.8	**3748**	3818	3917	3951.2	3765.8	**3729.5**	3762.8	3869.5	3906.5	3765.8	**3693.4**
90	3369.5	3504	3514.5	3334.2	**3330.1**	3345	3434.2	3490.8	3334.2	**3310.4**	3329.2	3411.8	3455.8	3334.2	**3288.5**
91	2501.8	2622	2823	2428.2	**2408.8**	2431.2	2559.2	2656.2	2428.2	**2388.5**	2400.8	2526	2590	2428.2	**2362**
92	3609	3743.2	3776	**3224.5**	3274.8	3559.5	3654	3719.5	**3199.8**	3237.4	3499.5	3603.2	3642.5	**3197**	3210.1
93	2202.2	2320.5	2422.8	2148	**2145.8**	2168.5	2261.8	2325.5	2148	**2138.9**	2148.5	2227.2	2269.8	2148	**2133**
94	4474.5	4579.2	4639.5	4441	**4427.2**	4433.2	4525	4597.8	4441	**4409.8**	4389.8	4494.8	4568.8	4441	**4380**
95	2702.5	2834.8	2872.5	2609.8	**2594.1**	2616.2	2748.2	2806.8	2609.8	**2578.9**	2581.8	2713	2762.2	2609.8	**2558.1**
96	3487.5	3658.5	3683.8	**3123.2**	3192.8	3418.2	3572	3628.5	**3123.2**	3153.1	3336.2	3529.8	3588.5	3123.2	**3108.4**
97	2560.8	3185.2	3427.5	2556.8	**2532.2**	2527.8	2776	3197	2556.8	**2511.4**	2496.2	2682	2813	2556.8	**2490.5**
98	5589	5698.8	5714.5	5548	**5516.1**	5524.8	5631	5682	5548	**5496.2**	5477.5	5584.8	5646.5	5548	**5461**
99	2157.2	2265.2	2322.5	**1986.5**	2013.8	2114.2	2204	2246	**1986.5**	2003.4	2071.8	2177	2194.2	1986.5	**1985.8**
100	3349	3434.5	3494.5	3313.5	**3309**	3317.8	3403.5	3454.2	3313.5	**3295.6**	3281.8	3388.5	3431.5	3313.5	**3271.6**
Mean	2168.9	2238.6	2274.5	2153.2	**2123.1**	2129.2	2196.3	2225.5	2152.7	**2100.5**	2098.9	2170.6	2180.9	2152.5	**2078.3**

Table 5.8: Results of applying CPO to the instances in the LOPS set.

Inst.	5m	30m	2h		Inst.	5m	30m	2h		Inst.	5m	30m	2h		Inst.	5m	30m	2h	
	UB	UB	LB	UB		UB	UB	LB	UB		UB	UB	LB	UB		UB	UB	LB	UB
1	538	530	387	527	26	1581	1486	880	1362	51	2130	1991	521	1734	76	—	—	639	2637
2	663	654	494	650	27	1833	1791	1246	1790	52	2540	2236	553	2091	77	—	—	604	2010
3	653	635	452	633	28	2185	2171	1396	2089	53	4711	3836	523	3860	78	—	—	636	1960
4	780	755	562	756	29	2256	2298	1452	2199	54	3074	2086	533	1661	79	—	—	560	2379
5	860	837	625	828	30	2473	1590	1116	1769	55	2416	2386	497	2295	80	—	—	654	2392
6	724	718	490	715	31	1296	1295	822	1218	56	2876	2939	510	2706	81	—	—	693	3377
7	964	938	659	923	32	1214	1208	776	1151	57	2413	2683	584	2169	82	—	5339	624	5276
8	1091	1044	739	1039	33	2698	2398	1469	2276	58	2520	2045	823	1993	83	—	3346	701	3355
9	1019	966	654	980	34	1545	1604	951	1483	59	3800	3998	511	3764	84	—	—	672	2231
10	902	900	547	790	35	3145	3099	1932	3049	60	2375	2236	587	2577	85	—	—	818	3786
11	1290	1231	857	1230	36	3167	2819	1788	2826	61	—	—	582	2845	86	—	—	918	3065
12	1257	1223	838	1180	37	1463	1671	924	1468	62	3845	3352	658	3341	87	—	—	696	3388
13	1084	1010	699	1011	38	1264	1200	832	1192	63	3169	2934	1392	2782	88	—	5963	671	5956
14	1557	1514	1052	1486	39	1880	1870	1214	1845	64	2960	2123	914	2104	89	—	—	1374	4068
15	1542	1476	975	1445	40	2463	2509	1552	2439	65	—	—	584	2061	90	—	—	646	3616
16	1516	1420	924	1382	41	2550	2522	1587	2524	66	—	—	611	1924	91	—	—	728	3449
17	1131	1080	763	1117	42	1732	1688	1111	1715	67	—	—	804	1930	92	—	—	712	3769
18	2014	1918	1415	1897	43	2784	2779	1737	2767	68	3858	3343	642	3238	93	—	—	751	2392
19	1236	1046	737	1028	44	4030	4063	2587	3908	69	3787	3346	615	3518	94	—	—	780	4883
20	1135	1132	671	1053	45	2378	2466	1446	2301	70	4037	4340	629	4172	95	—	—	726	3051
21	2104	1992	1378	1956	46	2439	2497	1539	2446	71	3771	2417	598	2435	96	—	—	1555	3662
22	1639	1642	985	1496	47	4195	3757	2518	4040	72	—	—	943	1949	97	—	—	1650	3976
23	1336	1128	762	1146	48	1377	1506	896	1473	73	4946	4304	624	4376	98	—	—	716	6117
24	2135	2073	1377	2010	49	4065	3012	2108	3233	74	5062	4668	713	4602	99	—	—	690	2825
25	1524	1442	892	1367	50	2244	1570	931	1505	75	4185	4177	711	4046	100	—	—	682	3717
															Mean	2263.4	2195.4		2402.2

5.4.4 Experiment With the FJS Scheduling Problem and the FJS Scheduling Problem With Sequencing Flexibility

In this section, with the aim of assessing the performance of the TS+DE method with respect to the state-of-the-art in the literature, numerical experiments with classical instances of the FJS and FJS with sequencing flexibility scheduling problems are conducted. Instances, whose main characteristics are shown in Table 5.2, correspond to the instances introduced in (Brandimarte, 1993; Hurink, Jurisch, and Thole, 1994; Barnes and Chambers, 1996; Dauzère-Pérès and Paulli, 1997; Birgin et al., 2014). TS+DE was run 50 times on the instances in sets YFJS, DAFJS, BR, BC, and DP; and 12 times in the instances in set HK. A CPU time limit of 2 hours was imposed. The performances of TS+DE and its competitors are reported in these experiments through the Relative Error (RE) of the best makespan $mks(M,p)$ that method "M" found when applied to instance p, with respect to a known LB $mks_{\text{LB}}(p)$, given by

$$RE(M,p) = 100\% \times \frac{mks(M,p) - mks_{\text{LB}}(p)}{mks_{\text{LB}}(p)}.$$

LBs for instances p in the sets BR, BC, DP, and HK were taken from (Mastrolilli and Gambardella, 1999) and (IBM ILOG CP Optimizer Developers, 2020). LBs for instances p in the sets YFJS and DAFJS were computed running CPO with a CPU time limit of 2 hours. TS+DE was compared with seven different methods from the literature that reported results in at least one of the considered sets, namely: (HA) hybrid GA and TS proposed in (Li and Gao, 2016); (HDE) hybrid DE with LS proposed in (Yuan and Xu, 2013); (HGTS) hybrid GA and TS proposed in (Palacios et al., 2015); (HGVNA) hybrid GA and VNS proposed in (Gao, Sun, and Gen, 2008); (BS) Beam Search (BS) algorithm introduced in (Birgin, Ferreira, and Ronconi, 2015); (KCSA) Knowledge-based Cuckoo Search (CS) Algorithm proposed in (Cao, Lin, and Zhou, to appear); and (ICATS) hybrid ICA and TS introduced in (Lunardi, Voos, and Cherri, 2019). The used LBs and the best solutions obtained by TS+DE and its ten competitors were gathered in tables and can be found in the appendix or in the supplementary material to this work (Lunardi et al., 2020) (accessible through `https://orbilu.uni.lu/handle/10993/43275`). Tables 5.9 and 5.10 show the results. In the tables, it is reported for each method M and each instances' set $\mathcal{S}$

$$\frac{1}{|\mathcal{S}|}\sum_{p\in\mathcal{S}} RE(M,p).$$

Besides, the tables also report how often each method found the best solution (among the solutions found by all the methods). The numerical values in both tables show that TS+DE, although proposed/developed to address the OPS scheduling problem,

achieves the best performance in all sets. It is worth noticing that the goal of this comparison is to analyse the effectiveness of the proposed approach. Efficiency is being neglected in the comparison, since methods being compared were run under different environments and with different stopping criteria.

Table 5.9: Comparison of TS+DE against other methods from the literature on classical instances of the FJS scheduling problem.

Set	HA		HDE		HGTS		HGVNA		TS+DE	
	RE	#best	RE	#best	RE	#best	RE	#best	RE	#best
BR	1.454	10	1.509	9	1.509	9	1.630	9	1.454	10
BC	0.052	15	0.054	14	0.056	14	0.247	9	0.000	21
DP	1.723	6	—	—	1.631	5	2.023	0	1.494	13
HK (E)	0.230	34	—	—	—	—	—	—	0.093	43
HK (R)	0.621	27	—	—	—	—	—	—	0.542	42
HK (V)	0.073	34	—	—	—	—	—	—	0.022	43

Table 5.10: Comparison of TS+DE against other methods from the literature on instances of the FJS scheduling problem with sequencing flexibility.

Set	CPO		BS		KCSA		ICATS		TS+DE	
	RE	#best	RE	#best	RE	#best	RE	#best		
YFJS	0.000	20	12.300	0	16.939	0	0.107	18	0.000	20
DAFJS	30.348	12	38.997	2	49.681	1	34.047	5	29.378	30

5.5 Summary

In this chapter, heuristic methods for the OPS scheduling problem were proposed. All proposed methods rely on a common representation scheme and a neighborhood adapted from the classical LS introduced in (Mastrolilli and Gambardella, 2000) for the FJS scheduling problem. While considering the sequencing flexibility in the LS is somehow immediate, this is definitely not the case of considering fixed operations, machines' downtimes, and resumable operations. Two population-based metaheuristics and two single-solution metaheuristics were considered and, at the end, a combined approach was the one that presented the best performance. The resulting method outperformed results obtained with CP Optimizer. When applied to instances of the FJS scheduling problem and instances of the FJS scheduling problem with sequencing flexibility, the introduced approach exhibited a competitive performance.

Chapter 6

Findings, Conclusions, and Future Outlook

The motivation and objectives of this study are outlined in Chapter 1. Several research questions to obtain the objectives of this study are answered throughout Chapters 2–5. This chapter is structured as follows. The main findings are summarised in Section 6.1. Conclusions are drawn in Section 6.2. Potential areas for future research outlook are discussed in Section 6.3.

6.1 Findings

The motivation of this study is to improve the production effectiveness of OPSs by planning a production schedule. The main objectives are to define the OPS scheduling problem and develop exact and heuristic optimization methods to minimize the amount of time required to manufacture a set of client's orders. The optimization methods must be suitable to be used in practice, in the sense that they must be able to fit with the requirements of the OPSs by being capable of effectively dealing with a large number of operations to be scheduled on several flexible machines. The main findings in this study are structured around the contributions that answer research questions mentioned in Chapter 1.

Several meetings with the industrial partner and the review of the literature study answer the question of "*through collaboration with an industrial partner, how to define and formulate a realistic yet tractable model of the OPS manufacturing system?*". Collaboration with the industrial partner was essential to identify the OPS manufacturing system's critical functionalities that needed to be considered in the scheduling problem. Concerning the review of related works, it turns out that most of the related works address the distinct features of the OPS scheduling problem in an isolated manner. Even though this helps in the understanding of each feature individually, the

applicability of their methods for the OPS scheduling problem is not possible due to the absence of the other constraints. It worth stating that the combination of the distinct constraints of the OPS scheduling problem generates highly complex constraints. Besides, the review of the works of literature related to optimization techniques for the FJS scheduling problem answers the question of "*how to solve the problem given the fact that the model presents several complex constraints and that size of instances in practice is substantial?*". It turns out that the state-of-the-art optimization methods for the FJS scheduling problem capable of solving large-sized instances are hybrid methods combining local search heuristic and metaheuristics. Although few, most of the works in which a practical scheduling problem related to the OPS scheduling problem are tackled, hybrid methods are commonly proposed to solve the problem.

To answer the question "*how to precisely formulate the OPS scheduling problem by means of MP and CP?*". An extensive review of the literature considering MILP and CP formulations for the FJS scheduling problem allowed to accurately describe and formulate the OPS scheduling problem with both modeling paradigms. The proposal of the MILP and the CP formulations enable the verification of the correctness of both formulations concerning the real problem. Numerical experiments which assess the capacity of IBM ILOG CPLEX MILP and IBM ILOG CP Optimizer solvers for dealing with the proposed formulations answers the research question "*what is the effectiveness and limitations of the MP and CP formulations with the commercial solvers IBM ILOG CPLEX and IBM ILOG CP Optimizer?*". It turns out that only small-sized instances can be solved with a certificate of optimality when the MILP formulation is tackled with an exact commercial solver. While the MILP solver is a general-purpose solver, the CP Optimizer solver was born as a solver dedicated to scheduling problems, with its own modeling language that fully explores the structure of the underlying problem. Thus, it is not an astonishment the latter to outperform the former to a large extent in the numerical experiments with the OPS scheduling problem. The obtained result is in agreement with a previous comparison of similar nature presented in (Vilím, Laborie, and Shaw, 2015) and (Laborie, 2018) for the JS and the FJS scheduling problems.

Numerical experiments involving the CP formulation and large-sized instances in which characteristics such as the number of machines and operations are similar to the instances that appear in practice answer the research question "*what is the size of the instances for which optimality can be proved, and how good is the incumbent solution when it cannot?*". It turns out that the CP Optimizer solver can deliver feasible solutions for large-sized instances, thus being an alternative to tackle the OPS scheduling problem in practice.

To answer the question "*how to encode and decode OPS scheduling solutions?*" related

works were analyzed, and it turns out that traditional representation schemes used for the FJS scheduling problem can not be directly used for the OPS scheduling problem due to the presence of the sequencing flexibility. An indirect representation scheme composed of two real vectors coupled with a simple algorithm allows the generation of feasible the machine assignment and a feasible operations sequencing for the OPS scheduling problem. Then, a constructive heuristic was proposed to answer the research question "*how to construct feasible OPS scheduling solutions*". Given the machine assignment and the permutation of the non-fixed operations, the constructive heuristic builds a feasible semi-active schedule by adding operation by operation following the topological order given by the permutation. By creating a feasible solution, the makespan can be evaluated.

It has been found that a classical and very efficient definition of the neighborhood for the FJS problem could be used as a base foundation for the definition neighborhood of an OPS scheduling solution. While the definition of the neighborhood by considering the sequencing flexibility is somehow immediate, this is definitely not the case of considering fixed operations, machines' downtimes, and resumable operations. In this way, the proposal of the neighborhood answer the research question "*how to define a set of neighbor solutions of an OPS scheduling solution to allow the applicability of LS heuristics or neighborhood-based methods?*".

To answer the research question, "*which metaheuristic frameworks are the most suitable for the OPS scheduling problem?*" an exhaustive analysis and customization of state-of-the-art methods for the FJS scheduling problem and metaheuristic frameworks have occurred. Achievements accomplished in the course of this work by means of many accepted publications in the area of combinatorial optimization and metaheuristics validate it. Two populational and two trajectory metaheuristics frameworks were considered and adapted to the OPS scheduling problem; in the end, a combined approach was introduced. It turns out that the combined approach achieved the best performance among the considered metaheuristics, outperforming the results obtained with CP Optimizer.

Experiments with classical instances of the FJS scheduling problem and instances of the FJS scheduling problem with sequencing flexibility show that the proposed combined approach presents competitive results when compared to other state-of-the-art methods, answering research question "*how the best performing method behaves when applied to particularizations of the OPS scheduling problem?*".

6.2 Conclusions

The purpose of this dissertation was to present a unique real-world scheduling problem, the OPS scheduling problem, and to investigate optimization techniques to plan a production schedule so that the production effectiveness of OPSs could be improved. The close collaboration with an industrial partner conceded a sharp perception of the current printing industry's scenario, allowing the identification of the critical features of the OPSs' manufacturing system. An investigation of the variations on the classical FJS scheduling problem resulted in the OPS scheduling problem being modeled as a FJS scheduling problem with sequencing flexibility, machine availability, sequence-dependent setup times, overlapping between operations with precedence constraints, release times, and fixed operations.

The problem was presented in a precise and universal language through a MILP formulation. The problem was also formulated using a modern CP-based scheduling language, which allows the usage of a solver dedicated to scheduling problems. Both models are validated and analyzed comparatively. While the MILP solver is a general-purpose solver, the CP Optimizer solver was born as a solver dedicated to scheduling problems, with its own modeling language that fully explores the structure of the underlying problem. Numerical experiments show that the latter to outperform the former to a large extent. The practical applicability of the CP solver was examined, taking into account that the CP solver is somehow exact and that the size of the problem instances in practice is often substantial. Numerical experiments show that the CP Optimizer was able to find a feasible solution to large-sized instances, thus being an alternative to tackle the OPS scheduling problem in practice. Nevertheless, accessing the quality of the solutions found by the solver was not possible due to the lack of ad-hoc heuristics for the problem. Furthermore, it was expected that ad-hoc methods that exploit the structure of the problem would prove to be more effective than the CP solver.

Alternative problem representations were considered, and a novel representation scheme composed of two real vectors was introduced. A decoding procedure was proposed allowing that, given the two real vectors denoting the machine assignment and sequencing of the operations, a feasible OPS scheduling solution can be constructed and its makespan evaluated. A classical FJS scheduling problem neighborhood function was extended to operate with the specificities of the OPS scheduling problem. Consequently, a local search procedure was proposed. Two population-based metaheuristics and two single-solution metaheuristics were considered, and, in the end, a combined approach was the one that presented the best performance. The resulting method outperformed results obtained with CP Optimizer. When applied to instances of the

FJS scheduling problem and instances of the FJS scheduling problem with sequencing flexibility, the introduced approach exhibited a competitive performance.

6.3 Future Outlook

Several exciting areas and directions for future research can be extended and recommended based on the research presented in this thesis. The following research topics can be pursued to bridge current gaps in the literature.

I. Increasing flexibility of the OPS scheduling problem by considering process plan flexibility can provide the potential for improved performance.

II. Transportation times among machines in the shop floor and production facilities can be included in the OPS scheduling problem. Moreover, the scheduling of transportation robots may provide significant performance improvement due to the broad adoption of AGVs in the OPSs.

III. Industry 4.0 and digital twin technologies can implement comprehensive control and monitoring of the manufacturing lifecycle in the plants. Therefore, a dynamic version of the OPS scheduling problem can be examined to consider real-time dynamic events such as job cancellations/arrivals, machine breakdowns, and processing delays.

IV. Verify the performance of alternative representation schemes and neighborhood functions for the OPS scheduling problem.

V. Consider Pareto optimization techniques for the OPS scheduling problem.

VI. Development of machine learning strategies to tune metaheuristics parameters.

Appendix A

Supplementary Material

The used LBs and the best makespan values obtained by TS+DE and its ten competitors were gathered in tables below. LBs for instances in sets BR, BC, DP, and HK were taken from (Mastrolilli and Gambardella, 1999) and (IBM ILOG CP Optimizer Developers, 2020). LBs for instances in sets YFJS and DAFJS were computed running CPO with a CPU time limit of 2 hours. The state-of-the-art methods reported are: hybrid GA and TS proposed in (Li and Gao, 2016), and named here as HGA; hybrid DE and LS proposed in (Yuan and Xu, 2013), and named HDE; hybrid GA and TS proposed in (Palacios et al., 2015), and named here as HGTS; hybrid GA and VNS proposed in (Gao, Sun, and Gen, 2008), and named here as HGVNA; BS algorithm introduced in (Birgin, Ferreira, and Ronconi, 2015); Knowledge-based Cuckoo Search Algorithm (KCSA) proposed in (Cao, Lin, and Zhou, to appear); and hybrid ICA and TS introduced in (Lunardi, Voos, and Cherri, 2019), named here as (ICATS). Additionally, tables also present results related to the GA and GWO proposed in (Lunardi, Voos, and Cherri, 2019), which were not considered in the experiments presented in Section 5.4.4 due to the fact that ICATS (proposed in the same work) outperforms both. In the tables, TS+DE$_b$ denotes the best makespan and TS+DE$_m$ the mean makespan obtained with the proposed combined metaheuristic method, i.e., TS+DE, over the respective set being presented.

Table A.1: The best makespan and the average makespan obtained with the proposed combined metaheuristic method over the Brandimarte (1993) instances.

Instance	LB	UB	HGA	HDE	HGTS	HGVNA	TS+DE$_b$	TS+DE$_m$
Mk01	40	40	40	40	40	40	40	40
Mk02	26	26	26	26	26	26	26	26
Mk03	204	204	204	204	204	204	204	204
Mk04	60	60	60	60	60	60	60	60
Mk05	168	172	172	172	172	172	172	172.12
Mk06	57	57	57	57	57	58	57	57.75
Mk07	133	139	139	139	139	139	139	139
Mk08	523	523	523	523	523	523	523	523
Mk09	307	307	307	307	307	307	307	307
Mk10	183	195	197	198	198	197	197	198.38

Table A.2: The best makespan and the average makespan obtained with the proposed combined metaheuristic method over the Barnes and Chambers (1996) instances.

Instance	LB	UB	HGA	HDE	HGTS	HGVNA	TS+DE$_b$	TS+DE$_m$
mt10c1	927	927	927	927	927	927	927	927
mt10cc	908	908	908	908	908	910	908	908
mt10x	918	918	918	918	918	918	918	918
mt10xx	918	918	918	918	918	918	918	918
mt10xxx	918	918	918	918	918	918	918	918
mt10xy	905	905	905	905	905	905	905	905
mt10xyz	847	847	847	847	847	849	847	847
setb4c9	914	914	914	914	914	914	914	914
setb4cc	907	907	907	907	907	914	907	907
setb4x	925	925	925	925	925	925	925	925
setb4xx	925	925	925	925	925	925	925	925
setb4xxx	925	925	925	925	925	925	925	925
setb4xy	910	910	910	910	910	916	910	910
setb4xyz	902	902	905	903	905	905	902	902
seti5c12	1169	1169	1170	1171	1170	1175	1169	1169
seti5cc	1135	1135	1136	1136	1136	1138	1135	1135
seti5x	1198	1198	1198	1200	1199	1204	1198	1198
seti5xx	1194	1194	1197	1197	1197	1202	1194	1194
seti5xxx	1194	1194	1197	1197	1197	1204	1194	1194
seti5xy	1135	1135	1136	1136	1136	1136	1135	1135
seti5xyz	1125	1125	1125	1125	1125	1126	1125	1125

Table A.3: The best makespan and the average makespan obtained with the proposed combined metaheuristic method over the Dauzère-Pérès and Paulli (1997) instances.

Instance	LB	UB	HGA	HGTS	HGVNA	TS+DE$_b$	TS+DE$_m$
01a	2505	2505	2505	2505	2518	2505	2505
02a	2228	2231	2230	2230	2231	2228	2232.44
03a	2228	2228	2229	2228	2229	2228	2229.25
04a	2503	2503	2503	2503	2515	2506	2519.5
05a	2189	2213	2212	2214	2217	2212	2218.78
06a	2162	2185	2197	2193	2196	2187	2193.12
07a	2206	2277	2279	2270	2307	2276	2301
08a	2061	2064	2067	2070	2073	2071	2074
09a	2061	2061	2065	2067	2066	2063	2064.75
10a	2197	2263	2287	2247	2315	2287	2306.67
11a	2017	2053	2060	2064	2071	2061	2065
12a	1969	2013	2027	2027	2030	2008	2012.5
13a	2161	2257	2248	2250	2257	2245	2254
14a	2161	2163	2167	2170	2167	2166	2167.62
15a	2161	2162	2163	2168	2165	2163	2163.62
16a	2148	2240	2249	2246	2256	2240	2257.44
17a	2088	2129	2140	2142	2140	2132	2134.12
18a	2057	2105	2132	2129	2127	2097	2101.75

Table A.4: The best makespan and the average makespan obtained with the proposed combined metaheuristic method over the Hurink, Jurisch, and Thole (1994) EData instances.

Instance	LB	UB	HGA	TS+DE$_b$	TS+DE$_m$
mt06	47	47	47	47	47
mt10	655	655	655	655	655
mt20	1022	1022	1022	1022	1022
la01	570	570	570	570	570
la02	529	529	529	529	529
la03	477	477	477	477	477
la04	502	502	502	502	502
la05	457	457	457	457	457
la06	799	799	799	799	799
la07	749	749	749	749	749
la08	765	765	765	765	765
la09	853	853	853	853	853
la10	804	804	804	804	804
la11	1071	1071	1071	1071	1071
la12	936	936	936	936	936
la13	1038	1038	1038	1038	1038
la14	1070	1070	1070	1070	1070
la15	1089	1089	1089	1089	1089
la16	717	717	717	717	717
la17	646	646	646	646	646
la18	663	663	663	663	663
la19	617	617	617	617	617
la20	756	756	756	756	756
la21	800	801	804	802	803.12
la22	733	734	738	734	734.75
la23	809	810	813	810	811
la24	773	774	777	774	775.12
la25	751	753	754	752	753.62
la26	1052	1052	1053	1052	1052.5
la27	1084	1084	1085	1084	1084
la28	1069	1069	1070	1069	1069
la29	993	993	994	994	994
la30	1068	1069	1069	1069	1069
la31	1520	1520	1520	1520	1520
la32	1657	1657	1658	1657	1657.75
la33	1497	1497	1497	1497	1497.5
la34	1535	1535	1535	1535	1535
la35	1549	1549	1549	1549	1549
la36	948	948	948	948	948
la37	986	986	986	986	986
la38	943	943	943	943	943
la39	922	922	922	922	922
la40	955	955	955	955	955

Table A.5: The best makespan and the average makespan obtained with the proposed combined metaheuristic method over the Hurink, Jurisch, and Thole (1994) RData instances.

Instance	LB	UB	HGA	TS+DE$_b$	TS+DE$_m$
mt06	47	47	47	47	47
mt10	686	686	686	686	686
mt20	1022	1022	1022	1022	1022
la01	570	570	570	570	570.75
la02	529	529	530	529	529
la03	477	477	477	477	477
la04	502	502	502	502	502
la05	457	457	457	457	457
la06	799	799	799	799	799
la07	749	749	749	749	749
la08	765	765	765	765	765
la09	853	853	853	853	853
la10	804	804	804	804	804
la11	1071	1071	1071	1071	1071
la12	936	936	936	936	936
la13	1038	1038	1038	1038	1038
la14	1070	1070	1070	1070	1070
la15	1089	1089	1090	1089	1089
la16	717	717	717	717	717
la17	646	646	646	646	646
la18	666	666	666	666	666
la19	700	700	700	700	700
la20	756	756	756	756	756
la21	808	833	835	833	836.75
la22	737	757	760	760	764.38
la23	816	832	840	839	842
la24	775	801	806	801	805
la25	752	785	789	785	789.75
la26	1056	1061	1061	1060	1061.75
la27	1085	1090	1089	1090	1090.88
la28	1075	1077	1079	1078	1078.75
la29	993	996	997	996	996.62
la30	1068	1078	1078	1078	1079.12
la31	1520	1520	1521	1520	1520
la32	1657	1658	1659	1658	1658
la33	1497	1498	1499	1498	1498
la34	1535	1535	1536	1535	1535.25
la35	1549	1549	1550	1549	1549.75
la36	1023	1023	1028	1028	1028.5
la37	1062	1062	1074	1067	1073.88
la38	954	954	960	960	963
la39	1011	1011	1024	1024	1024.12
la40	955	955	970	966	971.5

Table A.6: The best makespan and the average makespan obtained with the proposed combined metaheuristic method over the Hurink, Jurisch, and Thole (1994) VData instances.

Instance	LB	UB	HGA	TS+DE$_b$	TS+DE$_m$
mt06	47	47	47	47	47
mt10	655	655	655	655	655
mt20	1022	1022	1022	1022	1022
la01	570	570	570	570	570
la02	529	529	529	529	529
la03	477	477	477	477	477
la04	502	502	502	502	502
la05	457	457	457	457	457
la06	799	799	799	799	799
la07	749	749	749	749	749
la08	765	765	765	765	765
la09	853	853	853	853	853
la10	804	804	804	804	804
la11	1071	1071	1071	1071	1071
la12	936	936	936	936	936
la13	1038	1038	1038	1038	1038
la14	1070	1070	1070	1070	1070
la15	1089	1089	1089	1089	1089
la16	717	717	717	717	717
la17	646	646	646	646	646
la18	663	663	663	663	663
la19	617	617	617	617	617
la20	756	756	756	756	756
la21	800	801	804	802	803.12
la22	733	734	738	734	734.75
la23	809	810	813	810	811
la24	773	774	777	774	775.12
la25	751	753	754	752	753.62
la26	1052	1052	1053	1052	1052.5
la27	1084	1084	1085	1084	1084
la28	1069	1069	1070	1069	1069
la29	993	993	994	994	994
la30	1068	1069	1069	1069	1069
la31	1520	1520	1520	1520	1520
la32	1657	1657	1658	1657	1657.75
la33	1497	1497	1497	1497	1497.5
la34	1535	1535	1535	1535	1535
la35	1549	1549	1549	1549	1549
la36	948	948	948	948	948
la37	986	986	986	986	986
la38	943	943	943	943	943
la39	922	922	922	922	922
la40	955	955	955	955	955

Table A.7: The best makespan and the average makespan obtained with the proposed combined metaheuristic method over the Birgin et al. (2014) YFJS instances.

Instance	CPO		BS	GA	GWO	KCSA	ICA	ICATS	TS+DE$_b$	TS+DE$_m$
	LB	UB								
YFJS01	773	773	825	773	773	792	773	773	773	773
YFJS02	825	825	876	848	843	832	843	825	825	825
YFJS03	347	347	372	356	348	362	347	347	347	347
YFJS04	390	390	458	390	390	401	390	390	390	390
YFJS05	445	445	486	452	452	495	452	445	445	445
YFJS06	446	446	493	450	450	497	447	446	446	446
YFJS07	444	444	487	480	455	792	455	444	444	444
YFJS08	353	353	372	353	353	387	353	353	353	353
YFJS09	242	242	283	242	242	295	242	242	242	242
YFJS10	399	399	418	399	399	415	399	399	399	399
YFJS11	526	526	590	529	529	612	529	526	526	526
YFJS12	512	512	561	540	517	606	517	512	512	512
YFJS13	405	405	455	409	409	488	405	405	405	405
YFJS14	1317	1317	1380	1317	1317	1397	1317	1317	1317	1317
YFJS15	1239	1239	1310	1269	1270	1308	1270	1239	1239	1239
YFJS16	1222	1222	1387	1301	1301	1324	1254	1222	1222	1222
YFJS17	1133	1133	1304	1204	1204	1295	1167	1133	1133	1133
YFJS18	1220	1220	1364	1283	1283	1503	1221	1220	1220	1220
YFJS19	926	926	1256	1080	1153	1350	1080	941	926	926
YFJS20	968	968	1271	1204	1204	1290	1079	973	968	968

Table A.8: The best makespan and the average makespan obtained with the proposed combined metaheuristic method over the Birgin et al. (2014) DAFJS instances.

Instance	CPO		BS	GA	GWO	KCSA	ICA	ICATS	TS+DE$_b$	TS+DE$_m$
	LB	UB								
DAFJS01	257	257	277	257	257	264	257	257	257	257
DAFJS02	289	289	306	289	289	291	289	289	289	289
DAFJS03	576	576	576	576	576	592	576	576	576	576
DAFJS04	606	606	606	606	606	606	606	606	606	606
DAFJS05	384	384	425	421	421	395	424	389	384	384
DAFJS06	404	404	434	414	414	449	423	412	404	404.08
DAFJS07	505	505	542	583	583	566	610	512	505	505
DAFJS08	628	628	632	655	655	631	642	628	628	628
DAFJS09	324	461	482	474	483	490	466	464	460	460.04
DAFJS10	337	522	549	537	537	555	533	533	517	517
DAFJS11	658	658	675	732	732	701	750	659	658	658
DAFJS12	530	600	643	731	731	720	698	645	591	591.21
DAFJS13	306	636	670	655	655	707	653	653	633	633.54
DAFJS14	367	708	755	737	737	818	735	726	708	708
DAFJS15	512	640	705	736	747	818	747	671	631	632.25
DAFJS16	641	644	700	778	780	798	768	679	643	643
DAFJS17	309	777	824	806	812	904	800	787	772	772.29
DAFJS18	328	778	817	790	799	892	790	789	768	768.04
DAFJS19	512	512	545	540	546	585	540	524	512	512
DAFJS20	434	666	711	700	700	810	696	696	662	663.96
DAFJS21	504	771	839	810	810	959	803	803	757	759.04
DAFJS22	464	672	735	722	722	851	697	697	661	663.54
DAFJS23	450	467	490	515	515	537	519	476	460	460.58
DAFJS24	476	543	595	634	635	648	635	564	537	537
DAFJS25	584	699	774	810	810	879	783	752	696	696
DAFJS26	565	697	783	790	806	898	765	745	684	684.96
DAFJS27	503	784	856	876	876	981	842	831	773	773
DAFJS28	535	535	565	620	623	584	594	543	535	535
DAFJS29	609	630	663	744	748	710	725	654	615	618.46
DAFJS30	467	531	572	604	609	637	595	555	523	523.38

Bibliography

Al-Hinai, Nasr and Tarek Y ElMekkawy (2011). "Robust and Stable Flexible Job Shop Scheduling With Random Machine Breakdowns Using a Hybrid Genetic Algorithm". In: *International Journal of Production Economics* 132.2, pp. 279–291. DOI: 10.1016/j.ijpe.2011.04.020.

Ali, M., P. Siarry, and M. Pant (2012). "An Efficient Differential Evolution Based Algorithm for Solving Multi-Objective Optimization Problems". In: *European Journal of Operational Research* 217.2, pp. 404–416. DOI: 10.1016/j.ejor.2011.09.025.

Allahverdi, Ali, Jatinder ND Gupta, and Tariq Aldowaisan (1999). "A Review of Scheduling Research Involving Setup Considerations". In: *Omega* 27.2, pp. 219–239. DOI: 10.1016/S0305-0483(98)00042-5.

Allahverdi, Ali et al. (2008). "A Survey of Scheduling Problems With Setup Times or Costs". In: *European journal of operational research* 187.3, pp. 985–1032. DOI: 10.1016/j.ejor.2006.06.060.

Alvarez-Valdés, Ramón et al. (2005). "A Heuristic to Schedule Flexible Job-Shop in a Glass Factory". In: *European Journal of Operational Research* 165.2, pp. 525–534. DOI: 10.1016/j.ejor.2004.04.020.

Amjad, Muhammad Kamal et al. (2018). "Recent Research Trends in Genetic Algorithm Based Flexible Job Shop Scheduling Problems". In: *Mathematical Problems in Engineering* 2018. DOI: 10.1155/2018/9270802.

Andrade, R. et al. (2014). "MIP Models for Two-Dimensional Non-Guillotine Cutting Problems With Usable Leftovers". In: *Journal of the Operational Research Society* 65, pp. 1649–1663. DOI: 10.1057/jors.2013.108.

Andrade-Pineda, Jose L et al. (2019). "Scheduling a Dual-Resource Flexible Job Shop With Makespan and Due Date-Related Criteria". In: *Annals of Operations Research*, pp. 1–31. DOI: 10.1007/s10479-019-03196-0.

– (to appear). "Scheduling a Dual-Resource Flexible Job Shop With Makespan and Due Date-Related Criteria". In: *Annals of Operations Research*. DOI: 10.1007/s10479-019-03196-0.

Atashpaz-Gargari, Esmaeil and Caro Lucas (2007). "Imperialist Competitive Algorithm: An Algorithm for Optimization Inspired by Imperialistic Competition". In: *Congress on Evolutionary Computation*. IEEE, pp. 4661–4667. DOI: 10.1109/CEC.2007.4425083.

Bäck, Thomas, David B Fogel, and Zbigniew Michalewicz (1997). *Handbook of Evolutionary Computation*. CRC Press. ISBN: 0750303921.

Barnes, J. W. and J. B. Chambers (1996). *Flexible Job Shop Scheduling by Tabu Search*. Tech. rep. ORP96-09. Austin, TX: Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin.

Bertsekas, Dimitri P (1997). "Nonlinear Programming". In: *Journal of the Operational Research Society* 48.3, pp. 334–334. DOI: 10.1057/palgrave.jors.2600425.

Bianchi, Leonora et al. (2009). "A Survey on Metaheuristics for Stochastic Combinatorial Optimization". In: *Natural Computing* 8.2, pp. 239–287. DOI: 10.1007/s11047-008-9098-4.

Birgin, E. G., O. C. Romão, and D. P. Ronconi (2020). "The Multiperiod Two-Dimensional Non-Guillotine Cutting Stock Problem With Usable Leftovers". In: *International Transactions in Operational Research* 27, pp. 1392–1418. DOI: 10.1111/itor.12648.

Birgin, Ernesto G, JE Ferreira, and Débora P Ronconi (2015). "List Scheduling and Beam Search Methods for the Flexible Job Shop Scheduling Problem With Sequencing Flexibility". In: *European Journal of Operational Research* 247.2, pp. 421–440. DOI: 10.1016/j.ejor.2015.06.023.

Birgin, Ernesto G et al. (2014). "A MILP Model for an Extended Version of the Flexible Job Shop Problem". In: *Optimization Letters* 8.4, pp. 1417–1431. DOI: 10.1007/s11590-013-0669-7.

Blackstone, John H, Don T Phillips, and Gary L Hogg (1982). "A State-Of-The-Art Survey of Dispatching Rules for Manufacturing Job Shop Operations". In: *The International Journal of Production Research* 20.1, pp. 27–45. DOI: 10.1080/00207548208947745.

Blazewicz, Jacek et al. (2019). "Scheduling in Flexible Manufacturing Systems". In: *Handbook on Scheduling*. Springer, pp. 671–711. DOI: 10.1007/978-3-540-32220-7_14.

Brandimarte, P. (1993). "Routing and Scheduling in a Flexible Job Shop by Tabu Search". In: *Annals of Operations Research* 41.3, pp. 157–183. DOI: 10.1007/BF02023073.

Brucker, Peter and P Brucker (2007). *Scheduling Algorithms*. Vol. 3. Springer. DOI: 10.1007/978-3-540-69516-5.

Burke, Edmund K and Graham Kendall (2014). *Search Methodologies*. Springer US. DOI: 10.1007/978-1-4614-6940-7.

Çaliş, Banu and Serol Bulkan (2015). "A Research Survey: Review of AI Solution Strategies of Job Shop Scheduling Problem". In: *Journal of Intelligent Manufacturing* 26.5, pp. 961–973. DOI: 10.1007/s10845-013-0837-8.

Calleja, Gema and Rafael Pastor (2014). "A Dispatching Algorithm for Flexible Job-Shop Scheduling With Transfer Batches: An Industrial Application". In: *Production Planning & Control* 25.2, pp. 93–109. DOI: 10.1080/09537287.2013.782846.

Cao, Z., C. Lin, and M. Zhou (to appear). "A Knowledge-Based Cuckoo Search Algorithm to Schedule a Flexible Job Shop With Sequencing Flexibility". In: *IEEE Transcations on Automation Science and Engineering*. DOI: 10.1109/TASE.2019.2945717.

Chaudhry, Imran Ali and Abid Ali Khan (2016). "A Research Survey: Review of Flexible Job Shop Scheduling Techniques". In: *International Transactions in Operational Research* 23.3, pp. 551–591. DOI: 10.1111/itor.12199.

Chu, Paul C and John E Beasley (1998). "A Genetic Algorithm for the Multidimensional Knapsack Problem". In: *Journal of Heuristics* 4.1, pp. 63–86. DOI: 10.1023/A:1009642405419.

Chung, Daeyoung et al. (2005). "A New Approach to Job Shop Scheduling Problems With Due Date Constraints Considering Operation Subcontracts". In: *International Journal of Production Economics* 98.2, pp. 238–250. DOI: 10.1016/j.ijpe.2004.05.023.

Conway, Richard W, Louis W Miller, and William L Maxwell (2003). *Theory of Scheduling*. Dover. ISBN: 0486428176.

Damak, N. et al. (2009). "Differential Evolution for Solving Multi-Mode Resource-Constrained Project Scheduling Problems". In: *Computers & Operations Research* 36.9, pp. 2653–2659. DOI: 10.1016/j.cor.2008.11.010.

Darwin, Charles and William F Bynum (2009). *The Origin of Species by Means of Natural Selection: Or, the Preservation of Favored Races in the Struggle for Life*. Penguin Harmondsworth. ISBN: 1602061440.

Dauzère-Pérès, S. and J. Paulli (1997). "An Integrated Approach for Modeling and Solving the General Multiprocessor Job-Shop Scheduling Problem Using Tabu Search". In: *Annals of Operations Research* 70, pp. 281–306. DOI: 10.1023/A:1018930406487.

Deb, K. and R. B. Agrawal (1995). "Simulated Binary Crossover for Continuous Search Space". In: *Complex Systems* 9.2, pp. 115–148.

Deb, K. and S. Agrawal (1999). "A Niched-Penalty Approach for Constraint Handling in Genetic Algorithms". In: *Artificial Neural Nets and Genetic Algorithms*. Vienna: Springer Vienna, pp. 235–243. DOI: 10.1007/978-3-7091-6384-9_40.

Deb, Kalyanmoy and Debayan Deb (2014). "Analysing Mutation Schemes for Real-Parameter Genetic Algorithms". In: *International Journal of Artificial Intelligence and Soft Computing* 4.1, pp. 1–28. DOI: 10.1504/IJAISC.2014.059280.

Demir, Yunus and S Kürşat İşleyen (2013). "Evaluation of Mathematical Models for Flexible Job-Shop Scheduling Problems". In: *Applied Mathematical Modelling* 37.3, pp. 977–988. DOI: 10.1016/j.apm.2012.03.020.

Demir, Yunus and Selçuk Kürşat İşleyen (2014). "An effective genetic algorithm for flexible job-shop scheduling with overlapping in operations". In: *International Journal of Production Research* 52.13, pp. 3905–3921. DOI: 10.1080/00207543.2014.889328.

Faria Jr, Haroldo de, Willian T. Lunardi, and Holger Voos (2019). "A Parallel Multi-Population Biased Random-Key Genetic Algorithm for Electric Distribution Network Reconfiguration". In: *Genetic and Evolutionary Computation Conference Companion*, pp. 281–282. DOI: 10.1145/3319619.3321950.

Fattahi, Parviz, Fariborz Jolai, and Jamal Arkat (2009). "Flexible Job Shop Scheduling With Overlapping in Operations". In: *Applied Mathematical Modelling* 33.7, pp. 3076–3087. DOI: 10.1016/j.apm.2008.10.029.

Fattahi, Parviz, Mohammad Saidi Mehrabad, and Fariborz Jolai (2007). "Mathematical Modeling and Heuristic Approaches to Flexible Job Shop Scheduling Problems". In: *Journal of Intelligent Manufacturing* 18.3, pp. 331–342. DOI: 10.1007/s10845-007-0026-8.

Ford, Henry and Samuel Crowther (1926). *Today and Tomorrow*. Doubleday Page & Company, New York.

Gan, P. Y. and K. S. Lee (2002). "Scheduling of Flexible-Sequenced Process Plans in a Mould Manufacturing Shop". In: *International Journal of Advanced Manufacturing Technology* 20, pp. 214–222. DOI: 10.1007/s001700200144.

Gansner, Emden R et al. (1993). "A Technique for Drawing Directed Graphs". In: *Transactions on Software Engineering* 19.3, pp. 214–230. DOI: 10.1109/32.221135.

Gao, J., L. Sun, and M. Gen (2008). "A Hybrid Genetic and Variable Neighborhood Descent Algorithm for Flexible Job Shop Scheduling Problems". In: *Computers & Operations Research* 35.9, pp. 2892–2907. DOI: 10.1016/j.cor.2007.01.001.

Gao, Jie, Mitsuo Gen, and Linyan Sun (2006). "Scheduling jobs and maintenances in flexible job shop with a hybrid genetic algorithm". In: *Journal of Intelligent Manufacturing* 17.4, pp. 493–507. DOI: 10.1007/s10845-005-0021-x.

Garey, Michael R, David S Johnson, and Ravi Sethi (1976). "The Complexity of Flowshop and Job Shop Scheduling". In: *Mathematics of Operations Research* 1.2, pp. 117–129. DOI: doi.org/10.1287/moor.1.2.117.

Gholami, Mansour and Mostafa Zandieh (2009). "Integrating Simulation and Genetic Algorithm to Schedule a Dynamic Flexible Job Shop". In: *Journal of Intelligent Manufacturing* 20.4, p. 481. DOI: 10.1007/s10845-008-0150-0.

Glover, Fred (1986). "Future Paths for Integer Programming and Links to Artificial Intelligence". In: *Computers & Operations Research* 13.5, pp. 533–549. DOI: 10.1016/0305-0548(86)90048-1.

Glover, Fred and Manuel Laguna (1998). "Tabu Search". In: pp. 2093–2229. DOI: 10.1007/978-1-4613-0303-9_33.

Graham, Ronald L et al. (1979). "Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey". In: *Annals of Discrete Mathematics*. Vol. 5. Elsevier, pp. 287–326. DOI: 10.1016/S0167-5060(08)70356-X.

Grobler, Jacomine et al. (2009). "Particle Swarm Optimization and Differential Evolution for Multi-Objective Multiple Machine Scheduling". PhD thesis. University of Pretoria. URL: http://hdl.handle.net/2263/25790.

Gu, Jun et al. (1999). *Algorithms for the Satisfiability (SAT) Problem*. Tech. rep., pp. 379–572. DOI: 10.1007/978-1-4757-3023-4_7.

Ham, Andy M and Eray Cakici (2016). "Flexible Job Shop Scheduling Problem With Parallel Batch Processing Machines: Mip and Cp Approaches". In: *Computers & Industrial Engineering* 102, pp. 160–165. DOI: 10.1016/j.cie.2016.11.001.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems — AN Introductory Analysis With Applications to Biology, Control, and Artificial Intelligence*. Cambridge, MA: MIT Press. ISBN: 978-0-262-08213-6.

Hurink, J., B. Jurisch, and M. Thole (1994). "Tabu Search for the Job-Shop Scheduling Problem With Multi-Purpose Machines". In: *Operations-Research-Spektrum* 15.4, pp. 205–215. DOI: 10.1007/BF01719451.

Hussain, Kashif et al. (2019). "Metaheuristic Research: A Comprehensive Survey". In: *Artificial Intelligence Review* 52.4, pp. 2191–2233. DOI: 10.1007/s10462-017-9605-z.

Hutchinson, GK and KA Pflughoeft (1994). "Flexible Process Plans: Their Value in Flexible Automation Systems". In: *The International Journal of Production Research* 32.3, pp. 707–719. DOI: 10.1080/00207549408956962.

IBM ILOG CP Optimizer Developers (2020). *CPO 12.6 Results*. https://www.ibm.com/developerworks/community/blogs/jfp/resource/CPO126-Results.pdf. accessed on April 10, 2020.

IBM ILOG CPLEX Optimization Studio (2020). *CPLEX User's Manual, Version 12 Release 7*. URL: https://www.ibm.com/support/knowledgecenter/SSSA5P_12.7.1/ilog.odms.studio.help/pdf/usrcplex.pdf (visited on 06/08/2020).

Isnaini, Mohammad Mi'radj and Keiichi Shirase (2014). "Review of Computer-Aided Process Planning Systems for Machining Operation–Future Development of a Computer-Aided Process Planning System". In: *International Journal of Automation Technology* 8.3, pp. 317–332. DOI: 10.20965/ijat.2014.p0317.

Jain, Anant Singh and Sheik Meeran (1999). "Deterministic Job-Shop Scheduling: Past, Present and Future". In: *European Journal of Operational Research* 113.2, pp. 390–434. DOI: 10.1016/S0377-2217(98)00113-1.

Jensen, Mikkel T (2003). "Generating Robust and Flexible Job Shop Schedules Using Genetic Algorithms". In: *IEEE Transactions on Evolutionary Computation* 7.3, pp. 275–288. DOI: 10.1109/TEVC.2003.810067.

Jeong, Han-Il, Jinwoo Park, and Robert C Leachman (1999). "A Batch Splitting Method for a Job Shop Scheduling Problem in an Mrp Environment". In: *International Journal of Production Research* 37.15, pp. 3583–3598. DOI: 10.1080/002075499190194.

Jones, Albert, Luis C Rabelo, and Abeer T Sharawi (2001). "Survey of Job Shop Scheduling Techniques". In: *Wiley encyclopedia of electrical and electronics engineering*. DOI: 10.1002/047134608X.W3352.

Kan, AHG Rinnooy (2012). *Machine Scheduling Problems: Classification, Complexity and Computations*. Springer Science & Business Media. DOI: 10.1007/978-1-4613-4383-7.

Karaboga, Dervis and Bahriye Akay (2009). "A Comparative Study of Artificial Bee Colony Algorithm". In: *Applied Mathematics and Computation* 214.1, pp. 108–132. DOI: 10.1016/j.amc.2009.03.090.

Karimi, Sajad et al. (2017). "Scheduling Flexible Job-Shops With Transportation Times: Mathematical Models and a Hybrid Imperialist Competitive Algorithm". In: *Applied mathematical modellíng* 41, pp. 667–682. DOI: 10.1016/j.apm.2016.09.022.

Kennedy, James (2006). "Swarm Intelligence". In: *Handbook of Nature-Inspired and Innovative Computing*. Springer, pp. 187–219. ISBN: 9780080518268.

Kennedy, James and Russell Eberhart (1995). "Particle Swarm Optimization". In: *International Conference on Neural Networks*. Vol. 4. IEEE, pp. 1942–1948. DOI: 10.1109/ICNN.1995.488968.

Khorasanian, Danial and Ghasem Moslehi (2017). "Two-Machine Flow Shop Scheduling Problem With Blocking, Multi-Task Flexibility of the First Machine, and Preemption". In: *Computers & Operations Research* 79, pp. 94–108. DOI: 10.1016/j.cor.2016.09.023.

Kim, Y. K., K. Park, and J. Ko (2003). "A Symbiotic Evolutionary Algorithm for the Integration of Process Planning and Job Shop Scheduling". In: *Computers & Operations Research* 30, pp. 1151–1171. DOI: 10.1016/S0305-0548(02)00063-1.

Krajewski, Lee J et al. (1987). "Kanban, MRP, and shaping the manufacturing environment". In: *Management science* 33.1, pp. 39–57. DOI: 10.1287/mnsc.33.1.39.

Laborie, Philippe (2018). "An Update on the Comparison of MIP, CP and Hybrid Approaches for Mixed Resource Allocation and Scheduling". In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer, pp. 403–411. DOI: 10.1007/978-3-319-93031-2_29.

Laborie, Philippe and Daniel Godard (2007). "Self-Adapting Large Neighborhood Search: Application to Single-Mode Scheduling Problems". In: *Multidisciplinary International Conference on Scheduling: Theory and Applications*. Ed. by P. Baptiste et al. Paris, France, pp. 276–284.

Laborie, Philippe et al. (2018). "IBM ILOG CP Optimizer for scheduling". In: *Constraints* 23.2, pp. 210–250. DOI: 10.1007/s10601-018-9281-x.

Land, Ailsa H and Alison G Doig (2010). "An Automatic Method for Solving Discrete Programming Problems". In: *50 Years of Integer Programming 1958-2008*. Springer, pp. 105–132. DOI: 10.1007/978-3-540-68279-0_5.

Larranaga, Pedro et al. (1999). "Genetic Algorithms for the Travelling Salesman Problem: A Review of Representations and Operators". In: *Artificial Intelligence Review* 13.2, pp. 129–170. DOI: 10.1023/A:1006529012972.

Lawler, Eugene L et al. (1993). "Sequencing and Scheduling: Algorithms and Complexity". In: *Handbooks in Operations Research and Management Science* 4, pp. 445–522. DOI: 10.1016/S0927-0507(05)80189-6.

Lee, Sanghyup et al. (2012a). "Flexible Job-Shop Scheduling Problems With and/or Precedence Constraints". In: *International Journal of Production Research* 50.7, pp. 1979–2001. DOI: 10.1080/00207543.2011.561375.

– (2012b). "Flexible Job-Shop Scheduling Problems With And/Or Precedence Constraints". In: *International Journal of Production Research* 50.7, pp. 1979–2001. DOI: 10.1080/00207543.2011.561375.

Lei, Deming (2012). "Co-Evolutionary Genetic Algorithm for Fuzzy Flexible Job Shop Scheduling". In: *Applied Soft Computing* 12.8, pp. 2237–2245.

Leung, Joseph YT (2004). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC press. DOI: 10.1201/9780203489802.

Leung, Joseph YT et al. (2005). "Open Shops With Jobs Overlap — Revisited". In: *European Journal of Operational Research* 163.2, pp. 569–571. DOI: 10.1016/j.ejor.2003.11.023.

Li, X. and L. Gao (2016). "An Effective Hybrid Genetic Algorithm and Tabu Search for Flexible Job Shop Scheduling Problem". In: *International Journal of Production Economics* 174, pp. 93–110. DOI: 10.1016/j.ijpe.2016.01.016.

Loukil, Taicir, Jacques Teghem, and Philippe Fortemps (2007). "A Multi-Objective Production Scheduling Case Study Solved by Simulated Annealing". In: *European journal of operational research* 179.3, pp. 709–722. DOI: 10.1016/j.ejor.2005.03.073.

Lourenço, Helena R, Olivier C Martin, and Thomas Stützle (2003). "Iterated Local Search". In: *Handbook of Metaheuristics*, pp. 320–353. DOI: 10.1007/0-306-48056-5_11.

Lunardi, W. T. et al. (2020). *Metaheuristics for the Online Printing Shop Scheduling Problem – Supplementary material*. Tech. rep. 10993/43275. University of Luxembourg.

Lunardi, Willian T., Luiz H. Cherri, and Holger Voos (2018). "A Mathematical Model and a Firefly Algorithm for an Extended Flexible Job Shop Problem With Availability Constraints". In: *International Conference on Artificial Intelligence and Soft Computing*. Springer, pp. 548–560. DOI: 10.1007/978-3-319-91253-0_51.

Lunardi, Willian T. and Holger Voos (2018a). "An Extended Flexible Job Shop Scheduling Problem With Parallel Operations". In: *ACM SIGAPP Applied Computing Review* 18.2, pp. 46–56. DOI: 10.1145/3243064.3243068.

– (2018b). "Comparative Study of Genetic and Discrete Firefly Algorithm for Combinatorial Optimization". In: *33rd Annual ACM Symposium on Applied Computing*, pp. 300–308. DOI: 10.1145/3167132.3167160.

Lunardi, Willian T., Holger Voos, and Luiz H. Cherri (2018). "An Imperialist Competitive Algorithm for a Real-World Flexible Job Shop Scheduling Problem". In: *23rd IEEE International Conference on Emerging Technologies and Factory Automation*. Vol. 1. IEEE, pp. 402–409. DOI: 10.1109/ETFA.2018.8502596.

– (2019). "An Effective Hybrid Imperialist Competitive Algorithm and Tabu Search for an Extended Flexible Job Shop Scheduling Problem". In: *34th ACM/SIGAPP Symposium on Applied Computing*, pp. 204–211. DOI: 10.1145/3297280.3297302.

Lunardi, Willian T. et al. (2016). "Automated Decision Support IOT Framework". In: *21st IEEE International Conference on Emerging Technologies and Factory Automation*. IEEE, pp. 1–8. DOI: 10.1109/ETFA.2016.7733572.

Lunardi, Willian T. et al. (submitted). "Metaheuristics for the Online Printing Shop Scheduling Problem". In: *Manuscript Submitted to the European Journal of Operational Research*.

Lunardi, Willian T. et al. (to appear). "Mixed Integer Linear Programming and Constraint Programming Models for the Online Printing Shop Scheduling Problem". In: *Computers & Operations Research*.

Ma, Ying, Chengbin Chu, and Chunrong Zuo (2010). "A Survey of Scheduling With Deterministic Machine Availability Constraints". In: *Computers & Industrial Engineering* 58.2, pp. 199–211. DOI: 10.1016/j.cie.2009.04.014.

Mastrolilli, M. and L. M. Gambardella (1999). *Effective Neighborhood Functions for the Flexible Job Shop Problem: Appendix*. http://people.idsia.ch/~monaldo/fjspresults/fjsp_result.ps. accessed on April 10, 2020.

– (2000). "Effective Neighbourhood Functions for the Flexible Job Shop Problem". In: *Journal of Scheduling* 3.1, pp. 3–20. DOI: 10.1002/(SICI)1099-1425(200001/02)3:1<3::AID-JOS32>3.0.CO;2-Y.

Meng, Tao, Quan-Ke Pan, and Hong-Yan Sang (2018). "A Hybrid Artificial Bee Colony Algorithm for a Flexible Job Shop Scheduling Problem With Overlapping in Operations". In: *International Journal of Production Research* 56.16, pp. 5278–5292. DOI: 10.1080/00207543.2018.1467575.

Mirjalili, Seyedali and Andrew Lewis (2016). "The Whale Optimization Algorithm". In: *Advances in Engineering Software* 95, pp. 51–67. DOI: 10.1016/j.advengsoft.2016.01.008.

Mirjalili, Seyedali, Seyed Mohammad Mirjalili, and Andrew Lewis (2014). "Grey Wolf Optimizer". In: *Advances in Engineering Software* 69, pp. 46–61. DOI: 10.1016/j.advengsoft.2013.12.007.

Nelder, John A and Roger Mead (1965). "A Simplex Method for Function Minimization". In: *The computer journal* 7.4, pp. 308–313. DOI: 10.1093/comjnl/7.4.308.

Nie, Li et al. (2013). "A GEP-Based Reactive Scheduling Policies Constructing Approach for Dynamic Flexible Job Shop Scheduling Problem With Job Release Dates". In: *Journal of Intelligent Manufacturing* 24.4, pp. 763–774. DOI: 10.1007/s10845-012-0626-9.

Nocedal, Jorge and Stephen Wright (2006). *Numerical Optimization*. Springer Science & Business Media. DOI: 10.1007/978-0-387-40065-5.

Nuijten, Wilhelmus Petronella Maria and Emile HL Aarts (1996). "A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling". In: *European Journal of Operational Research* 90.2, pp. 269–284. DOI: 10.1016/0377-2217(95)00354-1.

Özgüven, Cemal, Lale Özbakır, and Yasemin Yavuz (2010). "Mathematical Models for Job-Shop Scheduling Problems With Routing and Process Plan Flexibility". In: *Applied Mathematical Modelling* 34.6, pp. 1539–1548. DOI: 10.1016/j.apm.2009.09.002.

Palacios, J. J. et al. (2015). "Genetic Tabu Search for the Fuzzy Flexible Job Shop Problem". In: *Computers & Operations Research* 54, pp. 74–89. DOI: 10.1016/j.cor.2014.08.023.

Pezzella, Ferdinando, Gianluca Morganti, and Giampiero Ciaschetti (2008). "A Genetic Algorithm for the Flexible Job-Shop Scheduling Problem". In: *Computers & Operations Research* 35.10, pp. 3202–3212. DOI: 10.1016/j.cor.2007.02.014.

Pine, B Joseph (1993). *Mass Customization*. Vol. 17. Harvard Business School Press Boston.

Pinedo, Michael (2012). *Scheduling*. Vol. 5. Springer. DOI: 10.1007/978-1-4614-2361-4.

Ploya, Gyoergy (1945). *How to Solve It: A New Aspect of Mathematical Method*. Princeton University Press. ISBN: 069111966X.

Puchinger, Jakob and Günther R Raidl (2005). "Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization: A Survey and Classification". In: *International Work-Conference on the Interplay Between Natural and Artificial Computation*. Springer, pp. 41–53. DOI: 10.1007/11499305_5.

Qin, A. K., V. L. Huang, and P. N. Suganthan (2008). "Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization". In: *IEEE Transactions on Evolutionary Computation* 13.2, pp. 398–417. DOI: 10.1109/TEVC.2008.927706.

Rajabinasab, Amir and Saeed Mansour (2011). "Dynamic Flexible Job Shop Scheduling With Alternative Process Plans: An Agent-Based Approach". In: *The International Journal of Advanced Manufacturing Technology* 54.9-12, pp. 1091–1107. DOI: 10.1007/s00170-010-2986-7.

Rossi, A. and M. Lanzetta (2020). "Integration of Hybrid Additive/Subtractive Manufacturing Planning and Scheduling by Metaheuristics". In: *Computers & Industrial Engineering* 144, Article ID 106428. DOI: 10.1016/j.cie.2020.106428.

Rossi, Andrea (2014). "Flexible Job Shop Scheduling With Sequence-Dependent Setup and Transportation Times by Ant Colony With Reinforced Pheromone Relationships". In: *International Journal of Production Economics* 153, pp. 253–267. DOI: 10.1016/j.ijpe.2014.03.006.

Rossi, Francesca, Peter Van Beek, and Toby Walsh (2006). *Handbook of Constraint Programming*. Elsevier. ISBN: 0444527265.

Salman, Ayed, Imtiaz Ahmad, and Sabah Al-Madani (2002). "Particle Swarm Optimization for Task Assignment Problem". In: *Microprocessors and Microsystems* 26.8, pp. 363–371. DOI: 10.1016/S0141-9331(02)00053-4.

Saygin, C and SE Kilic (1999). "Integrating Flexible Process Plans With Scheduling in Flexible Manufacturing Systems". In: *The International Journal of Advanced Manufacturing Technology* 15.4, pp. 268–280. DOI: 10.1007/s001700050066.

Shen, Xiao-Ning and Xin Yao (2015). "Mathematical Modeling and Multi-Objective Evolutionary Algorithms Applied to Dynamic Flexible Job Shop Scheduling Problems". In: *Information Sciences* 298, pp. 198–224. DOI: 10.1016/j.ins.2014.11.036.

Shen, Zuo-Jun Max, Roger Lezhou Zhan, and Jiawei Zhang (2011). "The Reliable Facility Location Problem: Formulations, Heuristics, and Approximation Algorithms". In: *Journal on Computing* 23.3, pp. 470–482. DOI: 10.1287/ijoc.1100.0414.

Sim, Kwang Mong and Weng Hong Sun (2003). "Ant Colony Optimization for Routing and Load-Balancing: Survey and New Directions". In: *Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans* 33.5, pp. 560–572. DOI: 10.1109/TSMCA.2003.817391.

Simon, Herbert A and Allen Newell (1958). "Heuristic Problem Solving: The Next Advance in Operations Research". In: *Operations research* 6.1, pp. 1–10. DOI: 10.1287/opre.6.1.1.

Sörensen, Kenneth (2015). "Metaheuristics—the Metaphor Exposed". In: *International Transactions in Operational Research* 22.1, pp. 3–18. DOI: 10.1111/itor.12001.

Sörensen, Kenneth and Fred Glover (2013). "Metaheuristics". In: *Encyclopedia of operations research and management science* 62, pp. 960–970. URL: pdfs.semanticscholar.org/e88e/71f348c831992a1ec4fff2beb1ee7cfc93b7.pdf.

Sörensen, Kenneth et al. (2012). "Metaheuristics for the Multimodal Optimization of Hazmat Transports". In: *Security Aspects of Uni-and Multimodal Hazmat Transportation Systems*, pp. 163–181. DOI: 10.1002/9783527664818.ch10.

Storn, Rainer and Kenneth Price (1997). "Differential Evolution–A Simple and Efficient Heuristic for Global Optimization Over Continuous Spaces". In: *Journal of Global Optimization* 11.4, pp. 341–359. DOI: 10.1023/A:1008202821328.

Talbi, El-Ghazali (2009). *Metaheuristics: From Design to Implementation*. Vol. 74. John Wiley & Sons. ISBN: 978-0-470-27858-1.

Tamssaouet, Karim, Stéphane Dauzère-Pérès, and Claude Yugma (2018). "Metaheuristics for the Job-Shop Scheduling Problem With Machine Availability Constraints". In: *Computers & Industrial Engineering* 125, pp. 1–8. DOI: 10.1016/j.cie.2018.08.008.

T'kindt, Vincent and Jean-Charles Billaut (2006). *Multicriteria Scheduling: Theory, Models and Algorithms*. Springer-Verlag Berlin Heidelberg. DOI: 10.1007/b106275.

Tsai, J.-T., J.-C. Fang, and J.-H. Chou (2013). "Optimized Task Scheduling and Resource Allocation on Cloud Computing Environment Using Improved Differential Evolution Algorithm". In: *Computers & Operations Research* 40.12, pp. 3045–3055. DOI: 10.1016/j.cor.2013.06.012.

Varrette, Sébastien et al. (2014). "Management of an Academic HPC Cluster: The UL Experience". In: *International Conference on High Performance Computing & Simulation (HPCS)*. IEEE, pp. 959–967. DOI: 10.1109/HPCSim.2014.6903792.

Vilcot, Geoffrey and Jean-Charles Billaut (2008). "A Tabu Search and a Genetic Algorithm for Solving a Bicriteria General Job Shop Scheduling Problem". In: *European Journal of Operational Research* 190.2, pp. 398–411. DOI: 10.1016/j.ejor.2007.06.039.

Vilím, Petr, Philippe Laborie, and Paul Shaw (2015). "Failure-Directed Search for Constraint-Based Scheduling". In: *Integration of AI and OR Techniques in Constraint Programming*. Ed. by Laurent Michel. Springer International Publishing, pp. 437–453. DOI: 10.1007/978-3-319-18008-3_30.

Vital-Soto, A., A. Azab, and M. F. Baki (2020). "Mathematical Modeling and a Hybridized Bacterial Foraging Optimization Algorithm for the Flexible Job-Shop

Scheduling Problem With Sequencing Flexibility". In: *Journal of Manufacturing Systems* 54, pp. 74–93. DOI: 10.1016/j.jmsy.2019.11.010.

Vitaliy, Feoktistov (2006). *Differential Evolution–In Search of Solutions*. DOI: 10.1007/978-0-387-36896-2.

Wagneur, Edouard and Chelliah Sriskandarajah (1993). "Openshops With Jobs Overlap". In: *European Journal of Operational Research* 71.3, pp. 366–378. DOI: 10.1016/0377-2217(93)90347-P.

Wang, L. et al. (2010). "A Novel Hybrid Discrete Differential Evolution Algorithm for Blocking Flow Shop Scheduling Problems". In: *Computers & Operations Research* 37.3, pp. 509–520. DOI: 10.1016/j.cor.2008.12.004.

Wang, Shijin and Jianbo Yu (2010). "An Effective Heuristic for Flexible Job-Shop Scheduling Problem With Maintenance Activities". In: *Computers & Industrial Engineering* 59.3, pp. 436–447. DOI: 10.1016/j.cie.2010.05.016.

Wang, Yi et al. (2017). "Industry 4.0: A Way From Mass Customization to Mass Personalization Production". In: *Advances in Manufacturing* 5.4, pp. 311–320. DOI: 10.1007/s40436-017-0204-7.

Wilbrecht, Jon K and William B Prescott (1969). "The Influence of Setup Time on Job Shop Performance". In: *Management Science* 16.4, B–274. DOI: 10.1287/mnsc.16.4.B274.

Wu, Jun, Jinsong Wang, and Zheng You (2010). "An Overview of Dynamic Parameter Identification of Robots". In: *Robotics and computer-integrated manufacturing* 26.5, pp. 414–419. DOI: 10.1016/j.rcim.2010.03.013.

Xiong, Hegen et al. (2017). "A Simulation-Based Study of Dispatching Rules in a Dynamic Job Shop Scheduling Problem With Batch Release and Extended Technical Precedence Constraints". In: *European Journal of Operational Research* 257.1, pp. 13–24. DOI: 10.1016/j.ejor.2016.07.030.

Xiong, Jian, Li-ning Xing, and Ying-wu Chen (2013). "Robust Scheduling for Multi-Objective Flexible Job-Shop Problems With Random Machine Breakdowns". In: *International Journal of Production Economics* 141.1, pp. 112–126. DOI: 10.1016/j.ijpe.2012.04.015.

Xu, Ye et al. (2015). "An Effective Teaching–Learning-Based Optimization Algorithm for the Flexible Job-Shop Scheduling Problem With Fuzzy Processing Time". In: *Neurocomputing* 148, pp. 260–268. DOI: 10.1016/j.neucom.2013.10.042.

Yang, Wen-Hwa (1999). "Survey of Scheduling Research Involving Setup Times". In: *International Journal of Systems Science* 30.2, pp. 143–155. DOI: 10.1080/002077299292498.

Yang, Xin-She (2010). "Firefly Algorithm, Stochastic Test Functions and Design Optimisation". In: *International Journal of Bio-Inspired Computation* 2.2, 78–84. DOI: 10.1504/IJBIC.2010.032124.

– (2012). "Nature-inspired mateheuristic algorithms: Success and new challenges". In: *arXiv preprint arXiv:1211.6658*. DOI: 10.4172/2324-9307.1000e101.

– (2013). "Multiobjective Firefly Algorithm for Continuous Optimization". In: *Engineering With Computers* 29.2, pp. 175–184. DOI: 10.1007/s00366-012-0254-1.

Yu, Lianfei et al. (2017). "An Extended Flexible Job Shop Scheduling Model for Flight Deck Scheduling With Priority, Parallel Operations, and Sequence Flexibility". In: *Scientific Programming* 2017. DOI: 10.1155/2017/2463252.

Yuan, Y. and H. Xu (2013). "Flexible Job Shop Scheduling Using Hybrid Differential Evolution Algorithms". In: *Computers & Industrial Engineering* 65.2, pp. 246–260. DOI: 10.1016/j.cie.2013.02.022.

Zandieh, Mostafa, SMT Fatemi Ghomi, and SM Moattar Husseini (2006). "An Immune Algorithm Approach to Hybrid Flow Shops Scheduling With Sequence-Dependent Setup Times". In: *Applied Mathematics and Computation* 180.1, pp. 111–127. DOI: 10.1016/j.amc.2005.11.136.

Zhang, Jiae and Jianjun Yang (2016). "Flexible Job-Shop Scheduling With Flexible Workdays, Preemption, Overlapping in Operations and Satisfaction Criteria: An Industrial Application". In: *International Journal of Production Research* 54.16, pp. 4894–4918. DOI: 10.1080/00207543.2015.1134839.

Zhang, Qiao, Hervé Manier, and M-A Manier (2012). "A Genetic Algorithm With Tabu Search Procedure for Flexible Job Shop Scheduling With Transportation Constraints and Bounded Processing Times". In: *Computers & Operations Research* 39.7, pp. 1713–1723. DOI: 10.1016/j.cor.2011.10.007.