N° d'ordre: 08/2017-C/MT

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de L'enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie Houari Boumediène
Faculté de Mathématiques

THÈSE
Présentée pour l'obtention du diplôme de DOCTORAT 3ème Cycle

EN : MATHEMATIQUES

Spécialité: Recherche Opérationnelle et Mathématiques Discrètes.

Par: Nour El-Houda TELLACHE

Sujet

ORDONNANCEMENT D'ATELIER EN PRÉSENCE D'UN GRAPHE DE CONFLITS

Soutenue publiquement le 16/03/2017, devant le jury composé de :

M. BELBACHIR Hacène	Professeur, USTHB,	Président
M. BOUDHAR Mourad	Professeur, USTHB,	Directeur de thèse
Mme BRAUNER-VETTIER Nadia	Professeur, UJF, France,	Examinatrice
M. BERRACHEDI Abdelhafid	Professeur, USTHB,	Examinateur
M. DE WERRA Dominique	Professeur, EPFL, Suisse,	Examinateur
M. YALAOUI Farouk	Professeur, UTT, France,	Examinateur
M. BENDRAOUCHE Mohamed	MCA, USDBlida,	Examinateur
M. CHERGUI Mohamed El-Amine	MCA, USTHB,	Examinateur

Ordonnancement d'atelier en présence d'un graphe de conflits

Résumé

Ce sujet de thèse concerne l'étude des problèmes d'ordonnancement d'atelier de type open shop et flow shop où certaines tâches sont en conflit et ne peuvent pas être exécutées simultanément sur deux machines différentes. Ces contraintes sont données par un graphe appelé graphe de conflit. Dans ce graphe, les sommets représentent les tâches et deux sommets sont adjacents si et seulement si les tâches correspondantes sont en conflit. L'objectif est de minimiser la date de fin de traitement de l'ordonnancement (makespan).

Nous traitons, tout d'abord, le open shop avec graphe de conflit. Nous montrons la NP-difficulté de différentes versions de ce problème et nous donnons des cas polynomiaux. Une approche heuristique à deux phases et des bornes inférieures sont ensuite proposées. Cette approche est basée sur la construction des couplages dans des graphes bipartis ainsi que sur une méthode d'insertion combinée avec Beam Search. Des expérimentations numériques ont été réalisées pour tester l'efficacité de l'approche proposée.

Le problème de flow shop avec graphe de conflit est étudié de façon similaire. Nous montrons que la minimisation du makespan est NP-difficile pour différentes versions et nous donnons quelques cas polynomiaux. Nous montrons aussi que les ordonnancements de permutation ne sont pas dominants sur deux machines et nous discutons un cas particulier où l'ordonnancement optimal peut être une permutation. D'autre part, nous proposons des bornes inférieures et une panoplie de méthodes: (a) des modèles mathématiques pour différentes versions de ce problème; (b) une méthode Branch and Bound pour le cas de deux machines et des temps d'exécution unitaires; (c) deux approches heuristiques pour le cas général. La solution obtenue par les approches heuristiques est comparée à la solution optimale obtenue par le modèle mathématique pour des instances de petites tailles et aux bornes inférieures pour des instances de grandes tailles. Finalement, des analyses expérimentales extensives ont été réalisées pour tester l'efficacité des méthodes proposées.

L'ordonnancement avec conflit est rencontré dans les problèmes d'ordonnancement sous contraintes de ressources quand les ressources sont non-partageables. Nous étudions dans cette thèse la relation entre l'ordonnancement d'atelier avec graphe de conflit et l'ordonnancement d'atelier sous contraintes de ressources. À partir de cette relation, de nouveaux résultats en complexité de ce dernier problème sont obtenus.

Mots clés: ordonnancement; open shop; flow shop; graphe de conflit; complexité; contrainte de ressource; heuristiques; bornes inférieures; programmation en nombres entiers; branch and bound; makespan.

Shop scheduling with conflict graph

Abstract

The research in this thesis addresses the problem of scheduling on dedicated machines subject to conflict constraints given by a simple and undirected graph G, called the conflict graph. Each vertex in G represents a job and jobs that are represented by adjacent vertices in G are conflicting and cannot be processed simultaneously on different machines. We mainly deal with two models of dedicated machines: open shop and flow shop. The objective is to find a schedule that minimizes the maximum completion time, also called the makespan.

We begin by considering the open shop problem with conflict graph. We first prove the NP-hardness of different versions of this problem. Then, we provide polynomial-time solvable cases, for which we devise simple and efficient algorithms. On the other hand, we present a two-phase heuristic approach and lower bounds for the general case. The heuristic approach is based on the construction of matchings in bipartite graphs as well as on a specific insertion technique combined with beam search. Finally, we test the efficiency of this approach and report our computational experiments that display satisfying results.

The flow shop scheduling problem with conflict graph is treated similarly. We first show that the problem of minimizing the maximum completion time is NP-hard for several relevant restricted versions. Then, we present some polynomial-time solvable cases. We also prove that the permutation schedules are not dominant even for two machines, and we discuss a special case for which an optimal schedule can be found by a permutation schedule. We complement our study by a description and analysis of algorithms for restricted cases as well as the general case. These algorithms include: mathematical models for different versions of the problem under study, a branch and bound algorithm for the case of two machines with unit-time operations and two heuristic approaches for the general problem. The quality of the solutions produced by the heuristic approaches is compared to five lower bounds for large instances and to the optimal solutions obtained by the mathematical model for small instances. Finally, extensive computational experiments performed on a wide range of test problems are provided to measure the performance of the proposed algorithms.

Scheduling with conflicts can be encountered in the resource-constrained scheduling problems when the resources are non-sharable. We study, in this thesis, the relation between shop scheduling with conflict graph and shop scheduling under resource constraints. From the developed relation, new complexity results of the later problem will be derived.

Keywords: scheduling; open shop; flow shop; conflict graphs; complexity; resource constraints; heuristics; lower bounds; mixed-integer linear programming; branch and bound; makespan.

To all the ones I love . . .

Acknowledgments

In the name of Allah, the Most Gracious and the Most Merciful.

First and foremost, Alhamdulillah, praises and thanks to Allah, the Almighty, for His showers of blessings throughout my research work to complete the research successfully.

I would like to express my deepest gratitude to my supervisor Prof. Mourad BOUDHAR for suggesting me this topic, without whose expert guidance, support, patience and encouragement none of this would have been possible. The extensive knowledge, vision, and creative thinking of Prof. Mourad BOUDHAR have been the source of inspiration for me throughout this work. He has taught me the methodology to carry out the research and to present the research works as clearly as possible. It was a great privilege and honour to work and study under his guidance. I am extremely grateful for what he has offered me.

Profuse thanks go equally to the committee members: Prof. Hacène BELBACHIR for kindly accepting to be president of the jury, Dr. Mohamed BENDRAOUCHE, Prof. Abdelhafid BERRACHEDI, Prof. Nadia BRAUNER-VETTIER, Dr. Mohamed El-Amine CHERGUI, Prof. Dominique DE WERRA and Prof. Farouk YALAOUI for kindly accepting to be examiners of the thesis. I would especially like to thank them for taking the time to evaluate this work and participate in my defense.

Contents

Introduction 1

1 Background and state of the art 3

1.1 Scheduling theory . . . 3

1.1.1 Scheduling on parallel machines . . . 4

1.1.2 Scheduling on dedicated machines . . . 5

1.2 Graph theory . . . 13

1.3 Scheduling with conflict/agreement graph . . . 16

1.3.1 Scheduling with conflict/agreement graph on identical machines . . . 16

1.3.2 Mutual exclusion scheduling . . . 18

1.4 Scheduling under resource constraints . . . 20

1.5 Conclusion . . . 22

2 Open shop problem 23

2.1 Related works . . . 24

2.2 Complexity issues . . . 25

2.2.1 NP-hardness results . . . 26

2.2.2 Well solvable cases . . . 35

2.3 Lower bounds . . . 42
2.4 Two-phase heuristic approach . . . 43
2.4.1 First phase . . . 43
2.4.2 Second phase . . . 45
2.5 Computational experiments . . . 50
2.6 Conclusion . . . 57

3 Flow shop problem **59**
3.1 Related works . . . 60
3.2 Complexity issues . . . 61
3.3 Permutation vs. non permutation schedules . . . 69
3.4 Lower bounds . . . 70
3.5 On the mathematical model . . . 72
3.5.1 Scheduling on m machines . . . 72
3.5.2 Scheduling on two machines with unit-time operations . . . 74
3.6 Branch and bound algorithm . . . 77
3.6.1 Branching rule . . . 77
3.6.2 Selecting rule . . . 77
3.6.3 Lower bounds . . . 78
3.6.4 Upper bounds (heuristics) . . . 80
3.6.5 Algorithm scheme . . . 81
3.7 Heuristic approaches . . . 82
3.7.1 Makespan computation . . . 83

3.7.2 First heuristic approach: H1mFSCn 85

3.7.3 Second heuristic approach: H2mFSCn 86

3.8 Computational experiments 88

3.8.1 Case of unit-time operations 88

3.8.2 Case of arbitrary processing times 95

3.9 Conclusion 105

Conclusions and future directions **107**

References **120**

Listing of figures

1.1.1 Optimal schedule for an instance of $O3||C_{max}$ with three jobs such that each colour represents a job. 6

1.1.2 Optimal schedule for an instance of $F3||C_{max}$ with three jobs such that each colour represents a job 9

1.1.3 Schedules for an instance of $F4||C_{max}$ with two jobs such that each colour represents a job. (a) one permutation schedule. (b) other permutation schedule. (c) optimal schedule. 10

1.2.1 The Petersen graph (on the left) and its complement graph (on the right) and the graph with grey edges is the complete graph. 14

1.2.2 (a) bipartite graph. (b) 3-partite graph. (c) split graph. (d) the Grötzsch graph which is a triangle-free graph 14

1.4.1 (a) resource requirements of each job. (b) associated conflict graph. (c) a feasible schedule. 21

2.2.1 Instance I of Theorem 2.2: agreement graph $\overline{G} = (S_1, S_2; \overline{E})$. 27

2.2.2 Instance I of Theorem 2.2: schedule of the jobs of $V_{M'}$ and their corresponding jobs of V_Y and V_Z. 28

2.2.3 Instance I of Theorem 2.2: schedule of the jobs of V_i. 28

2.2.4 Instance I of Theorem 2.2: schedule of the remaining jobs of V_M and the jobs of V_Y and V_Z. 28

2.2.5 Instance I of Theorem 2.4: gadget $\overline{G_m} = (V_m, \overline{E_m})$. 30

2.2.6 Instance I of Theorem 2.4: schedule of the jobs of $\overline{G_{m'}}$. 31

2.2.7 Instance I of Theorem 2.4: schedule of the internal jobs of $\overline{G_m}$. 32

2.2.8 Instance I of Theorem 2.4: schedule corresponding to Case 2. 33

2.2.9 Instance I of Theorem 2.4: schedule corresponding to Case 3. 33

2.2.10 Some transformations from $\overline{G} = (V, \overline{E})$ to $\overline{G'} = (V', \overline{E'})$: (a) jobs with one operation on different machines,(b) jobs with two operations, (c) jobs with one operation on a same machine, (d) jobs with different numbers of operations. 36

2.2.11 Application of the algorithm of Theorem 2.9: (a) processing times. (b) transformation from $\overline{G} = (V, \overline{E})$ to $\overline{G'} = (V', \overline{E'})$ and the construction of the optimal schedule, where the dashed edges represent the paths obtained after applying the algorithm proposed in [53]. 37

2.2.12 Application of the algorithm of Theorem 2.11: (a) agreement graph $\overline{G} = (V, \overline{E})$. (b) bipartite graph $R = (V_1 \cup V_2; U)$ associated, where the dashed edges represent a maximum matching. (c) an optimal schedule. 38

2.2.13 Construction of the network graph of Theorem 2.12: (a) agreement graph $\overline{G} = (V, \overline{E})$. (b) network $N = (V, U_c, C)$ associated. 39

2.2.14 Application of the algorithm of Theorem 2.12: (a) processing times. (b) a solution of the maximum flow problem of the network $N = (V, U_c, C)$, where the value of the maximum flow is equal to 9. (c) an optimal schedule. 41

2.2.15 Application of the algorithm of Theorem 2.13: (a) agreement graph $\overline{G} = (V, \overline{E})$. (b) 3-partite graph $R = (V_1 \cup V_2 \cup V_3, U)$ associated, where the dashed edges represent a maximum matching. (c) an optimal schedule. 42

2.4.1 Variants of beam search. 46

2.4.2 Example 2.1: (a) processing times. (b) conflict graph $G = (V, E)$. 47

2.4.3 Example 2.1: computing a min-max matching in B_{S_1}. 48

2.4.4 Example 2.1: an "optimal" schedule with $C_{max} = 373$. 49

2.4.5 Example 2.1: beam search tree. 49

2.5.1 Comparison: Beam2 vs Beam3 with respect to APD(%). 57

2.5.2 Comparison: Beam2 vs Beam3 with respect to Nopt. 57

3.2.1 Instance I of Theorem 3.2. (a) processing times. (b) agreement graph $\overline{G} = (V, \overline{E})$. . . 62

3.2.2 2-Partition reduces to $F2|ConfG = (V, E)|C_{max}$. 63

3.2.3 2-Partition reduces to $F2|ConfG = (V, E), pmtn|C_{max}$. 65

3.2.4 Hamiltonian path problem reduces to $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$. 66

3.2.5 Application of the algorithm of Theorem 3.9. (a) processing times. (b) construction of $\overline{G}'$ and the optimal schedule. 68

3.2.6 Instance of $F2|ConfG = (V, E)|C_{max}$. (a) conflict graph $G = (V, E)$. (b) processing times. (c) an optimal schedule. (d) an optimal permutation schedule. 69

3.3.1 Transformation of an optimal schedule of $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$. 70

3.6.1 Conflict graph G. (a) Example 3.1. (b) Example 3.2. 81

3.6.2 Example 3.2: search tree of the branch and bound algorithm. 82

3.7.1 Example 3.3: (a) processing times. (b) conflict graph $G = (V, E)$. 84

3.7.2 Example 3.3: schedule of the sequence $J_2 J_3 J_1 J_4$ given by Algorithm 7. 85

3.8.1 B&B3: Number of unsolved instances as a function of p. 94

3.8.2 Performance of the lower bounds for $m = 5$. 101

3.8.3 Performance of the lower bounds for $m = 10$. 101

3.8.4 Performance of the lower bounds for $m = 20$. 101

3.8.5 Comparison: H1mFSCn vs H2mFSCn for $m = 5$. 101

3.8.6 Comparison: H1mFSCn vs H2mFSCn for $m = 10$. 102

3.8.7 Comparison: H1mFSCn vs H2mFSCn for $m = 20$. 102

Listing of tables

1.3.1 Previous complexity results of the SCG/SAG on identical machines. 18

1.3.2 Previous complexity results of the MES problem. 19

2.1.1 Previous complexity results of the open shop under resource constraints. 25

2.5.1 HmOSCn: experimental results for 4OSC4. 51

2.5.2 HmOSCn: experimental results for 5OSC5. 52

2.5.3 HmOSCn: experimental results for 7OSC7. 53

2.5.4 HmOSCn: experimental results for 10OSC10. 54

2.5.5 HmOSCn: experimental results for 15OSC15. 55

2.5.6 HmOSCn: experimental results for 20OSC20. 56

2.5.7 HmOSCn: best results of the benchmark problems. 58

2.6.1 Complexity results of the OSC problem. 58

2.6.2 New complexity results of the open shop problem under resource constraints. 58

3.1.1 Previous complexity results of the flow shop under resource constraints. 61

3.5.1 Size complexity of the MILP models of $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ 77

3.8.1 Performance of the lower bounds lb_3 and lb_6. 89

3.8.2 Performance of the upper bounds (heuristics). 90

3.8.3 Performance of B&B1, B&B3 and B&B4. 90

3.8.4 Performance of the MILP models (P_3), (P_4) and (P_5). 93

3.8.5 Average CPU times (in seconds) required by B&B3 and (P_4) for $p \geq 0.8$. 94

3.8.7 Computational results of (P_1), H1mFSCn and H2mFSCn on small instances. 96

3.8.6 Comparison between (P_1) and (P_2) for $m = 2$. 99

3.8.8 Performance of the lower bounds. 100

3.8.9 Computational results of H1mFSCn and H2mFSCn on large instances. 102

3.9.1 Complexity results of the FSC problem. 106

3.9.2 New complexity results of the flow shop problem under resource constraints. 106

Listing of algorithms

1 NEH heuristic for $Fm|prmu|C_{max}$ [116]. 13
2 GMIN: greedy algorithm for the maximum independent set problem 15
3 Algorithm for the preemptive version of the two-machine OSC problem. 40
4 Variant of Dijsktra's algorithm for min-max path 44
5 HmOSCn: heuristic approach for the m-machine OSC problem. 47
6 $G(n, p)$ Erdős Rényi method [52] 50
7 Makespan computation for $Fm|ConfG = (V, E), prmu|C_{max}$ 83
8 H1mFSCn: first heuristic approach for the m-machine FSC problem. 85
9 H2mFSCn: second heuristic approach for the m-machine FSC problem 87

Terminology

n number of jobs to be scheduled.

m number of machines.

J_j job j ($j = 1, \ldots, n$).

M_i machine i ($i = 1, \ldots, m$).

J_{ij} operation i of J_j, i.e. the operation of job J_j processed on machine M_i.

p_{ij} processing time of operation J_{ij}.

p_j processing time of job J_j.

C_j completion time of job J_j.

r_j ready time or release date of J_j, which is the time at which job J_j is ready for processing.

d_j due date of J_j, which specifies a time limit by which J_j should be completed. Usually, penalty functions are defined in accordance with due dates.

$L_j = C_j - d_j$ lateness of J_j. Note that L_j may be positive or negative (in which case it is early).

Fm the simple flow shop with m machines.

Om the simple open shop with m machines.

Pm scheduling on m identical machines.

pmtn preemption (i.e. interruption) of an operation (or job) is permitted. The preempted operation (or job) may be resumed at a later time.

perm in the simple flow shop, this signifies that permutation schedules only are considered: those schedules in which the same job order is maintained at all machines. Such a shop is called a permutation flow shop.

prec precedence relationships exist between jobs.

intree the precedence constraints have the form of an intree, which is a connected acyclic graph where every node has precisely one direct successor, except for the root node which has no successor. It may be called a terminally rooted tree or assembly tree.

outtree the precedence constraints have the form of an outtree, which is an intree with all precedence relations reversed.

$no - wait$ jobs are not allowed to wait between two successive machines, i.e. a job once finished on one machine must immediatly be started on the next machine.

block with blocking constraint a job, having completed processing on a machine, remains on this machine and blocks it, until the next machine downstream becomes available for processing.

C_{max} the completion time of the last job to leave the system. This criterion, commonly called the makespan, has been by far the most exhaustively studied.

$\sum C_j$ total completion time of all jobs, $\sum_{j=1}^{n} C_j$, which is equivalent to the mean completion time (they differ only by a constant factor n).

L_{max} the maximum lateness, L_{max}, is defined as $\max\{L_1, \ldots, L_n\}$. It measures the worst violation of the due dates.

The last thing one discovers in composing a work is what to put first.

Blaise Pascal

Introduction

Scheduling is a decision-making process that concerns the allocation of resources to a set of activities with the aim of optimizing one or more objectives while satisfying some constraints. It has been a subject of a significant amount of literature since the early 1950s. In today's world of global competition, effective scheduling becomes imperative in order to meet customer requirements as quickly as possible while maximizing the profits [127]. Of course, there are many aspects concerning approaches for modelling and solving scheduling problems which are of general methodological importance. Emphasis has been on investigating machine scheduling problems where jobs represent activities and machines represent resources. Scheduling problems belong to a broad class of combinatorial optimization problems, and the majority of these problems are computationally hard and it is extremely unlikely that someone could find an efficient method (i.e. a polynomial algorithm) for solving them exactly.

In this thesis, we will be concerned with the machine scheduling problems in which a set of jobs has to be processed on a set of machines. A job consists of a list of operations, each of which requires processing on a given machine during a period of a given length; two operations belonging to the same job cannot be processed at the same time. Each machine can process at most one job at a time. A schedule specifies for each operation the time interval in which it is processed. The objective is to find a schedule that minimizes the makespan, i.e. the overall time needed to process all the jobs, subject to given constraints. When the operations of each job can be processed in any order, the model is referred to as an open shop and when the jobs have the same processing order through the machines, the model is referred to as a flow shop.

Scheduling problems are often subject to many types of constraints that appear in many forms: precedence constraints such as "the design of the interface can be started when library programming interface is frozen and the analog-to-digital hardware is 75% completed"; availability constraints such as "three skilled machinists are available in the second shift, four are available in the third shift"; time delay between two successive operations such as "the transfer of a product from one machine to the next requires a certain amount of transportation time"; and combinations such as "The injection moulding machine can run three shifts between maintenance cycles", etc. In this work, we consider an important type of constraints that arises when some jobs conflict with each other, in which case we cannot process them at the same time on different machines. We study the problem when the conflict constraints are given by

a simple and undirected graph, called the conflict graph. Each vertex in this graph represents a job and jobs that are represented by adjacent vertices are conflicting and cannot be processed simultaneously on different machines. These constraints model practical situations in which jobs may share the same resources. For example, processes in a distributed operating system that need access to the same file or device are in conflict. Other applications will be discussed in Chapter 1.

This thesis investigates two scheduling problems: open shop and flow shop problems with conflict graph. It is divided into three chapters.

We begin in Chapter 1 by giving basic concepts on scheduling theory and graph theory required for the remainder of this thesis. Previous works on some machine scheduling and graph theory problems, that are related to the topic of this research, are given. Then, we introduce the problem of scheduling with conflict graph, discuss some motivations and survey the state of the art. The aim of this chapter is to provide essential background on the topic.

Chapter 2 deals with the open shop problem with conflict graph (OSC in short). We address, in the first part, the difficulty of the problem: we prove that OSC is NP-hard even for important restricted problems and we exhibit polynomial-time solvable cases. The second part is devoted to the description and experimental analysis of a two-phase heuristic approach proposed for the general OSC problem. The quality of the solution produced by the heuristic approach is compared to lower bounds that we design for this purpose.

Chapter 3 is devoted to a similar treatment of the flow shop problem with conflict graph (FSC in short). The first part studies the complexity of the FSC problem for various restricted cases and discusses the permutation and the non-permutation schedules when introducing the conflict graph. The second part presents lower bounds on the makespan and goes into a variety of solution methods proposed for the general FSC problem as well as for relevant restricted cases. The proposed algorithms include mathematical models, branch and bound algorithm and heuristic approaches. We supplement our study by extensive computational experiments to measure the performance of the proposed approaches.

At the end of the thesis we describe the main contributions of this research and draw some conclusions. Furthermore, we present the lines of research that can be further investigated on the basis of the results presented in this work.

By indirections find directions out.

William Shakespeare

1

Background and state of the art

The topic of this dissertation, shop scheduling with conflict graph, is introduced in this chapter as follows. Section 1.1 provides some important background informations on scheduling theory and goes through the most relevant related works published in the literature on machine scheduling, including: identical machines, open shop and flow shop problems. In Section 1.2, we give the standard graph theoretic concepts used throughout this thesis and discuss some graph theory problems that will be used in the next chapters. In Section 1.3, we present the problem of scheduling with conflict graph, discuss its applications and survey the state of the art of this problem. The last section of the chapter describes the resource-constrained scheduling problem and analyses the relation between this problem and the problem of scheduling with conflict graph. The purpose of this chapter is to provide, in a central place, the theoretical concepts and the previous works that are of direct relevance to this thesis. References are given at the appropriate places for the reader interested in a more detailed exposition.

1.1 Scheduling theory

Motivated and stimulated by questions that arise in production planning and computer control, scheduling theory has become an important subarea of combinatorial optimization, located at the interface between applied mathematics, computer science, and operations research. In its broadest sense, 'scheduling is the allocation of resources over time to perform a collection of tasks' [7], and 'scheduling is con-

cerned with the optimal allocation of scarce resources to activities over time' [97]. The goal is to optimize one or more objectives. The resources and activities can take many different forms. The resources may be machines in a workshop, runways at an airport, crews at a construction site, processing units in a computing environment, and so on. The activities may be operations in a production process, take-offs and landings at an airport, stages in a construction project, executions of computer programs, and so on. Each activity may have a certain priority level, an earliest possible starting time and a due date. The objectives can also take many different forms. One objective may be the minimization of the completion time of the last activity and another may be the minimization of the number of activities completed after their respective due dates.

Some of the first publications in scheduling appeared in Naval Research Logistics Quarterly in the early fifties and contained results by Smith [139], Johnson [88] and Jackson [85]. During the sixties a significant amount of work was done on dynamic programming and integer programming formulations of scheduling problems. After Richard Karp's famous paper on complexity theory [92], the research in the seventies focused mainly on the complexity hierarchy of scheduling problems. In the eighties several different directions were pursued in academia and industry with an increasing amount of attention paid to stochastic scheduling problems. Also, as personal computers started to permeate manufacturing facilities, scheduling systems were being developed for the generation of usable schedules in practice.

In general, a scheduling problem can be modelled by a collection of jobs requiring processing in a certain machine environment. The objective is to sequence these jobs, subject to given constraints, in such a way that one or more performance criteria are optimized. The great variety of scheduling problems motivated Graham et al. [69] to introduce a systematic notation that could serve as a basis for a classification scheme. The notation is composed of three fields $\alpha|\beta|\gamma$. The α field describes the machine environment and contains just one entry. The β field provides details of processing characteristics and constraints and may contain no entry at all, a single entry, or multiple entries. The γ field describes the objective to be minimized and often contains a single entry. We refer to [18, 127] for a detailed analysis on the different entries of this notation (see also Terminology).

There are two general machine environments in classical scheduling theory. The machines may be either parallel, i.e. performing the same functions, or dedicated i.e. specialized for the execution of certain jobs. Scheduling on parallel machines will be discussed in Section 1.1.1 and scheduling on dedicated machines is the subject of Section 1.1.2. For the complexity theoretic concepts, the reader may refer to [18, 60].

1.1.1 Scheduling on parallel machines

Parallel machines may be interpreted as central processors which are able to process every job (i.e. every program). Three types of parallel machines are distinguished depending on their speeds. If all machines have equal job processing speeds, then we call them identical. If the machines differ in their speeds, but

the speed of each machine is constant and does not depend on the job, then they are called uniform. Finally, if the speeds of the machines depend on the particular job processed, then they are called unrelated. The following section presents a brief overview of the case of identical machines, since we will discuss this model in Section 1.3.

1.1.1.1 Identical machines

The problem consists of scheduling a set of jobs characterized by their processing times on a set of identical machines. Each job is assigned to a single machine and each machine can process at most one job at a time. The objective is to minimize the maximum completion time (makespan). This problem, i.e. $Pm||C_{max}$, is not easy to solve, since even simple cases such as scheduling on two machines $P2||C_{max}$ can be proved to be NP-hard by a reduction from the PARTITION problem [60]. However, by allowing preemptions of jobs, it appears that problem $Pm|pmtn|C_{max}$ can be solved very efficiently by an $O(n)$-time algorithm of McNaughton [109]. Now, if the jobs are subject to precedence constraints, i.e. $Pm|prec|C_{max}$, the problem is at least as hard as $Pm||C_{max}$ even when restricted to unit-time jobs [62]. However, constraining the problem further and assuming that the precedence graph takes the form of a forest, either an intree or an outtree, results in a problem that is easily solvable [78]. Moreover, If the number of machines is limited to 2, the problem is solvable in $O(n^2)$-time even for arbitrary precedence graphs, $P2|prec, p_j = 1|C_{max}$, by an algorithm of Coffman and Graham [37].

Since there is no hope of finding a polynomial time algorithm for $Pm||C_{max}$, many approximation methods for this problem have been developed during the last couple of decades. One such methods is the Longest Processing Time first (LPT) heuristic which arranges the jobs in a list in non-increasing order of their processing times. Then, at each step the first available machine is selected to process the first available job on the list. This heuristic tries to place the shorter jobs more towards the end of the schedule, where they can be used for balancing the loads. For more results on identical machines scheduling, see [18, 127].

1.1.2 Scheduling on dedicated machines

Completely different from the above are dedicated machines in which jobs require processing on more than one machine following a certain order known as the processing route. If the processing routes of the jobs are not given in advance, and have to be arbitrarily determined, the scheduling problem is called an open shop. If each job has a fixed and specific processing route, the problem is a job shop. If the processing routes are fixed and are the same for all the jobs, the problem is called a flow shop. Since this thesis is focused on the open shop and the flow shop models, the following two sections are mainly devoted to the literature review of these two problems.

1.1.2.1 Open shop scheduling

The regular open shop scheduling problem consists of a finite set of jobs that has to be processed on a given set of machines. Each job comprises of a set of operations sometimes called tasks. Each operation requires exactly one machine for processing and has a specific processing time. The operations of each job can be processed in any order, but only one at a time. Each machine can process at most one operation at a time. The objective is to schedule all operations, i.e. determine their start times, so as to minimize the maximum of the completion times of all operations (makespan). Figure 1.1.1 uses the so-called Gantt chart (a graphical presentation of a work schedule in which each operation is represented by a bar laid out along a horizontal time axis, the bar's location and length showing the time position and duration of operation [51]) to represent an optimal schedule for an example of three jobs in a three-machine open shop.

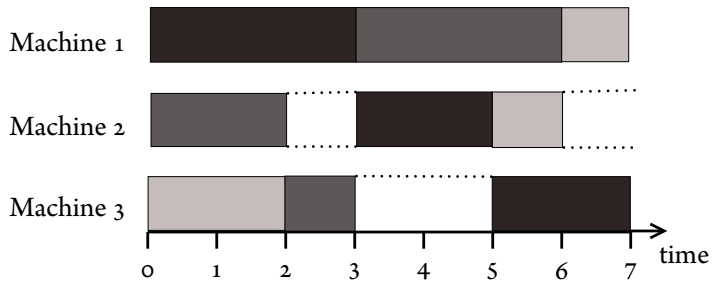


Figure 1.1.1: Optimal schedule for an instance of $O3||C_{max}$ with three jobs such that each colour represents a job.

We present in this section a review about the open shop problem. We start by the complexity results, then we pass to the exact and the approximation methods proposed for $Om||C_{max}$ with $m \geq 3$.

Complexity One of the earliest and most significant efforts related to the open shop scheduling problem was reported by Gonzalez et al. [67] who proposed an $O(n)$-time algorithm for the problem $O2||C_{max}$. These authors showed that preemptions result in a polynomial time algorithm. That is, problem $Om|pmtn|C_{max}$ can be optimally solved. In the general case, however, it turns out that problem $Om||C_{max}$ becomes NP-hard as the number of machines increases to 3 by a reduction from the partition problem [67]. Pinedo [127] presented a simple dispatching rule LAPT (Longest Alternative Processing Time First) which also solves $O2||C_{max}$ in polynomial time. For more work on $Om|pmtn|C_{max}$, see Gonzalez [66].

Most open shop problems are NP-hard and only few non-preemptive problems are known to be polynomial (see [145]). For instance, the two-machine case with two distinct release times, $O2|\{0, r\}|C_{max}$, such that one group of jobs has a zero release time, and the other group has a release time $r > 0$, is shown

to be NP-hard [69]. Its preemptive version, $O2|pmtn, \{0, r\}|C_{max}$, can be solved in polynomial time by linear programming [36]. Moreover, problems $O2|tree|C_{max}$, $O2||\sum C_j$ and $O2||L_{max}$ are NP-hard in the strong sense as proven in [69], [1] and [96] respectively. However, some special structured open shop scheduling problems are still polynomially solvable. For example, Fiala [54] gave a polynomial time algorithm, which solves the problem for arbitrary m, whenever the sum of processing times for one machine is large enough with respect to the maximal processing time. Algorithms for m-machine problems with one or two dominating machines were proposed by Strusevich [140] and summarized in Tanaev et al. [145]. Brucker et al. [28] developed a general method of solving various open shop problems with unit-time operations. They showed that this later problem can be polynomially transformed to a special preemptive scheduling problem on m identical machines. A similar result holds for open shop problems with unit-time operations and no-wait in process. Many results previously published as well as new results have been derived by using these transformations [28, 145].

Exact methods As mentioned above, the open shop problem $Om||C_{max}$ is NP-hard for $m \geq 3$. Studies additionally have shown that within the class of intractable problems the open shop problem belongs to the especially hard ones [29, 72]. As an example, while the famous job shop scheduling problem (which is a close relative of the open shop problem) was easily solvable for instances with up to 100 operations, see e.g. [3, 33, 108], researchers have found difficulties in solving instances of the open shop problem with less than 50 operations. Only few exact solution methods are available for the open shop scheduling problem. A branch and bound algorithm which applies a block-oriented branching scheme and some basic constraint propagation methods for reducing the search tree has been proposed by Brucker et al. [29]. Guéret et al. [72] improved this algorithm by using an intelligent backtracking technique which replaces the simple depth-first search used by the former. They further applied some additional search tree reduction methods in their branch and bound algorithm based on forbidden intervals, i.e. time intervals in which no operation can start or end in an optimal solution [71]. All these exact solution methods are capable of solving smaller open shop instances for which they naturally show a better performance than the heuristic methods. However, even for simple, but larger open shop instances for which the heuristic methods easily find an optimal solution, the performance of the exact solution methods is rather poor, since the search space reduction methods applied are not sufficient to handle the combinatorial explosion. To overcome this, Dorndorf et al. [50] examined additional concepts for reducing the search space. The key to the efficiency of their branch and bound algorithm lies in the following approach: instead of analysing and improving the search strategies for finding solutions, they focused on constraint propagation based methods for reducing the search space. The computational experiments they performed showed that many problem instances were solved to optimality for the first time in a short amount of computation time. On the other hand, Liaw [100] proposed a mixed integer programming formulation for the open shop problem. They used this model to develop an iterative improvement approach based on Bender's decomposition.

Approximation methods Due to the exceptionally intractable nature of the open shop scheduling problem, different approximation methods have been employed to find near optimal values in shorter amounts of time.

Simple list scheduling heuristics based on priority dispatching rules have been examined by Guéret and Prins [70]. Matching algorithms are discussed by Bräsel et al. [27] and Guéret and Prins [70, 71]. The goal of these algorithms is to compute subsets of operations as balanced as possible by solving successively weighted maximum cardinality matching problems in bipartite graphs. Another important class of heuristics are the insertion techniques combined with beam search which have been introduced by Werner and Winkler [160] for the job shop problem and generalized by Bräsel et al. [27] for the open shop problem. We will return to this later class of heuristics and to the matching algorithms in Section 2.4. The shifting bottleneck procedure, originally designed for the job shop problem, has been adapted by Ramudhin and Marier [131] to the open shop problem. Liaw [100] presented an iterative improved approach based on Benders' decomposition applied to a mixed integer programming formulation. They also developed a heuristic dispatching rule for generating initial feasible solutions. Naderi et al. [115] analysed the permutation list encoding scheme, which suffers from redundancy. Many researchers have turned toward the utilization of rank matrix encoding scheme to overcome this shortcoming at the expense of losing other advantages of permutation list. So Naderi et al. [115] presented four efficient theorems to avoid this redundancy and based on these theorems and on an insertion and reinsertion operators they proposed four constructive heuristics for the open shop problem.

As far as metaheuristics are concerned, Liaw [103] presented a hybrid genetic algorithm which incorporates a local improvement procedure based on tabu search into a basic genetic algorithm. This author also proposed within the same year a tabu search technique [101], and a simulated annealing algorithm [102]. Prins [130] proposed another genetic algorithm providing excellent results for the benchmark from the literature. It is shown that good initial solutions in the genetic algorithm can improve the probability of obtaining high quality final solution. Colak and Agarwal [39] developed a neural network based metaheuristic approach that allows integration of domain specific knowledge. Learning strategies imply improved neighbour solutions. Blum and Sampels [19, 20] applied a hybrid approach that combines ant colony optimization with beam search. Shaa and Hsu [137] presented a novel particle swarm optimization along with four decoding schemes based on the principal of active and non-delay schedules. Some of these methods, especially the genetic algorithm of Prins and the ant colony optimization of Blum and Sampels, show a very good performance, and for specific classes of open shop instances, they often are able to find optimal solutions. However, in general, the solutions found for arbitrary open shop instances are of course of a suboptimal nature.

1.1.2.2 Flow shop scheduling

Because of many economic and industrial applications, the flow shop problem has been concentrated on by many researchers with diverse classical assumptions and different objective functions and by im-

plementing various optimization techniques. The formulation of the flow shop problem is the same as for the open shop problem except that the order of processing operations comprising one job is the same of all the jobs.

Thus, the regular flow shop problem consists of different machines that perform operations of jobs. All jobs have the same processing order through the machines, i.e. a job is composed of an ordered list of operations where the i^{th} operation of each job is determined by the same machine required and the processing time on it. The problem is to find the job sequences on the machines which minimize the maximum completion time (makespan) given the additional constraints that (a) operations which belong to the same job and (b) operations which use the same machine cannot be processed simultaneously. Figure 1.1.2 represents an example of three job (the same jobs of Figure 1.1.1) in a three-machine flow shop.

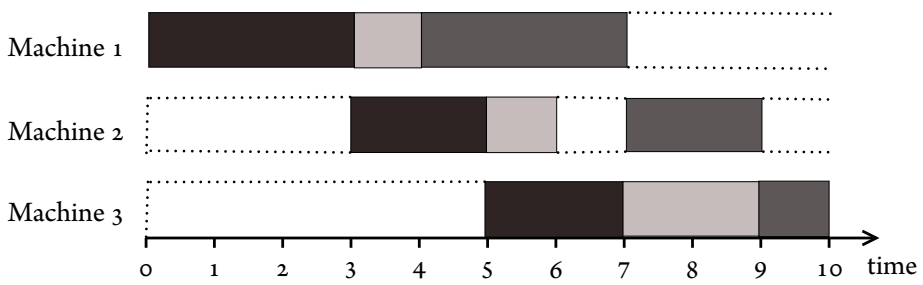


Figure 1.1.2: Optimal schedule for an instance of $F3||C_{max}$ with three jobs such that each colour represents a job

Most of the literature on flow shop scheduling is limited to a particular case of flow shop: the permutation flow shops, in which each machine processes the jobs in the same order. Thus, in a permutation flow shop once the job sequence on the first machine is fixed it will be kept on all remaining machines. The resulting schedule is called permutation schedule. Using the notation specified in Pinedo [127], this problem is denoted $Fm|prmu|C_{max}$. Of course, changing the sequence of jobs between the machines may at times result in a smaller makespan. But, if we could confine our attention to permutation schedules only, our search for an optimal schedule would be greatly simplified, since there are just $n!$ job orderings, but $(n!)^m$ ways the jobs can be independently ordered on each machine. There are situations where only permutation schedules are acceptable, either because it is technologically impossible for one job to pass another, or by management fiat, to keep things operationally simple. In addition, there are cases when it can be shown that a permutation schedule will always be optimal over all schedules. Indeed, Johnson [88] showed that there exists an optimal flow shop schedule with the same job order on the first two machines M_1 and M_2 as well as the same job order on the last two machines M_{m-1} and M_m. This implies that there is an optimal permutation schedule for $F2||C_{max}$ and for $F3||C_{max}$. However, Potts et al. [129] showed that this restriction can be costly for $m \geq 4$ in terms of the deviation of the makespans of the optimal permutation schedule and the optimal flow shop schedule (see e.g. Figure 1.1.3). They showed that there are instances for which the objective value of the optimal permutation

schedule is much worse (in a factor more than $\frac{1}{2}\sqrt{m}$) than that of the optimal flow shop schedule.

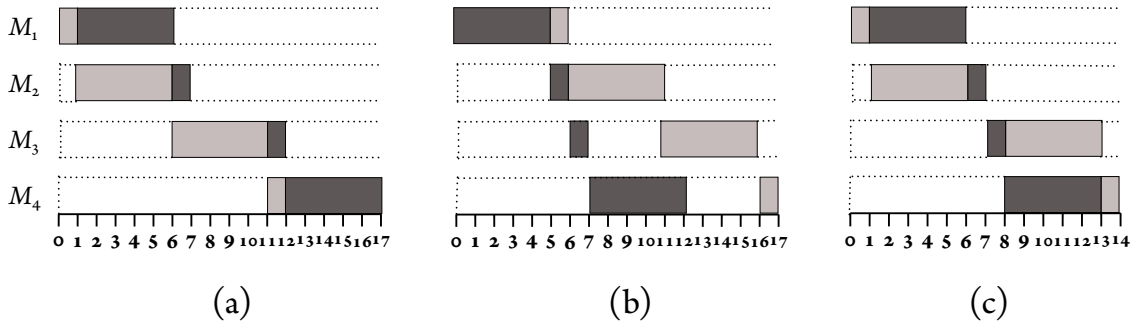


Figure 1.1.3: Schedules for an instance of $F4||C_{max}$ with two jobs such that each colour represents a job. (a) one permutation schedule. (b) other permutation schedule. (c) optimal schedule.

In what follows, we survey the existing literature about the flow shop problem. We start by the complexity results, then we pass to the exact and the approximation methods proposed for this problem.

Complexity The $F2||C_{max}$ was one of the first problems to be analysed in the early days of Operations Research and led to a classical paper [88] in scheduling theory by S.M. Johnson. The author proved that $F2||C_{max}$ can be solved in $O(nlogn)$-time and the proposed algorithm is commonly referred to as Johnson's rule. This algorithm can be applied to $F3||C_{max}$ and yields an optimal schedule if the intermediate machine is non-bottleneck, i.e. it can process any number of jobs at the same time. Formally, the intermediate machine (M_2) is non-bottleneck if one of the following three conditions, that originally appeared in [30, 31, 88, 142], is satisfied:

- $\min_j\{p_{1j}\} \geq \max_j\{p_{2j}\}$ or
- $\min_j\{p_{3j}\} \geq \max_j\{p_{2j}\}$ or
- $\min\{p_{1j}, p_{3j}\} \geq p_{2j}, j = 1, \ldots, n.$

Under one of these conditions, the application of Johnson's rule to the problem with two machines, and processing times $p_{1j}+p_{2j}$ and $p_{2j}+p_{3j}$ on the first and the second machines, respectively, gives an optimal schedule for $F3||C_{max}$. In the general case, however, it turns out that $F3||C_{max}$ is strongly NP-hard [61].

Gonzalez and Sahni [68] considered the preemptive version of the flow shop problem and showed that the problem $F2|pmtn|C_{max}$ can be solved in polynomial time using Johnson's rule. Unfortunately, the problem becomes NP-hard in the strong sense for three machines $F3|pmtn|C_{max}$. Now, if the jobs are not all available simultaneously but have different release times, r_j, the two-machine flow shop problem turns out to be NP-hard in the strong sense, i.e. problem $F2|r_j|C_{max}$ [98]. This result holds for the preemptive version $F2|r_j, pmtn|C_{max}$ [35]. Another sub-case of flow shop scheduling is that with no-wait

constraints where a job once finished on one machine must immediately be started on the next machine. The two-machine case, i.e. problem $F2|no-wait|C_{max}$, can be solved to optimality in $O(nlogn)$-time by an algorithm of Gilmore and Gomory [63]. However, $Fm|no-wait|C_{max}$ is NP-hard for fixed $m \geq 3$ [134]. In the case of no intermediate storage between successive machines, when a machine finishes with the processing of a job, that job cannot proceed to the next machine if that machine is busy; the job must remain on the first machine occupying it and blocking the next job. This phenomenon is referred to as blocking. With two machines $F2|block|Any$ is equivalent to $F2|no-wait|Any$ [51]. When the definition of precedence constraints $J_j \prec J_k$ specifies that job J_j must complete its processing on each machine before job J_k may start processing on that machine then the two-machine flow shop problem with tree or series-parallel precedence constraints and makespan minimization is solvable in polynomial time [110, 112, 138]. For the case of general precedence constraints, Monma [111] showed that $F2|prec|C_{max}$ is NP-hard in the strong sense. In some recent works, Baptiste and Timkovsky [9] and Averbakh et al. [5] showed that the problems $F2|prec, r_j, p_{ij} = 1|\sum C_j$ and $Fm|p_{ij} = 1, intree|\sum C_j$, respectively, are polynomial.

Since the publication of the seminal paper of Johnson [88], the flow shop problem has become one of the most intensively investigated topics in scheduling theory. This interest is not only motivated by its practical relevance, but also by its deceptive simplicity and challenging hardness. Though, the flow shop problem is still considered as a very hard nut to crack. Indeed, up to the mid of the 1990s, the best available branch and bound algorithms experience difficulty in solving instances with 15 jobs and 4 machines. It is interesting to observe that at about the same time, instances of the celebrated travelling salesman problem with few hundreds of vertices could be solved quite routinely. In the following two sections, we have tried to incorporate the main exact and approximation methods proposed for the permutation flow shops.

Exact methods One of the early branch and bound procedures used to find an optimal permutation schedule is described by Ignall et al. [81] and, independently by Lomnicki [104]. These papers considered the three-machine case. Since then, several additional branch and bound algorithms have been published, a review can be found in [94]. Carlier et al. [34] proposed two branch and bound algorithms in which they adapted the disjunctive graph model to flow shop problems. In the branching rule used, the jobs get fixed from the beginning and the end of the schedule. Their second algorithm can solve up to 10-job 100-machine, 50-job 10-machine and 20-job 20-machine instances. Ladhari et al. [94] presented another branch and bound algorithm with tighter lower bounds. Their algorithm produced optimal solutions for a number of well-known instances with up to 2000 operations ($n = 200$ and $m = 10$), and randomly generated instances with up to 8000 operations ($n = 2000$ and $m = 4$).

In addition to branch and bound algorithms, several authors have proposed mathematical formulations including Wagner [159], Wilson [161], Manne [107] and Liao and You [99]. The first two models use the classic assignment problem for the assignment of jobs to various sequence positions, whereas the last two models use pairs of dichotomous constraints to assign jobs to sequence positions. In comparing the

number of binary variables required in each model, Pan [124] concluded that, for the permutation flow shop problems, Manne's model is the best Mixed Integer Linear Programming (MILP) formulation, Wagner's model is second best MILP formulation and Wilson's model trails behind Wagner's formulation. However, Tseng et al. [156] empirically evaluated the computational efforts required by the above four MILP models to solve the same set of permutation flow shop problems. This study revealed the following ranking: Wagner's model is the best MILP formulation, Wilson's model is second best MILP formulation and Manne's model trails behind Liao–You's formulation. The above rankings are different from those reported by Pan [124], which implies that the use of the binary number of variables may not be sufficient to assess the computational complexity of various MILP formulations.

Approximation methods The complexity of the flowshop scheduling problem renders exact methods impractical for instances of more than a few jobs and/or machines. This is the main reason for the various approximation methods proposed in the literature, including heuristics and metaheuristics, some of which will be explained in this Section.

A non-exhaustive list of heuristics includes Page [122], Palmer [123], Campbell et al. [32], Gupta [73], Dannenbring [45], Nawaz et al. [116] and Hundal et al. [79]. Reviews and classifications of $Fm|prmu|C_{max}$ heuristics can be found in [56, 132, 135]. Palmer [123] was the first to introduce the concept of a job slope index. The slope index gives a large value to jobs that tend to proceed from shorter to longer processing times in the sequence of operations. The jobs are then scheduled by non-increasing order of this index, which leads to a computational complexity of $O(nm + nlogn)$. This idea has been used in later papers, for example Gupta [73] proposed a modification of Palmer's slope index which exploited some similarities between scheduling and sorting problems. Hundal et al. [79] proposed an extension to Palmer's heuristic by exploiting the fact that when m is an odd number, Palmer's slope index returns the value 0 for the machine $(m + 1)/2$ and consequently ignores it. In order to overcome this, two other sets of slope indices are computed. Then, two more schedules are produced and the best one is selected. Both heuristics by Campbell et al. [32] and Dannenbring [45] used Johnson's rule. The former generates a set of $m - 1$ two-machine problems by splitting the m machines into two groups. Then Johnson's two-machine algorithm is applied to find the $m - 1$ schedules, followed by selecting the best one. The second heuristic constructs only one two-machine problem, but the processing times for the jobs reflect the behavior of Palmer's slope index. In 1983, Nawaz et al. [116] proposed a heuristic commonly known as NEH. The study of Park et al. [126] led to the conclusion that NEH significantly outperformed all 15 examined heuristics. Later, the authors of [56, 74, 135, 143, 157] acknowledged or confirmed the superiority of NEH. This heuristic considers lengthy jobs early in the sequence and then carries out insertions as shown in Algorithm 1. The NEH heuristic is not only the most powerful heuristic to date, but also very simple to implement. It has a total time complexity of $O(n^3m)$. Taillard [143] reduced this complexity to $O(n^2m)$ by accelerating the calculation of the makespans of the partial sequences. This improved version, denoted hereafter by NEH-T, will be used in Section 3.7.

Algorithm 1 NEH heuristic for $Fm|prmu|C_{max}$ [116].

Input: Data of the flow shop problem.

1: Order the n jobs by non-increasing sums of processing times on the machines;
2: Take the first two jobs and schedule them in order to minimize the makespan as if there were only two jobs;
3: **for** $k = 3, \ldots, n$ **do**
4: Insert the k^{th} job into each of the k possible positions of the partial sequence found in the previous step and keep the best one defining the lowest C_{max};
5: **end for**

Output: The final sequence and the corresponding C_{max}.

However, even the NEH heuristic fails to reach solutions even within 7% from the optimum, in some difficult problem instances. Thus, it became evident that new methods should be followed for larger instances, and academic interest switched to artificial intelligence optimization methods known as metaheuristics. The current benchmark for Simulated Annealing (SA) algorithms is the implementation of Osman et al. [120]. Alternative SA implementations exhibiting robust performance with respect to temperature cooling are presented by Ishibuchi et al. [84] who show that the solution quality is comparable to that of Osman et al. [120]. Taillard [143] developed a Tabu Search (TS) algorithm that uses the NEH heuristic for the initial schedule. The TS given by Nowicki et al. [118] improved upon the majority of the benchmark makespan values, and the improved values were found much faster than in Taillard [143]. Zobolas et al. [162] presented a hybridization of a genetic algorithm and a variable neighbourhood search. This algorithm can solve instances of size up to 200 jobs and 20 machines and performs near optimally across all instances of the Taillard suite. This list of metaheuristics is not exhaustive, but a detailed literature review about the approximation methods proposed for $Fm|prmu|C_{max}$ can be found in [51].

1.2 Graph theory

We begin with a summary of the basic graph-theoretic terminology used in this thesis. Only those concepts that are of direct relevance to this work are mentioned here. For more details, the reader may refer to [26, 64] or any other graph theory text.

A graph consists of a set of vertices and a set of edges. Each edge is defined by a pair of distinct vertices referred to as the endpoints of the edge. If the pairs are unordered, the graph is undirected and if they are ordered, the graph is directed. Undirected graphs are denoted by $G = (V, E)$ where V is the set of vertices and E is the set of edges. A simple graph is a graph without loops and multiple edges. A path of length k is a sequence of edges $e_1, e_2, \ldots, e_k$ such that for all i, $2 \leq i \leq k$, e_i shares one of its endpoints with e_{i-1} and the other with e_{i+1}. A cycle is a closed path, that is, a path that begins and ends on the same vertex. The graph induced by a set of vertices $U \subset V$ is the subgraph $G[U] = (U, F)$ where F contains

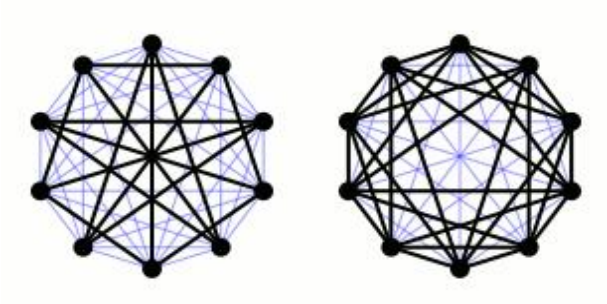


Figure 1.2.1: The Petersen graph (on the left) and its complement graph (on the right) and the graph with grey edges is the complete graph.

exactly those edges in E that join vertices in U. A partial graph of a graph $G = (V, E)$ is the subgraph $G' = (V, E')$ such that $E' \subset E$. A graph with n vertices in which every two vertices are adjacent is a complete graph K_n. The complement of a graph $G = (V, E)$, is the graph $\overline{G} = (V, \overline{E})$, with the same vertex set but whose edge set consists of the edges not present in G. The graph sum $G + \overline{G}$ on a n-vertex graph G is therefore the complete graph K_n, as illustrated in Figure 1.2.1.

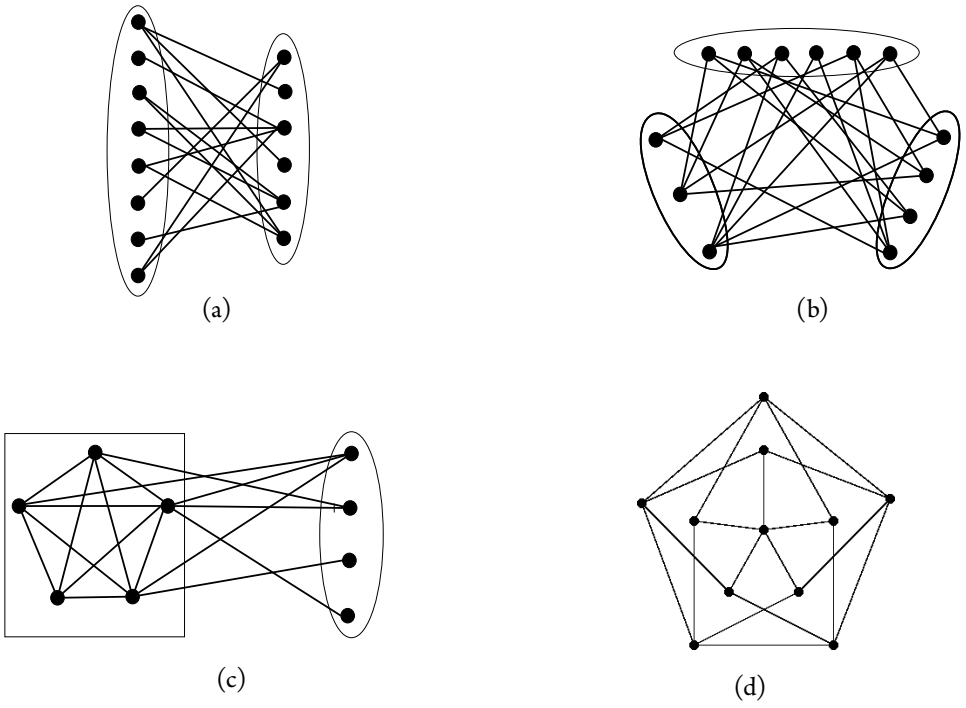


Figure 1.2.2: (a) bipartite graph. (b) 3-partite graph. (c) split graph. (d) the Grötzsch graph which is a triangle-free graph

We describe some graph classes, as pertinent to this thesis, in what follows. A *bipartite graph* is one whose vertex set can be partitioned into two disjoint sets in such a way that each edge joins a vertex of

the first set with a vertex of the second set, whereas if the vertex set can be partitioned into three disjoint sets such that no two vertices within the same set are adjacent, then the graph is said to be *3-partite*. A graph is a *split graph* if it can be partitioned in an independent set (a subset of vertices that share no edges) and a clique (a subset of vertices such that its induced subgraph is complete). A *triangle-free graph* is a graph in which no three vertices form a triangle of edges, observe that this class includes the class of bipartite graphs (see Figure 1.2.2).

Finding an independent set of maximum cardinality is one of the most important optimization problems (denoted MIS)[26]. This problem will be encountered in several stages of this thesis. Since MIS problem is NP-hard, a simple greedy algorithm, called GMIN [136], will be used. This algorithm is summarized in Algorithm 2.

Algorithm 2 GMIN: greedy algorithm for the maximum independent set problem

Input: Graph $G = (V, E)$.

1: $I = \emptyset$;
2: **while** $V \neq \emptyset$ **do**
3: Choose a vertex, say v_i, in G of minimum degree;
4: $I = I \cup \{v_i\}$;
5: Remove v_i and its neighbors from G;
6: **end while**

Output: I.

We will also deal with the problem of finding a maximum matching in a number of algorithms that will be presented throughout this thesis. A matching in a graph is a set of edges chosen in such a way that no two edges share a vertex. A maximum matching is a matching that contains the largest possible number of edges. There are several algorithms for finding a maximum matching, the complexity of the most efficient by Kariv and Even [91] being $O(n^{2.5})$. Matching problems are often concerned with bipartite graphs. For this class of graphs, we apply Hopcroft–Karp algorithm [77].

Now, we discuss briefly the Minimum Path Cover (MPC) problem; a problem that is used in Section 3.2 in the study of the complexity of a restricted case of our problem.

Minimum path cover problem A path cover of a graph is a family of vertex-disjoint paths that covers all the vertices of this graph. The MPC problem is to find a path cover of minimum cardinality. This problem has been well studied in graph theory and finds numerous practical applications in database designing, networking, establishing ring protocol, VLSI designing, code optimization [24], and mapping parallel programs into parallel architectures [113, 128]. Clearly this problem generalizes the Hamiltonian path problem [60]. The MPC problem is known to be NP-hard even when the input is restricted to interesting special graph classes including planar graphs, bipartite graphs, chordal graphs, chordal bipartite graphs and strongly chordal graphs. However, polynomial-time algorithms exist for trees, block graphs, interval graphs, cographs, bipartite distance-hereditary graphs, bipartite permutation graphs,

cocomparability graphs and distance-hereditary graphs (see [4, 41, 80] for more details about the complexity). Some upper bounds on the minimum number of paths in a path cover has been proposed in the literature (see, e.g. [25, 106, 117, 119]). These upper bounds are related to other graphical invariants. To our knowledge, there is no branch and bound algorithm nor a mathematical model for the MPC problem on general graphs.

1.3 Scheduling with conflict/agreement graph

In scheduling applications, many additional constraints may occur. In this thesis, we are interested in a variant that arises when some jobs may conflict with each other, in which case we cannot schedule them at the same time on different machines. We study the problem when the conflict constraints are given by a simple and undirected graph $G = (V, E)$, called the conflict graph. Each vertex in G represents a job and jobs that are represented by adjacent vertices in G are conflicting and cannot be processed simultaneously on different machines. The graph $\overline{G} = (V, \overline{E})$, called the agreement graph, denotes the complement of the conflict graph $G = (V, E)$ unless mentioned otherwise. It is clear that the problem of Scheduling with Conflict Graph (SCG) and the problem of Scheduling with Agreement Graph (SAG) are polynomially equivalent.

The SCG problem models important practical situations in scheduling under resource constraints when the resources are non-sharable. In this later setting, we have certain resources which can only be made available to one job at a time. Two jobs which demand such a resource cannot be processed simultaneously. For example, processes in a distributed operating system that need access to the same file or device are in conflict. We return to the problem of scheduling under resource constraints in Section 1.4. Applications exist also in exam timetabling, see e.g. [10, 12]. Other applications will be given in Section 1.3.2.

Following the three field classification $\alpha/\beta/\gamma$ of Graham et al. [69], we denote the scheduling problem we are studying by $\alpha|ConfG = (V, E)|\gamma$, where $ConfG = (V, E)$ indicates the presence of a conflict graph $G = (V, E)$ over the jobs. As far as we know, scheduling with conflict/agreement graphs has been studied only on identical machines. In the following two sections, we survey the existing literature about the SCG problem on identical machines. In Section 1.3.1, we will be concerned with the case of arbitrary processing times, and continue to the restricted case of unit-time jobs in Section 1.3.2.

1.3.1 Scheduling with conflict/agreement graph on identical machines

We focus in this section on scheduling with conflict/agreement graph on identical machines with the objective of minimizing the makespan. This problem is NP-hard for two identical machines even if the conflict graph is an independent set (the problem becomes $P2||C_{max}$) [60]. Even et al. [53] provided

an exact algorithm for the SCG problem on two identical machines and jobs sizes in $\{1, 2\}$. They also showed that for jobs sizes in $\{1, 2, 3, 4\}$ the problem is APX-hard and they left the question of the exact status of the case of jobs sizes in $\{1, 2, 3\}$ as an open problem. For this later case, they gave an algorithm with a $4/3$-approximation. Then, they considered the on-line version of the SCG problem where each job has a release time that specifies when the job arrives, and scheduling decisions at time t can only depend on jobs that have arrived before or at time t. Hence, the conflict graph is revealed as the jobs arrive. For this version, they presented lower and upper bounds for the competitive ratio of deterministic on-line algorithms. Specifically, they proved a lower bound of $2 - 1/m$ on the competitive ratio of any deterministic on-line algorithm for m machines and unit-time jobs, and an upper bound of 2 when the algorithm is not restricted computationally. For three machines, they showed that an efficient greedy algorithm achieves this bound. For two machines, they presented an algorithm that achieves a competitive ratio of $2 - 1/7$ when the number of jobs is known in advance to the algorithm. Other authors have also considered the on-line version of the SCG problem including Irani and leung (1996)[82], Motwani et al. [114] and Irani and leung (2003) [83]. Indeed, Motwani et al. [114] considered a processing environment with an unbounded number of machines. They showed that when jobs arrive at integral times and have one unit of processing time, the competitive ratio for the makespan is 2. However, when jobs arrive at arbitrary times and have arbitrary processing times, they gave an upper bound of 3 on the competitive ratio for the makespan using a model which allows for preemption. Irani and leung [83] discussed the version involving unit-time jobs and an unbounded number of machines with the objective of minimizing the maximum response time (i.e. job completion time minus its release time) but they focused on two special classes of graphs: bipartite and interval graphs. These two classes model two applications in traffic intersection control and session scheduling in high speed local area networks with spatial reuse.

Now, we return to the offline version of the SCG problem in which we have a complete knowledge of the jobs including their processing times, conflict graph and release times. Bendraouche and Boudhar [11] considered the open problem given in [53] about the complexity of the SCG problem on two identical machines and jobs sizes in $\{1, 2, 3\}$. They proved that this problem is NP-hard in the strong sense even when restricted to complements of bipartite graphs. Then, they found that if the jobs have release times in $\{0, r\}$, with r arbitrary, the problem with jobs sizes in $\{1, 2\}$ becomes NP-hard in the strong sense even for complements of bipartite graphs. They also proposed an $O(n^3)$-time algorithm when the agreement graph is bipartite $\overline{G} = (S_1, S_2; \overline{E})$, where S_1 contains unit-time jobs and the processing times of the jobs of S_2 are arbitrary. Finally, they proposed lower bounds and heuristics derived from a list algorithm for the general SAG problem on m identical machines along with an experimental study.

The same authors presented in [13] a work in which they closed definitely the complexity status of the SAG problem on two identical machines with two fixed processing times. Indeed, they proved that for two machines and jobs sizes in $\{a, 2a + b\}$ ($a \geq 1$ and ($b \geq 1$ or $-a < b < 0$)), the SAG problem is strongly NP-hard when restricted to a bipartite agreement graph. Furthermore, they showed that the SAG problem on identical machines is polynomially equivalent to a special case of the resource-

constrained scheduling problem. New complexity results of this latter have been derived, in particular they have established its complexity for two machines with two fixed processing times.

Table 1.3.1: Previous complexity results of the SCG/SAG on identical machines.

SCG/SAG problem	Complexity	Reference
$m = 2$, $G = (V, E)$, $p_j \in \{1, 2\}$	polynomial	[53]
$m = 2$, $\overline{G}$ bipartite graph, $p_j \in \{1, 2, 3\}$	NP-hard	[11]
$m = 2$, $\overline{G}$ bipartite graph, $r_j \in \{0, r\}$, $p_j \in \{1, 2\}$	NP-hard	[11]
$m = 2$, $\overline{G} = (S_1, S_2; \overline{E})$ a bipartite graph, $p_j = 1$ for $J_j \in S_1$	polynomial	[11]
$m = 2$, $\overline{G}$ bipartite graph, $p_j \in \{a, 2a + b\}$ with $a \geq 1$ and $(b \geq 1$ or $-a < b < 0)$	NP-hard	[13]
$m = 2$, $\overline{G} = (A \cup B; \overline{E})$, $G[A]$ independent set, $\lvert A\rvert = 2a$ $G[B]$ has a fixed structure, $\lvert B\rvert = 2a - \beta$, $p_j \in \{1, 2, 3\}$	NP-hard	[12]
$m = 2$, $\overline{G} = (A \cup B; \overline{E})$, $G[A]$ independent set, $\lvert A\rvert = 2a$ $G[B]$ has a fixed structure, $\lvert B\rvert = 2a - \beta$, $r_j \in \{0, r\}$, $p_j \in \{1, 2\}$	NP-hard	[12]
$\overline{G} = (C, S; \overline{E})$ split graph, $p_j = 1$ for $J_j \in C$	polynomial	[12]
$m = n$, $\overline{G}$ complement of bipartite graph, $r_j \in \{0, 1, 2\}$, $p_j = 1$	NP-hard	[12]
$m = n$, $\overline{G}$ complement of bipartite graph, r_j odd, $p_j = 1$	polynomial	[12]
$m = n$, $\overline{G}$ complement of bipartite graph r_j even, $p_j = 1$	polynomial	[12]
$m = n$, $\overline{G}$ complement of bipartite graph $r_j \in \{0, r\}$, $p_j = p$	polynomial	[12]

Recently, Bendraouche and Boudhar extended in [12] the two NP-hardness results that they presented in [11] to another graph structures. They also proposed an $O(n^3)$-time algorithm for the SAG problem on m identical machines when the agreement graph is a split graph $\overline{G} = (C, S; \overline{E})$, where the clique C contains unit-time jobs and the processing times of the jobs of the independent set S are arbitrary. Further complexity results were provided for the case of complements of bipartite agreement graphs and $m = n$, followed by some approximation results for the SAG problem. Table 1.3.1 summarizes the complexity results of the SCG/SAG problem on identical machines.

1.3.2 Mutual exclusion scheduling

Baker and Coffman [6] have called Mutual Exclusion Scheduling (shortly MES) the problem of scheduling with conflict graph on m identical machines, under minimization of the makespan, such that each job requires one unit of processing times. This problem corresponds exactly to a minimum colouring of the conflict graph G such that each colour appears at most m times.

Table 1.3.2: Previous complexity results of the MES problem.

MES problem	Complexity	Reference
Line graphs	polynomial	[2]
Chordal graphs, fixed $m \geq 3$	NP-hard	[42]
Split graphs	polynomial	[22, 105]
Complements of interval graphs	polynomial	[22, 105]
Cographs, m fixed	polynomial	[105]
Complements of comparability graphs, fixed $m \geq 3$	NP-hard	[105]
Complements of line-graphs, fixed $m \geq 3$	NP-hard	[38]
Bipartite graphs, m fixed	polynomial	[22, 76]
Bipartite graphs and cographs	NP-hard	[22]
Interval graphs, fixed $m \geq 4$	NP-hard	[22]
Forests and trees	polynomial	[6, 87]
Collection of disjoint cliques	polynomial	[46]
Complements of strongly chordal graphs	polynomial	[44]
Permutation and comparability graphs, fixed $m \geq 6$	NP-hard	[86]
Bounded treewidth graphs	polynomial	[21]
Complements of triangulated graphs	polynomial	[89]
Threshold, proper circular-arc and proper interval graphs	polynomial	[58]
Interval graphs and circular-arc graphs, $m = 3$	open	-
Tolerance graphs, fixed $m \geq 3$	NP-hard	[58]
Proper tolerance graphs, fixed $m \geq 3$	open	-

The MES problem arises in load balancing the parallel solution of partial differential equations (pde's) by domain decomposition [6, 14]. The domain for the pde's is decomposed into regions where each region corresponds to a subcomputation. The decomposition may be chosen so that the predicted subcomputation times are approximately equal. The subcomputations are scheduled on m machines so that subcomputations corresponding to regions that touch at even one point are not performed simultaneously. Gardi [58] presented anther application related to interval graphs met during the development of the software Bamboo for workforce planning, edited by the firm Experian-Prologia SAS [8]. In this application, we have a set of daily jobs to assign to employees, each job has a starting and an ending dates. An employee is able to execute correctly a set of jobs if they do not overlap during the day. For several reasons (regulation of work, security, maintenance of machines), an employee must not execute more than m jobs in a day (generally $m \leq 5$). Then, the question is how to execute all the jobs with a minimum number of employees? Obviously, a planning describing which jobs have to be assigned to each employee is required. We construct a graph in which each vertex represents a job, and two vertices are adjacent if their corresponding jobs overlap during the day. Then, the problem consists of finding a minimum colouring of the obtained interval graph such that each colour appears at most m times, which corresponds exactly to the MES problem for interval graphs. When the planning is cyclic (the same jobs recur each day and some of them spread out over two consecutive days), we obtain the same problem but for circular-arc graphs. When there are not enough employees to execute all the jobs (e.g., because some employees are absent), it is interesting to allow overlaps between certain jobs during the assignment. In this case, the problem relates to tolerance graphs. The MES problem can also be found

in constructing school course timetables [93], scheduling in communication systems [82], assigning operations to processors, where the operations are given in a flow graph [22]. Other application are mentioned in traffic intersection control, frequency assignment in cellular networks, and session management in local area networks [75].

Finding a minimum colouring of a graph is a celebrated NP-hard problem [92], then MES is also NP-hard for arbitrary conflict graph and fixed $m \geq 3$. For $m = 2$, the problem is equivalent to the maximum matching problem in the agreement graph and, therefore, can be solved in polynomial time. For $m = 3$, the complexity of MES is the same as that of the NP-hard partition into triangles in the agreement graph. However, there are some special graph classes on which this problem is solvable in polynomial time. Table 1.3.2 summarizes the previous complexity results.

1.4 Scheduling under resource constraints

Since conflicts between jobs may be generated by shared resources, we will be considering in this section the resource-constrained scheduling problem. In this problem, besides machines, the jobs may require for their processing some additional scarce resources. A resource is called renewable, if only its total usage, i.e. temporary availability at every moment, is constrained. A resource is called non-renewable, if only its total consumption, i.e. integral availability up to any given moment, is constrained (in other words this resource once used by some job cannot be assigned to any other job), for instance money and fuel. In the viewpoint of resource divisibility, we distinguish two resource categories: discrete (i.e. discretely-divisible) and continuous (i.e. continuously divisible) resources. In other words, by a discrete resource we will understand a resource which can be allocated to jobs in discrete amounts from a given finite set of possible allocations, which in particular may consist of one element only. Continuous resources, on the other hand, can be allocated in arbitrary, a priori unknown, amounts from given intervals [18].

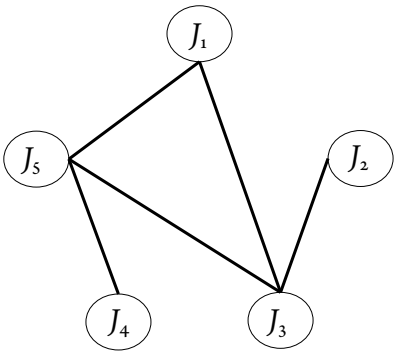
In what follows, we focus on the open shop and the flow shop scheduling problems under resource constraints. By J_{ij} we denote the operation of job J_j on machine M_i, $i = 1, \ldots, m$, $j = 1, \ldots, n$. Let us consider the following setting in which we have a set $\{R_s, s = 1, \ldots, r\}$ of r additional renewable resources, each resource R_s is discretely divisible and its usage at any time is constrained by an available amount $|R_s|$. Each operation J_{ij} requires $R_s(J_{ij})$ units of resource R_s. In this case, the conflicts arise between operations, in general, because the operations of a job may have different requirements of a given resource. For example, consider two jobs J_l and J_k that have to be processed on two machines. Only one resource R_s is required for the execution of the jobs such that: $|R_s| = 1$, $R_s(J_{1l}) = R_s(J_{1k}) = R_s(J_{2k}) = 1$ and $R_s(J_{2l}) = 0$. It follows that J_{1l} and J_{2k} are in conflict but J_{2l} and J_{1k} are not. Thus, this problem cannot be modelled by a conflict graph over the jobs.

If we assume that the operations of each job have the same requirement of each resource, then a conflict

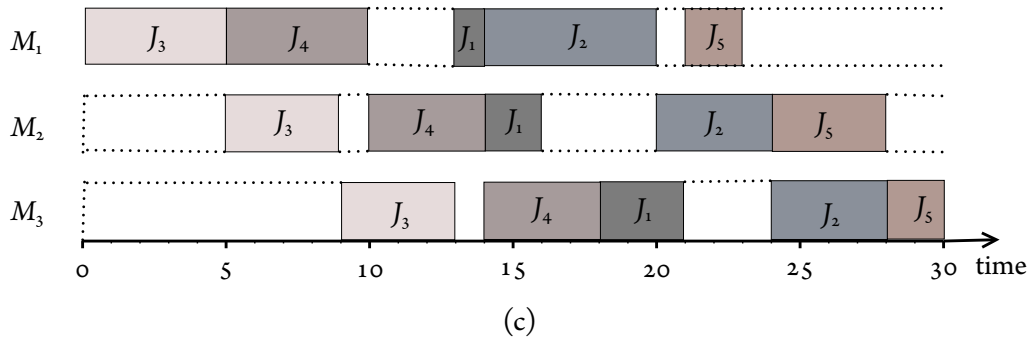
arises between a subset of jobs if the total requirement of at least one resource exceeds its capacity. In general, this problem can be modelled by a conflict hypergraph. Indeed, consider the following example of 3 jobs $\{J_1, J_2, J_3\}$ to be processed on three machines. Suppose that we have one renewable resource R_s such that $|R_s| = 2$ and $R_s(J_{ij}) = 1; i = \overline{1,3}, j = \overline{1,3}$. Hence, J_1 and J_2 are not in conflict. The same for J_1 and J_3 and also for J_2 and J_3. The conflict graph is then an independent set, which means that all the jobs can be processed simultaneously. This is not possible since the total requirement of all the jobs exceeds the resource capacity. Therefore, this case requires a conflict hypergraph. However, in special cases a conflict graph suffices, e.g. when $m = 2$ (examples will be given in the beginning of Sections 2.2.2 and 3.2) or when $|R_s| = 1$. This later case will be discussed in Sections 2.1 and 3.1. Now, if $m = 2$ and $\max_{k,l;k\neq l}\{R_s(J_{1k}) + R_s(J_{2l})\} \leq |R_s|, \forall R_s$, i.e. the available amount of each resource is sufficient to process the operations on the two machines, then there are no conflicts between the jobs and the conflict graph associated is an independent set. An example of a setting that can be modelled by a conflict graph is given in Figure 1.4.1, where 5 jobs have to be processed on a three-machine flow shop. Three resources are required for the execution of the jobs, such that $|R_s| = 1, s = 1, \ldots, 3$.

J_1	J_2	J_3	J_4	J_5
$\{R_1\}$	$\{R_2\}$	$\{R_1, R_2\}$	$\{R_3\}$	$\{R_1, R_3\}$

(a)



(b)



(c)

Figure 1.4.1: (a) resource requirements of each job. (b) associated conflict graph. (c) a feasible schedule.

Blazewicz et al.[15] expanded the three field classification, $\alpha/\beta/\gamma$, to cover the resource constrained scheduling problems. Parameter β denotes the job and resource characteristics. The presence of additional resources is specified by $res\lambda\sigma\delta$, where $\lambda, \sigma, \delta \in \{., k\}$ represent respectively the number of resource types, resource availabilities and resource requirements. A dot "." indicates that the corre-

sponding parameter can take any integer value, whereas a positive integer k indicates that the number of resource types is equal to k, each resource is available in the amount of k units and the resource requirements of each operation are equal to at most k units. We denote by $res^t\lambda\sigma\delta$ the case in which the operations of each job J_j require the same amount of each resource R_s and we refer to this requirement by $R_s(J_j)$, i.e. $R_s(J_{1j}) = \ldots = R_s(J_{mj}) = R_s(J_j)$, $s = 1, \ldots, r$, $j = 1, \ldots, n$. Observe that the scheduling problem $a/res^t\lambda\sigma\delta/\gamma$ is a subproblem of $a/res\lambda\sigma\delta/\gamma$.

1.5 Conclusion

Not only are scheduling problems with conflict graphs relevant from a practical point of view, they also pose challenging research questions. Some of these questions are resolved for the case of identical machines as shown in the state of the art of this chapter. It is of interest then to consider the case of dedicated machines. This thesis is concerned with two models of dedicated machines: the open shop problem which is addressed in Chapter 2 and the flow shop problem which is the subject of Chapter 3.

All things are difficult before they are easy

Thomas Fuller

2

Open shop problem

We study in this chapter the open shop scheduling problem with conflict graph. The inputs consist of a finite set $\{J_j, j = 1, \ldots, n\}$ of n jobs that has to be processed on a given set $\{M_i, i = 1, \ldots, m\}$ of m machines and a simple and undirected graph $G = (V, E)$ over the jobs, called the conflict graph. Each job J_j consists of m operations J_{ij} $(i = 1, \ldots, m)$, where J_{ij} has to be processed on machine M_i for p_{ij} time units. The processing times p_{ij} are nonnegative numbers. The operations of each job can be processed in any order, but only one at a time. Each machine can process at most one operation at a time. Each vertex in G represents a job and jobs that are represented by adjacent vertices in G are conflicting and cannot be processed simultaneously on different machines. The objective is to find a schedule that minimizes the maximum completion time also called the makespan. This problem is referred to as Open Shop scheduling with Conflict graph (OSC for short) and can be solved in polynomial time if G is a clique (all the operations are processed in disjoint time intervals and $C_{max} = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$), or if G is an independent set and the jobs are processed on two machines (the problem is the basic two-machine open shop problem and can be solved using the Longest Alternate Processing Time first (LAPT) rule proposed in [127] or an algorithm proposed earlier in [67], see Section 1.1.2).

Our primary objective in this chapter may be considered as a quest for the answer to the questions: *can we give an efficient algorithm that produces an optimal schedule for the OSC problem?* If the situation is bad for the general case, then, *does it improve for restricted cases?* The answers to these questions are the chief content of the first two sections of this chapter. We start in Section 2.1 by studying the relation between the OSC problem and the open shop problem under resource constraints because conflicts

between jobs may arise from shared resources (see Section 1.4). The previous complexity results of this later problem are then presented. In Section 2.2, we discuss the complexity of the OSC problem. We first prove some NP-hardness results regarding the two-machine and three-machine versions. Then, we present polynomial-time solvable cases, for which we devise simple and efficient algorithms. Furthermore, new complexity results of the open shop problem under resource constraints are derived.

Discovering that a problem is NP-hard is usually just the beginning of work on that problem and the question that arises: *what can we do when faced with NP-hard problems?* A common approach is to look for good approximate solutions. Insofar as the OSC problem is concerned, we present in Section 2.4 a two-phase heuristic approach. This approach is based on the construction of matchings in bipartite graphs as well as on a specific insertion technique combined with beam search. The quality of the obtained solutions is compared to lower bounds that will be discussed in Section 2.3. Finally, extensive computational experiments performed on a wide range of test problems are provided in Section 2.5 to measure the performance of the proposed approach.

2.1 Related works

In this section, we first prove that the OSC problem is polynomially equivalent to a special case of the open shop problem under resource constraints. Then, we survey the existing literature concerning the complexity of this later problem. The reader may refer to Section 1.4 for details about the differences between shop scheduling with conflict graph and shop scheduling under resource constraints.

Proposition 2.1 ([148]). *$O|ConfG = (V, E)|C_{max}$ is polynomially equivalent to $O|res^t.11|C_{max}$.*

Proof. Given any instance of the first problem, we construct an instance of the second problem as follows. The set of jobs is the same for the two problems. For each edge $e_s = \{J_k, J_l\} \in E$, we introduce a resource R_s with availability $|R_s| = 1$. The requirements of this resource are such that $R_s(J_k) = R_s(J_l) = 1$ and $R_s(J_j) = 0$, for all $j \notin \{k, l\}$.

Conversely, for a given instance of the second problem, we construct an instance of the first problem as follows. The set of jobs is the same for the two problems. The conflict graph G is defined by the relation: $\{J_k, J_l\} \in E$ if and only if there exists at least a resource R_s such that $R_s(J_k) + R_s(J_l) > 1$.

Thus, two jobs can be executed simultaneously in the OSC problem if and only if they do not share the same resource in $O|res^t.11|C_{max}$ and the theorem follows. □

Since our problem is related to the open shop problem under resource constraints, we focus in the following review of the literature on this problem. Blazewicz et al. [15, 16] presented an overview and

Table 2.1.1: Previous complexity results of the open shop under resource constraints.

Problem	Complexity	Reference
$O2\|res\ldots, p_{ij} = 1\|C_{max}$	$O(n^{2.5})$	[15]
$O2\|res111, chain, p_{ij} = 1\|C_{max}$	strongly NP-hard	[16]
$O2\|res\ldots, pmtn\|C_{max}$	$O(n^3)$	[90]
$O2\|res1..\|C_{max}$	$O(n^3)$	[90]
$O2\|res211\|C_{max}$	weakly NP-hard	[90]
$O2\|res.11\|C_{max}$	strongly NP-hard	[90]
$O3\|res1.., p_{ij} = 1\|C_{max}$	strongly NP-hard	[16]
$O3\|res.11, p_{ij} = 1\|C_{max}$	strongly NP-hard	[16]

complexity classification for the case of renewable resources (Table 2.1.1). De Werra et al. studied in [47–49] the preemptive version of the m-machine open shop problem with renewable and nonrenewable resources, they formulated the problem in terms of edge coloring in a bipartite multigraph and they exhibited a polynomial algorithm for the case where the curve of the resource availability is of staircase type. Jurisch and Kubiak [90] considered the two-machine open shop with renewable resources. They first showed that the preemptive case can be solved in $O(n^3)$ through a reduction to the max-flow problem, then they converted the preemptive solution into a non-preemptive schedule when the number of resources is limited to one. They also proved the NP-hardness of two particular cases (Table 2.1.1). Tautenhahn and Woeginger [146] studied the open shop problem with unit-time operations when the availability of the resources changes over time. They proposed polynomial algorithms for several objective functions when the number of machines, the number of resources and the demand for every resource are bounded by a constant. They also proved the NP-hardness of most of the remaining cases. Oulamara et al. [121] considered the two-machine open shop problem in which the processing time of an operation is decomposed into a preparation part and an execution part. The preparation period should precede the execution period and requires renewable resources. The authors proved the NP-hardness of different versions of the problem and they derived a well solvable case. Table 2.1.1 summarizes all the complexity results of the open shop problem with renewable resources.

2.2 Complexity issues

As we have seen in Section 1.1.2, the basic open shop problem is solvable in $O(n)$-time for two machines and it becomes NP-hard as the number of machines increases to 3 [67]. A natural question then is: *will the open shop problem with $m = 2$ remain polynomial when we introduce the conflict graph?* unfortunately, it will not. Indeed, we show in Section 2.2.1 that the two-machine OSC problem with $p_{ij} \in \{1, 2, 3\}$ is NP-hard in the strong sense even when we restrict our attention to complements of bipartite graphs. We also consider the case of three machines and show that even for unit-time operations the OSC problem is NP-hard in the strong sense. From this later result, we prove the complexity of a variant of a well

known problem in graph theory. Section 2.2.2 covers more positive results, in which we present efficient algorithms for restricted cases of three machines with unit-time operations and two machines. We also prove that preemptions result in an efficient algorithm for two machines. Furthermore, new complexity results of the open shop problem under resource constraints are derived throughout these two sections.

Since $O|ConfG = (V, E)|C_{max}$ is polynomially equivalent to $O|res^t.11|C_{max}$, we start by analysing the results of Table 2.1.1. Jurisch and Kubiak [90] proved that $O2|res.11|C_{max}$ is strongly NP-hard. However, in the instance used in the reduction process, the operations of some jobs have not the same requirement of the resources. Then, this instance is not an instance of $O2|res^t.11|C_{max}$. They also showed that $O2|res211|C_{max}$ is NP-hard in the ordinary sense, similarly the instance used in the reduction process is not an instance of $O2|res^t211|C_{max}$. In these instances, we can find two jobs in which two operations belonging to different machines are in conflict and the two others are not. Thus, the complexity of the OSC problem cannot be derived from these complexity results.

2.2.1 NP-hardness results

In the following theorem, we show that the two-machine OSC problem is NP-hard in the strong sense even when restricted to complements of bipartite graphs and the operations of each job have the same processing times in $\{1, 2, 3\}$.

Theorem 2.2 ([148]). *The two-machine OSC problem with $p_{1j} = p_{2j}$ and $p_{ij} \in \{1, 2, 3\}$ is NP-hard in the strong sense for G being a complement of a bipartite graph.*

Proof. We prove this result by means of a reduction from the 3-dimensional matching problem (3-DM), which is known to be NP-complete in the strong sense (see for e.g. [60]). It is defined as follows. Given three disjoint sets X, Y and Z each of size q, and a set $M \subseteq X \times Y \times Z$ of triples. Is there a subset $M' \subseteq M$ of q triples such that each element of $X \cup Y \cup Z$ is contained in exactly one of these triples? (this proof is inspired by Theorem 1 of [11]).

Given an arbitrary instance of 3-DM, we build an instance I of the two-machine OSC problem as follows. Let $V = V_M \cup V_Y \cup V_Z \cup V_D \cup V_Q \cup V_R$ be the jobs set such that:

- V_Y, V_Z, V_R: jobs with an operation on each machine requiring one unit of processing time.
- V_M, V_Q: jobs with an operation on each machine requiring two units of processing time.
- V_D: jobs with an operation on each machine requiring three units of processing time.

The jobs of V_M are in correspondence with the elements of M so that each element $(x, y, z) \in M$ have a corresponding job $J(x, y, z)$ of V_M. The jobs of V_Y and V_Z are in correspondence with the elements

of Y and Z, respectively, so that each element $y \in Y$ corresponds to a job J_y of V_Y and each element $z \in Z$ corresponds to a job J_z of V_Z. Therefore, $|V_Y| = |V_Z| = q$. Suppose that $X = \{x_1, \ldots, x_q\}$, then for each i $(i = 1, \ldots, q)$ let $M_i = \{(x, y, z) \in M : x = x_i\}$ so that $\bigcup_{i=1}^{q} M_i = M$ and let V_{M_i} be the subset of V_M corresponding to M_i. For each i $(i = 1, \ldots, q)$, the set V_{M_i} corresponds to a set V_{D_i}, also the set V_{D_i} is associated with a set V_{Q_i} and the set V_{Q_i} corresponds to a set V_{R_i} such that $|V_{D_i}| = |V_{Q_i}| = |V_{R_i}| = |M_i| - 1$. The sets V_D, V_Q and V_R are such that: $V_D = \bigcup_{i=1}^{q} V_{D_i}$, $V_Q = \bigcup_{i=1}^{q} V_{Q_i}$, $V_R = \bigcup_{i=1}^{q} V_{R_i}$. One can see that $|V_D| = |V_Q| = |V_R| = |M| - q$. The graph $\overline{G} = (V, \overline{E})$ is defined as follows: the sets V_Y and V_Z, as well as the sets V_{M_i}, V_{D_i}, V_{Q_i} and V_{R_i} each form an independent set of vertices in $\overline{G}$ for all i $(i = 1, \ldots, q)$. Every job $J(x, y, z)$ of V_M corresponding to the element (x, y, z) of M is adjacent to two jobs J_y and J_z corresponding to y and z, respectively $(J_y \in V_Y$ and $J_z \in V_Z)$. For each i $(i = 1, \ldots, q)$, all the jobs of V_{M_i} are joined to all of the jobs of V_{D_i}. For all i $(i = 1, \ldots, q)$ the jobs of V_{D_i} are in one-to-one correspondence with those of V_{Q_i}, that is each job of V_{D_i} is adjacent only to its corresponding job in V_{Q_i}. Similarly the jobs of V_{Q_i} are in one-to-one correspondence with those of V_{R_i} (see Figure 2.2.1). Note that the graph $\overline{G}$ is bipartite since it can be written as $\overline{G} = (S_1, S_2; \overline{E})$ where $S_1 = V_M \cup V_Q$ and $S_2 = V_Y \cup V_Z \cup V_D \cup V_R$ are the two independent sets forming a partition of the vertex set of $\overline{G}$.

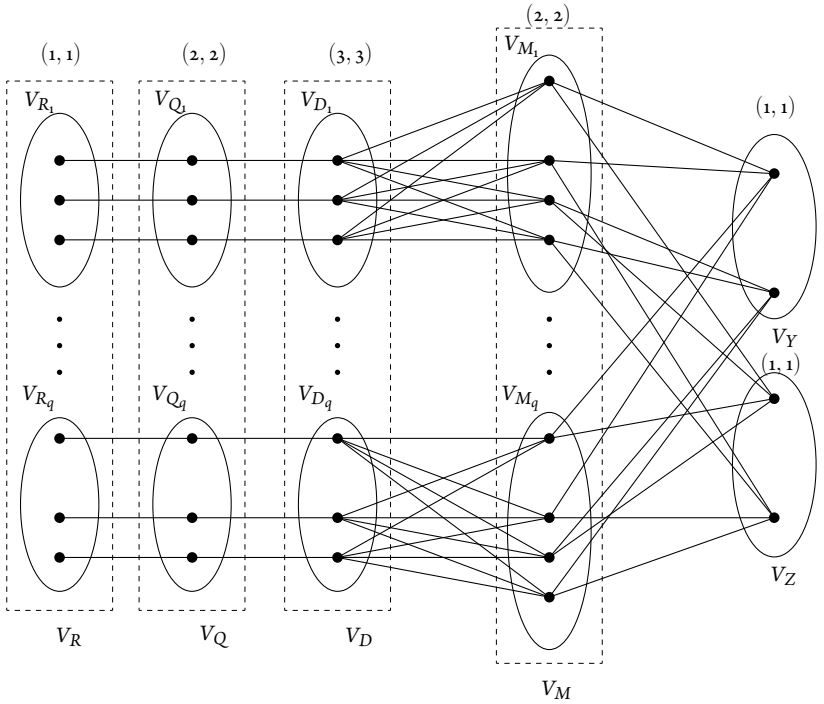


Figure 2.2.1: Instance I of Theorem 2.2: agreement graph $\overline{G} = (S_1, S_2; \overline{E})$.

We show that there exists a feasible schedule σ for I with makespan $C_{max}(\sigma) \leq 8|M| - 4q$ if and only if 3-DM has a solution.

Suppose that 3-DM has a solution M'. We construct a feasible schedule σ for I as follows. Let $V_{M'} =$

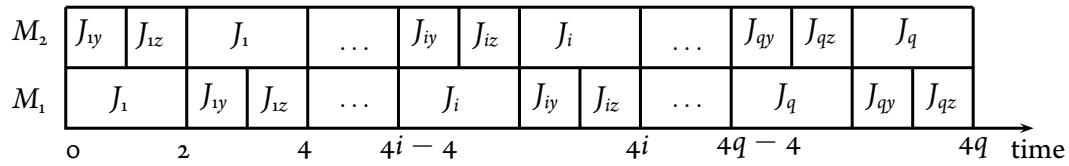


Figure 2.2.2: Instance I of Theorem 2.2: schedule of the jobs of $V_{M'}$ and their corresponding jobs of V_Y and V_Z.

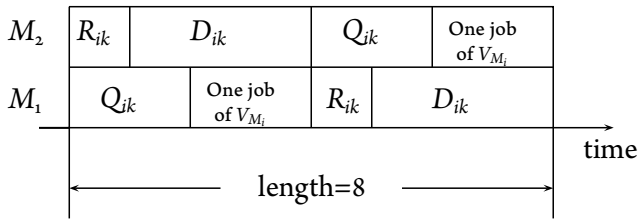


Figure 2.2.3: Instance I of Theorem 2.2: schedule of the jobs of V_i.

$\{J_1, J_2, ..., J_q\}$ be the subset of V_M corresponding to M' such that $J_i \in V_{M_i}$ $(i = 1, \ldots, q)$. We schedule these jobs as well as their corresponding jobs of the sets V_Y and V_Z within the time interval $[0, 4q]$ as indicated in Figure 2.2.2. For each i $(i = 1, \ldots, q)$, let $V_i = V_{D_i} \cup V_{Q_i} \cup V_{R_i} \cup V_{M_i}$. Note that $(\bigcup_{i=1}^{q} V_i) \cup V_Y \cup V_Z = V$. Let $V_{R_i} = \{R_{i1}, R_{i2}, ..., R_{i(|M_i|-1)}\}$, $V_{Q_i} = \{Q_{i1}, Q_{i2}, ..., Q_{i(|Mi|-1)}\}$ and $V_{D_i} = \{D_{i1}, D_{i2}, ..., D_{i(|M_i|-1)}\}$, where the job R_{ik} is connected to the job Q_{ik} and the job Q_{ik} is connected to the job D_{ik} for all k $(k \in \{1, 2, \ldots, |M_i| - 1\})$. For every $i \in \{1, \ldots, q\}$, we schedule the jobs of V_i as follows. Recall that one job of V_i, that is the job J_i of V_{M_i} corresponding to M', has already been scheduled as shown in Figure 2.2.2. We schedule the remaining jobs of V_i into $(|M_i| - 1)$ time intervals of length 8 as shown in Figure 2.2.3. To obtain a complete schedule of the jobs of $V_i \setminus \{J_i\}$, we process the jobs of all the quadruplets of the form $\{Q_{ik}, R_{ik}, D_{ik}$, one job of V_i (different from $J_i)\}$ successively as described in Figure 2.2.3, without idle times $(k = 1, \ldots, (|M_i| - 1))$. By scheduling the sets $V_i \setminus \{J_i\}$ $(i = 1, \ldots, q)$ successively followed by the schedule of Figure 2.2.2, we obtain a feasible schedule σ for I with $C_{max}(\sigma) = \sum_{i=1}^{q} 8(|M_i| - 1) + 4q = 8|M| - 4q$.

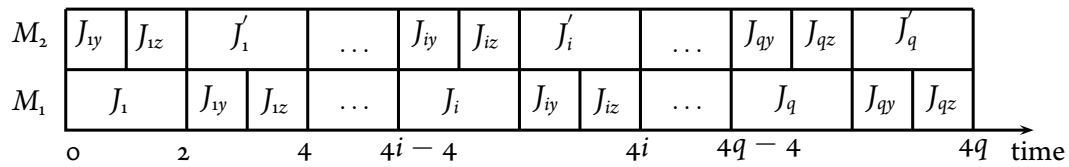


Figure 2.2.4: Instance I of Theorem 2.2: schedule of the remaining jobs of V_M and the jobs of V_Y and V_Z.

Conversely, assume that there is a feasible schedule σ for I with $C_{max}(\sigma) \leq (8|M| - 4q)$. Since the processing times of all the jobs sum up to $(8|M| - 4q)$ on both machines, then there is no idle time on M_1 and M_2 whatsoever. By construction of the graph $\overline{G}$, $(|M_i| - 1)$ quadruplets of the form $\{Q_{ik}, R_{ik}, D_{ik},$

one job of the set $V_{Mi}\}$ must be scheduled as shown in Figure 2.2.3, for every i $(i = 1, \ldots, q)$. Note that the job of V_{M_i} scheduled simultaneously with D_{ik} on M_1 is not necessarily the same as the one scheduled simultaneously with D_{ik} on M_2. The space occupied by all of these jobs on each machine is equal to $[\sum_{i=1}^{q} 8(|M_i|-1)] = 8|M|-8q$. The remaining space is, therefore, equal to $(8|M|-4q)-(8|M|-8q) = 4q$ for each machine, which must be occupied by the remaining jobs of V_M and the $2q$ jobs of V_Y and V_Z without idle times. By construction of the graph $\overline{G}$, the only schedule that avoids an idle time on M_1 and M_2 is the one depicted in Figure 2.2.4, where the jobs $J_i, J_i^{'} \in V_{Mi}$ may be the same $(i = 1, \ldots, q)$. The jobs J_i $(i = 1, \ldots, q)$ correspond to a subset $M' \subseteq M$ with q disjoint elements producing, therefore, a solution for 3-DM. If the jobs J_i and $J_i^{'}$ $(i = 1, \ldots, q)$ are different, we will have another solution of 3-DM represented by the jobs $J_i^{'}$ $(i = 1, \ldots, q)$. The reduction used is polynomial, which completes the proof of Theorem 2.2. □

It has been shown in [90] that $O2|res.11|C_{max}$ is strongly NP-hard. The following result shows that even if the operations of each job have the same requirement of each resource and $p_{ij} \in \{1, 2, 3\}$ with $p_{1j} = p_{2j}$, the problem remains NP-hard in the strong sense.

Corollary 2.3 ([148]). *Problem $O2|res^t.11, p_{ij} \in \{1, 2, 3\}|C_{max}$ with $p_{1j} = p_{2j}$ is NP-hard in the strong sense.*

Proof. Consequence of Proposition 2.1 and Theorem 2.2. □

Now, we restrict our attention to the case of unit-time operations. As shown in Table 2.1.1, $O3|res.11, p_{ij} = 1|C_{max}$ is strongly NP-hard. However, the instance used in the reduction process is not an instance of the problem $O3|res^t.11, p_{ij} = 1|C_{max}$ and the conflicts are between operations in general. In the following theorem we show that the OSC problem with unit-time operations is strongly NP-hard for $m \geq 3$.

Theorem 2.4 ([148]). *The three-machine OSC problem with $p_{ij} = 1$ is NP-hard in the strong sense for arbitrary conflict graph.*

Proof. We prove this result by means of a reduction from 3-DM, that has been used in Theorem 2.2. Given an arbitrary instance of 3-DM, we construct an instance I of the three-machine OSC problem with $p_{ij} = 1$ as follows. Let $X = \{x_1, x_2, \ldots, x_q\}$, $Y = \{y_1, y_2, \ldots, y_q\}$ and $Z = \{z_1, z_2, \ldots, z_q\}$. For each triple $m = (x_i, y_j, z_k) \in M$, we construct the graph $\overline{G}_m = (V_m, \overline{E}_m)$ depicted in Figure 2.2.5. The set V_m contains 18 "internal" vertices $\{a_1^m, \ldots, a_6^m\} \cup \{b_1^m, \ldots, b_6^m\} \cup \{c_1^m, \ldots, c_6^m\}$ and 6 "external" vertices x_i, $\overline{x_i}$, y_j, $\overline{y_j}$, z_k and $\overline{z_k}$, where $\overline{x_i}$ (respectively $\overline{y_j}$ and $\overline{z_k}$) corresponds to x_i (respectively y_j and z_k). Each vertex of V_m represents a job of I. The set $\overline{E}_m$ contains 45 agreement edges between the jobs of V_m. The agreement graph $\overline{G} = (V, \overline{E})$ is obtained from the union of all the gadgets $\overline{G_m}$ as follows: $\overline{G} = (V, \overline{E}) = (\bigcup_{m \in M} V_m, \bigcup_{m \in M} \overline{E}_m)$. Notice that the external vertices are the only vertices in V_m which are shared with other gadgets and which can be incident with further edges beyond those in $\overline{E}_m$. Whereas the internal vertices cannot be connected to other gadgets by an edge in $\overline{E}$.

We now prove that 3-DM has a solution if and only if there exists a feasible schedule σ for I with $C_{max}(\sigma) \leq 18|M| + 6q$.

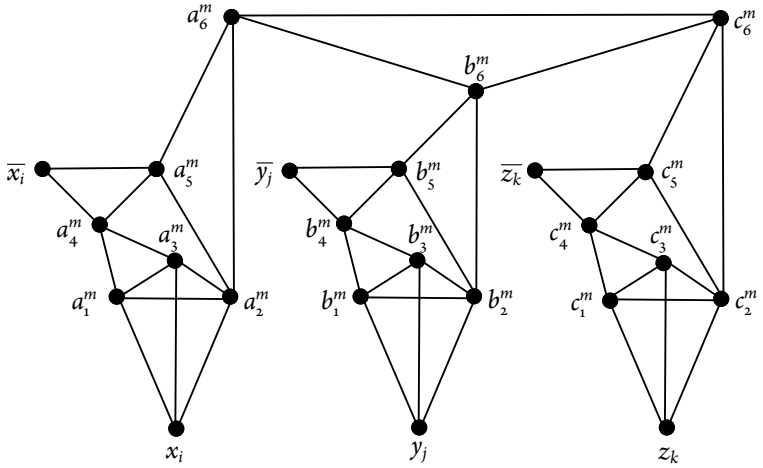


Figure 2.2.5: Instance I of Theorem 2.4: gadget $\overline{G_m} = (V_m, \overline{E_m})$.

First assume that 3-DM has a solution M'. We construct a schedule σ for I as follows. For each $m' = (x'_i, y'_j, z'_k) \in M'$, the jobs of $\overline{G_{m'}}$ are processed as shown in Figure 2.2.6. In this figure (as well as in Figures 2.2.7, 2.2.8 and 2.2.9), each job is split into three operations and each operation is associated with a number representing its machine. Every single triangle joins three operations processed simultaneously on the three machines. The schedule of the jobs of $\overline{G_{m'}}$ is obtained by processing the jobs x'_i, $a_1^{m'}$, $a_2^{m'}$ and $a_3^{m'}$ simultaneously without idle times, which requires a time interval of length 4 (the same for the jobs y'_j, $b_1^{m'}$, $b_2^{m'}$ and $b_3^{m'}$ and for the jobs z'_k, $c_1^{m'}$, $c_2^{m'}$ and $c_3^{m'}$). Then, we process the jobs $\overline{x'_i}$, $a_4^{m'}$ and $a_5^{m'}$ simultaneously in a time interval of length 3 (the same for the jobs $\overline{y'_j}$, $b_4^{m'}$ and $b_5^{m'}$ and for the jobs $\overline{z'_k}$, $c_4^{m'}$ and $c_5^{m'}$). Finally, we process the jobs $a_6^{m'}$, $b_6^{m'}$ and $c_6^{m'}$ simultaneously in a time interval of length 3. In summary, 24 time units are needed to process all the jobs of $\overline{G_{m'}}$. Therefore, the jobs of all the gadgets corresponding to the triples of M' are scheduled in a time interval of length $24q$. Notice that all the external jobs have been scheduled. It remains the internal jobs of the gadgets corresponding to triples not in M'. For each triple $m = (x_i, y_j, z_k) \in M \setminus M'$, the internal jobs of $\overline{G_m}$ are scheduled as shown in Figure 2.2.7. We process the jobs a_1^m, a_3^m and a_4^m simultaneously without idle times in a time interval of length 3 (the same for the jobs b_1^m, b_3^m and b_4^m, for c_1^m, c_3^m and c_4^m, for a_2^m, a_5^m and a_6^m, for b_2^m, b_5^m and b_6^m and for the jobs c_2^m, c_5^m and c_6^m). Then, 18 time units are required to process these jobs. The internal jobs of all the gadgets corresponding to triples in $M \setminus M'$ are schedules in a time interval of length $18(|M| - q)$. A complete schedule σ is obtained by concatenating all the partial schedules. The schedule σ is feasible since it satisfies all the conflict constraints, and $C_{max}(\sigma) = 24q + 18(|M| - q) = 18|M| + 6q$.

Conversely, suppose that there exists a feasible schedule σ for I with $C_{max}(\sigma) \leq 18|M| + 6q$. Since the total processing time of jobs on each machine is $18|M| + 6q$, then there is no idle time on all the machines and $C_{max}(\sigma) = 18|M| + 6q$. In what follows, we associate a configuration $(\alpha_i^m, \beta_j^m, \gamma_k^m)$ with

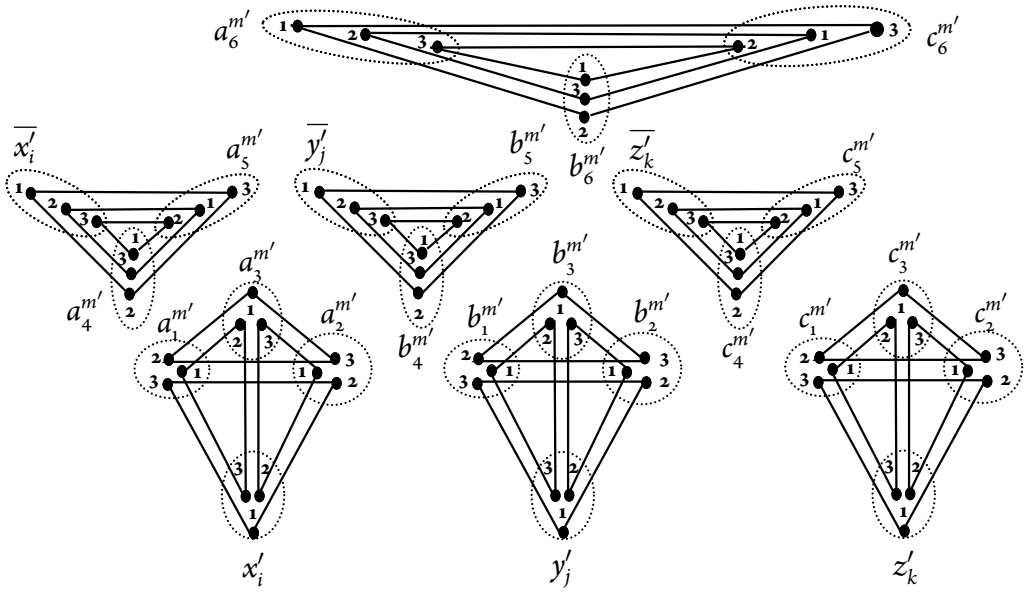


Figure 2.2.6: Instance I of Theorem 2.4: schedule of the jobs of $\overline{G_{m'}}$.

each $m = (x_i, y_j, z_k) \in M$, where a_i^m (respectively β_j^m and γ_k^m) indicates the number of operations of x_i (respectively y_j and z_k) processed inside the gadget $\overline{G_m}$. We shall need the following lemmas.

Lemma 2.5. *For every $m = (x_i, y_j, z_k) \in M$, we have $a_i^m, \beta_j^m, \gamma_k^m \in \{0, 3\}$.*

Proof. Let us first prove that $a_i^m \in \{0, 3\}$. Consider the job $\overline{x_i}$ and let $\overline{a_i^m}$ be the number of operations of $\overline{x_i}$ processed inside the gadget $\overline{G_m}$. Four cases may occur depending on the values of $\overline{a_i^m}$.

Case 1: $\overline{a_i^m} = 0$**.** Since there is no idle time on the three machines, the job a_4^m has to be processed simultaneously with the jobs a_1^m and a_3^m in a time interval of length 3. Also, the job a_5^m has to be processed simultaneously with the jobs a_2^m and a_6^m, which requires 3 time units (see Figure 2.2.7). Therefore, $a_i^m = 0$.

Case 2: $\overline{a_i^m} = 1$**.** The only schedule that avoids idle times on the three machines is the schedule given in Figure 2.2.8, where the remaining two operations of a_4^m are processed simultaneously with two operations of each of the jobs a_1^m and a_3^m. Similarly, the remaining two operations of a_5^m have to be processed simultaneously with two operations of each of the jobs a_6^m and a_2^m. We are left with one operation of each of the jobs a_1^m, a_2^m and a_3^m, that have to be processed simultaneously to avoid idle times. Therefore, $a_i^m = 0$.

Case 3: $\overline{a_i^m} = 2$**.** The schedule of this case is depicted in Figure 2.2.9. To avoid idle times, the remaining operation of a_4^m has to be processed simultaneously with two operations of a_1^m and a_3^m. Similarly, the remaining operation of a_5^m has to be processed simultaneously with two operations of a_6^m and a_2^m. We are left with two operations of each of the jobs a_1^m, a_2^m and a_3^m. Two ways of scheduling these operations can be observed:

- (a) The remaining operations of a_1^m, a_2^m and a_3^m are scheduled simultaneously in a time interval of

length 2, so we have $a_i^m = 0$. Recall that two operations of $\overline{x_i}$ have been already scheduled. Then, the operations of x_i will be scheduled, in another gadget containing x_i, with at most one operation of $\overline{x_i}$. However, from Cases 1 and 2 we have that if at most one operation of $\overline{x_i}$ is processed inside a given gadget, then none of the operations of x_i can be processed with the jobs of this gadget without inducing idle times. Therefore, (a) cannot occur.

- (b) The remaining operations of a_1^m, a_2^m and a_3^m are scheduled simultaneously with the operations of x_i. To avoid idle times, all the operations of x_i have to be processed inside this gadget (see Figure 2.2.9), so we have $a_i^m = 3$.

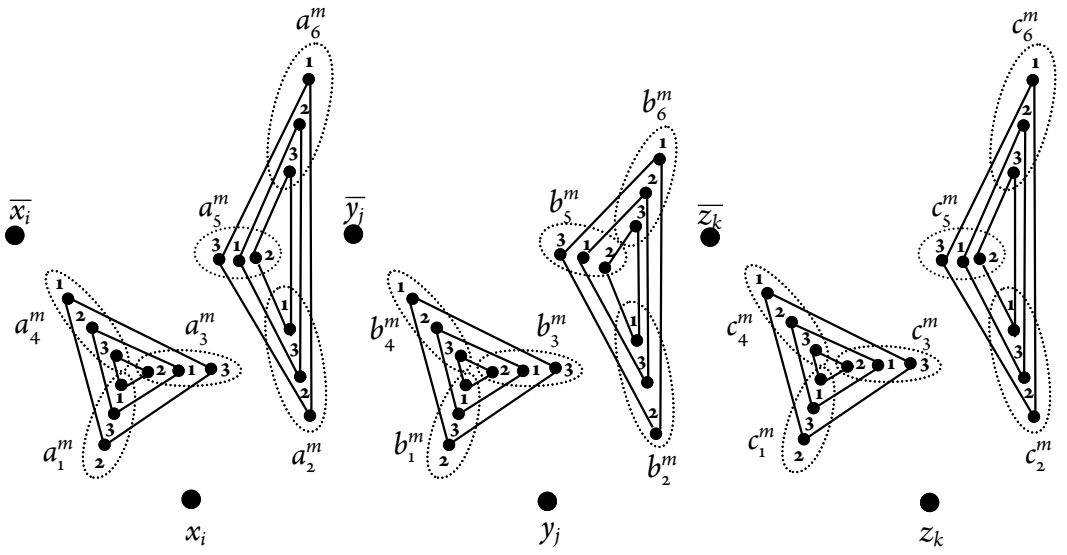


Figure 2.2.7: Instance I of Theorem 2.4: schedule of the internal jobs of $\overline{G_m}$.

Case 4: $\overline{a_i^m} = 3$**.** In this case all the operations of a_4^m and a_5^m have been scheduled simultaneously with the operations of $\overline{x_i}$ in a time interval of length 3. Two ways of scheduling the operations of a_1^m, a_2^m and a_3^m can be observed:

- (a) The operations of a_1^m, a_2^m and a_3^m are scheduled simultaneously in a time interval of length 3, so we have $a_i^m = 0$. Recall that all the operations of $\overline{x_i}$ have been already scheduled. Then, the operations of x_i will be scheduled inside another gadget containing x_i without processing any operation of $\overline{x_i}$. However, from Case 1 we have that if none of the operations of $\overline{x_i}$ is processed inside a given gadget, then none of the operations of x_i can be processed with the jobs of this gadget without inducing idle times. Therefore, (a) cannot occur.

- (b) The operations of a_1^m, a_2^m and a_3^m are scheduled simultaneously with the operations of x_i. To avoid idle times, all the operations of x_i have to be processed inside this gadget as shown in Figure 2.2.6. Therefore, $a_i^m = 3$.

In all the possible cases, we have $a_i^m \in \{0, 3\}$. Similarly, we prove that $\beta_j^m, \gamma_k^m \in \{0, 3\}$ and the result follows. □

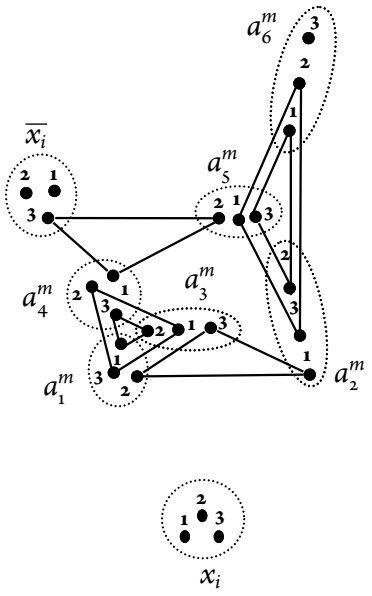


Figure 2.2.8: Instance I of Theorem 2.4: schedule corresponding to Case 2.

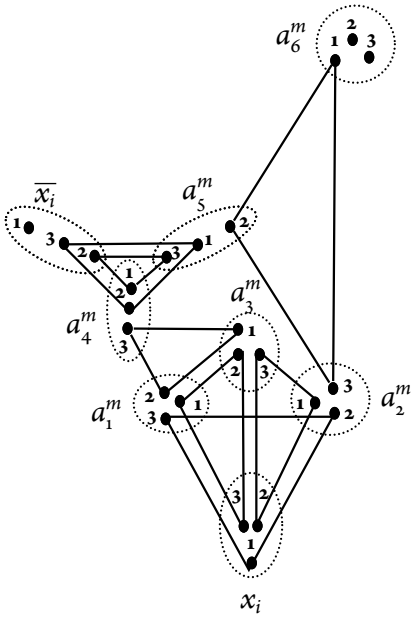


Figure 2.2.9: Instance I of Theorem 2.4: schedule corresponding to Case 3.

Lemma 2.6. *For every $m = (x_i, y_j, z_k) \in M$, we have $a_i^m = \beta_j^m = \gamma_k^m$.*

Proof. We first show that $a_i^m = \beta_j^m$. Since $a_i^m, \beta_j^m \in \{0, 3\}$ from Lemma 2.5, we suppose without loose of generality that $a_i^m = 3$ and $\beta_j^m = 0$. When $a_i^m = 3$, only the Cases 3 and 4 of Lemma 2.5 may occur. In these cases, at most one operation of a_6^m is processed simultaneously with a_2^m and a_5^m. To avoid idle times, at least two operations of a_6^m have to be processed simultaneously with at least two operations of each of the jobs b_6^m and c_6^m. However, at most one operation of b_6^m can be processed simultaneously with a_6^m and c_6^m. Indeed, we have $\beta_j^m = 0$ which is possible only in the Cases 1 and 2 of Lemma 2.5. In these cases, at least two operations of b_6^m are scheduled simultaneously with b_2^m and b_5^m. Then, at most one operation of b_6^m can be processed simultaneously with a_6^m and c_6^m. This implies that, an idle time will occur in one of the machines when processing a_6^m, which contradicts the fact that there is no idle times in σ. Therefore, $a_i^m = \beta_j^m$. Similarly, we prove that $\beta_j^m = \gamma_k^m$, then $a_i^m = \beta_j^m = \gamma_k^m$. The lemma follows. □

Now, we consider the set M' of triples corresponding to all the gadgets $\overline{G_m}$ of $\overline{G}$ with $a_i^m = \beta_j^m = \gamma_k^m = 3$. As a consequence of Lemmas 2.5 and 2.6, M' covers all the elements in X, Y and Z. Furthermore, each element of $X \cup Y \cup Z$ is contained in exactly one triple of M' because each job has exactly three operations. Therefore, M' contains q disjoint triples, and a solution of 3-DM follows. The reduction used is polynomial, which completes the proof of Theorem 2.4. □

As it is mentioned in Table 2.1.1, $O3|res.11, p_{ij} = 1|C_{max}$ is strongly NP-hard. The following corollary shows that this result holds even if the operations of each job have the same requirement of each resource.

Corollary 2.7 ([148]). *The problem $O3|res^t.11, p_{ij} = 1|C_{max}$ is NP-hard in the strong sense.*

Proof. Consequence of Proposition 2.1 and Theorem 2.4. □

Based on the result of the previous theorem, we study the difficulty of the following variant of Partition Into Triangles (PIT)[60]. An instance of this variant consists of a 3-partite graph $R = (V_1 \cup V_2 \cup V_3, U)$, where $V_i = \{v_{i1}, \ldots, v_{in}\}$ is an independent set $(i = \overline{1,3})$ and the edges of U can be partitioned into cycles of length 6 of the form $\{v_{1l}, v_{2k}, v_{3l}, v_{1k}, v_{2l}, v_{3k}, v_{1l}\}$, $(l \neq k)$ $l, k = \overline{1,n}$. Can the vertices of R be partitioned into n disjoint sets each containing exactly 3 vertices, such that each of these sets induces a triangle in R?

Observe that graph R has the following characteristics.

Remark 2.1. *Every vertex of R has even degree.*

Remark 2.2. *The vertices v_{1j},v_{2j} and v_{3j} form an independent set, $\forall\, j = \overline{1,n}$.*

We have not been able to locate any results in the literature regarding the complexity of PIT on the particular 3-partite graph R (this problem is denoted P3P-PIT). PIT is known to be NP-hard in the strong sense for arbitrary 3-partite graphs [60]. Van Rooij, Van Kooten Niekerk and Bodlaender [158] proved that PIT problem is polynomial-time solvable on graphs of maximum degree three, but graph R may have vertices of degree greater than 3. They also established NP-hardness results for 4-regular graphs, however their graphs are not 3-partite. Recently, Ćustić, Klinz and Woeginger [43] showed that PIT on 6-regular 3-partite graphs is NP-hard, but their graph is an arbitrary 6-regular 3-partite graph.

Theorem 2.8. *P3P-PIT is NP-hard in the strong sense.*

Proof. The candidate used in the reduction process is $O3|ConfG = (V, E), p_{ij} = 1|C_{max}$. This problem is shown in Theorem 2.4 to be NP-hard in the strong sense. It is defined as follows: given a set $\{J_1, \ldots, J_n\}$ of n jobs that has to be processed on three machines such that $p_{ij} = 1$; $\forall\, i = \overline{1,3}, j = \overline{1,n}$, and a conflict graph $G = (V, E)$ over the jobs. Does there exist a feasible schedule σ such that $C_{max}(\sigma) \leq n$.

Given an arbitrary instance I of $O3|ConfG = (V, E), p_{ij} = 1|C_{max}$, we construct the graph $R = (V_1 \cup V_2 \cup V_3, U)$ as follows:

- Each operation J_{ij} of I is associated with a vertex v_{ij} in V_i, $i = \overline{1,3}, j = \overline{1,n}$.

- For each two jobs J_l and J_k, such that $\{J_l, J_k\} \notin E$, we construct a cycle in R of length 6 of the form $\{v_{1l}, v_{2k}, v_{3l}, v_{1k}, v_{2l}, v_{3k}, v_{1l}\}$.

Thus, two operations can be processed simultaneously if and only if their corresponding vertices are adjacent in R.

We show that there exists a feasible schedule σ for I such that $C_{max}(\sigma) \leq n$ if and only if P3P-PIT has a solution.

Suppose that there is a feasible schedule σ for I such that $C_{max}(\sigma) \leq n$. Since the processing times of all the jobs sum up to n on each machine, then there is no idle time on all the machines and $C_{max}(\sigma) = n$. Therefore, for each time unit, we have three operations processed simultaneously on the three machines. By construction of the graph R, these operations correspond to three vertices that are adjacent in R. Since the operations of I are in one-to-one correspondence with the vertices of R, a solution of P3P-PIT follows.

Conversely, assume that P3P-PIT has a solution. We construct a schedule σ for I by processing the operations corresponding to each triangle of the solution of P3P-PIT simultaneously on the three machines, which requires one time unit for each triangle. The obtained schedule is feasible since it satisfies all the conflict constraints, and $C_{max}(\sigma) = n$. The reduction used is polynomial, which completes the proof of the theorem. □

2.2.2 Well solvable cases

In this section, we discuss special cases that can be solved efficiently. Jurisch and Kubiak presented in [90] a polynomial algorithm for the problem $O2|res1..|C_{max}$ which is one of the most important polynomial cases of the two-machine open shop problem under resource constraints. Since $O2|res^t1..|C_{max}$ is a subproblem of $O2|res1..|C_{max}$, then $O2|res^t1..|C_{max}$ can also be solved in polynomial time. In what follows, we construct the conflict graph $G = (V, E)$ associated with the problem $O2|res^t1..|C_{max}$ and let R_s be the required resource such that $|R_s| = r$. Each job of $O2|res^t1..|C_{max}$ is associated with a vertex in V, let $A = \{J_j | R_s(J_j) \leq \lfloor \frac{r}{2} \rfloor\}$ and $B = \{J_j | R_s(J_j) > \lfloor \frac{r}{2} \rfloor\}$. Clearly, $V = A \cup B$. The conflict edges are defined by the relation: $\{J_l, J_k\} \in E$ if and only if $R_s(J_k) + R_s(J_l) > r$. Therefore, the subgraph $G[A]$ induced by the vertices of A is an independent set and the subgraph $G[B]$ induced by the vertices of B is a clique. Now, observe that if $J_k \in A$ and $J_l \in B$ with $R_s(J_k) + R_s(J_l) > r$, then J_k is adjacent to each job J_j of B such that $R_s(J_j) \geq R_s(J_l)$ and J_l is adjacent to each job J_j of A such that $R_s(J_j) \geq R_s(J_k)$. Therefore, the obtained conflict graph is a particular split graph. For instance, the conflict graph associated with the problem $O2|res^t111|C_{max}$ is a collection of a clique $G[B]$ and an independent set $G[A]$, where no edge exists between the sets A and B.

In the following, we consider the case of an arbitrary conflict graph and we show that the two-machine OSC problem with $p_{ij} \in \{0, 1, 2\}$ is solvable in polynomial time.

Theorem 2.9 ([148]). *For an arbitrary conflict graph, the two-machine OSC problem with $p_{ij} \in \{0, 1, 2\}$ is solvable in $O(n^{2.5})$ time.*

Proof. (P) in this proof denotes the two-machine OSC problem with $p_{ij} \in \{0, 1, 2\}$. Let I be an instance of (P) and $\overline{G} = (V, \overline{E})$ is the agreement graph associated. We define an auxiliary graph $\overline{G'} = (V', \overline{E'})$ from $\overline{G}$ as follows (see Figure 2.2.10). The vertex set V' contains one vertex for each job of V that comprises a unique operation, and two vertices for each job of V that comprises two operations. Observe that the vertices of V' represent the operations of the jobs of V. The edge set $\overline{E'}$ consists of agreement edges between operations. An agreement edge exists between two operations if their corresponding jobs are adjacent in $\overline{G}$ subject to the constraints that there is no agreement edge between two operations of the same machine and between two operations of the same job. If we regard the vertices of $\overline{G'}$ as jobs (with a unique operation), one can verify that $\overline{G'}$ is an agreement graph over these jobs.

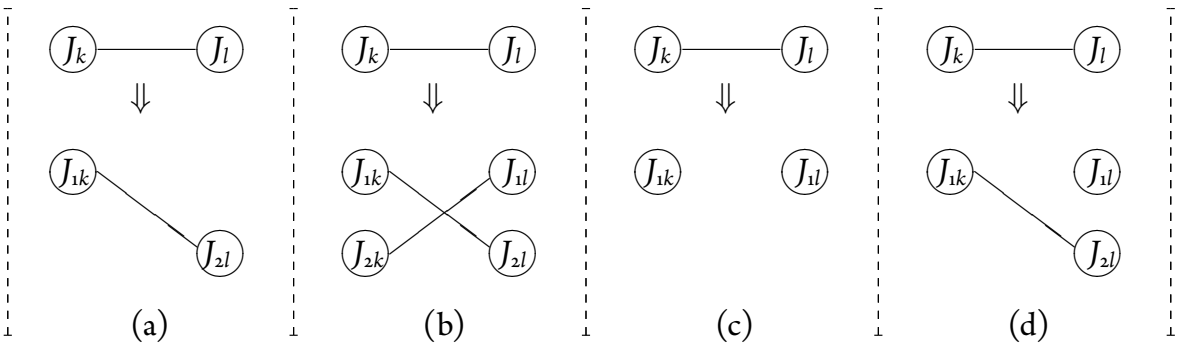


Figure 2.2.10: Some transformations from $\overline{G} = (V, \overline{E})$ to $\overline{G'} = (V', \overline{E'})$: (a) jobs with one operation on different machines,(b) jobs with two operations, (c) jobs with one operation on a same machine, (d) jobs with different numbers of operations.

Let I' be an instance of (P), and $\overline{G'}$ be the agreement graph associated. Observe that an optimal schedule for I' is also optimal for I. We consider another instance I'' derived from I' by relaxing the constraints on the operations, such that each operation can be processed on any of the two machines. I'' can now be viewed as an instance of the two-machine SCI problem with jobs sizes in $\{1, 2\}$, and can be solved in polynomial time using an exact algorithm proposed in [53]. By applying this algorithm, we obtain a partition of $\overline{G'}$ into disjoint cycles of even lengths and paths. Each path or cycle induces a partial schedule in which jobs (operations of I) alternate between the two machines. An optimal schedule σ for I'' is obtained by concatenating all the partial schedules, the corresponding makespan is equal to the sum of the lengths of the partial schedules. Recall that two adjacent jobs in I'' correspond to two operations of different machines, then if the first operation of each path (resp. cycle) is processed on its respective machine, the remaining operations of this path (resp. cycle) will be processed on their

respective machines too. Furthermore, when constructing σ the first operation of each path or cycle can be processed either on M_1 or M_2 without changing the makespan. We suppose that the first operation of each path or cycle in σ is processed on its respective machine. In this case, σ is a feasible schedule for the original instance I' but it remains optimal for the relaxed instance I'', thus σ is an optimal schedule for I'. Since an optimal schedule for I' is also optimal for I, then σ is an optimal schedule for I. The graph $\overline{G'}$ is constructed in $O(n^2)$, the algorithm proposed in [53] runs in $O(n^{2.5})$ time. Therefore, an optimal schedule for I can be found in $O(n^{2.5})$ and the theorem follows. □

From this theorem, new complexity result of a restricted case of $O2|res.11|C_{max}$ can be derived. Indeed, the following corollary shows that when the operations of a job have the same requirement of each resource and $p_{ij} \in \{0, 1, 2\}$, the problem can be solved in polynomial time.

Corollary 2.10 ([148]). *The problem $O2|res^t.11, p_{ij} \in \{0, a, 2a\}|C_{max}$ is solvable in polynomial time for any fixed integer a.*

Proof. Consequence of Proposition 2.1 and Theorem 2.9. □

	J_1	J_2	J_3	J_4	J_5
M_1	2	2	1	0	1
M_2	2	0	1	1	0

(a)

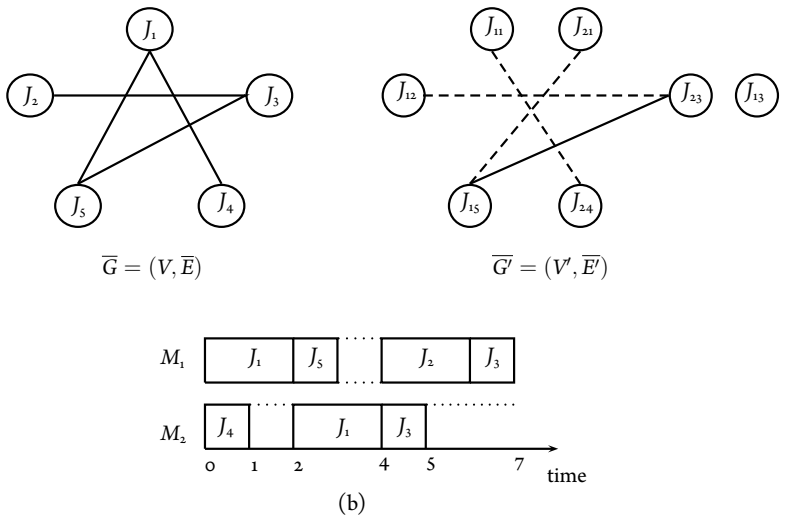


Figure 2.2.11: Application of the algorithm of Theorem 2.9: (a) processing times. (b) transformation from $\overline{G} = (V, \overline{E})$ to $\overline{G'} = (V', \overline{E'})$ and the construction of the optimal schedule, where the dashed edges represent the paths obtained after applying the algorithm proposed in [53].

Figure 2.2.11 illustrates the algorithm of Theorem 2.9 using a 5-job, 2-machine instance. From the graph $\overline{G'} = (V', \overline{E'})$, the algorithm proposed in [53] constructs first another graph $\overline{G''}$. The vertex set of $\overline{G''}$ contains one vertex for each unit-time operation, and two vertices for each operation whose processing time equals 2, these two vertices are denoted type-a and type-b. The edge set of $\overline{G''}$ consists of agreement edges between operations. In other words, agreement edges are induced by $\overline{G'}$ subject to the constraint that there are no agreement edges between a type-a vertex and a type-b vertex. In the second step, the

algorithm finds a maximum matching in $\overline{G''}$. The obtained maximum matching for our example is: type-a of J_{11} - J_{24}, type-a of J_{21} - J_{15} and J_{12} - J_{23}. Now, for each edge of this matching, there is an edge in $\overline{E'}$. If an edge in the matching is incident to a type-a or a type-b of a vertex v, then the corresponding edge in $\overline{E'}$ is incident to the vertex v. The edges corresponding to the above maximum matching are the dashed edges in Figure 2.2.11.(b). The graph induces by these edges can be partitioned into disjoint cycles of even lengths and paths (in this example we have only paths), each one induces a schedule in which operations alternate between the two machines. The optimal schedule is obtained by the concatenation of the obtained partial schedules as depicted in Figure 2.2.11.(b).

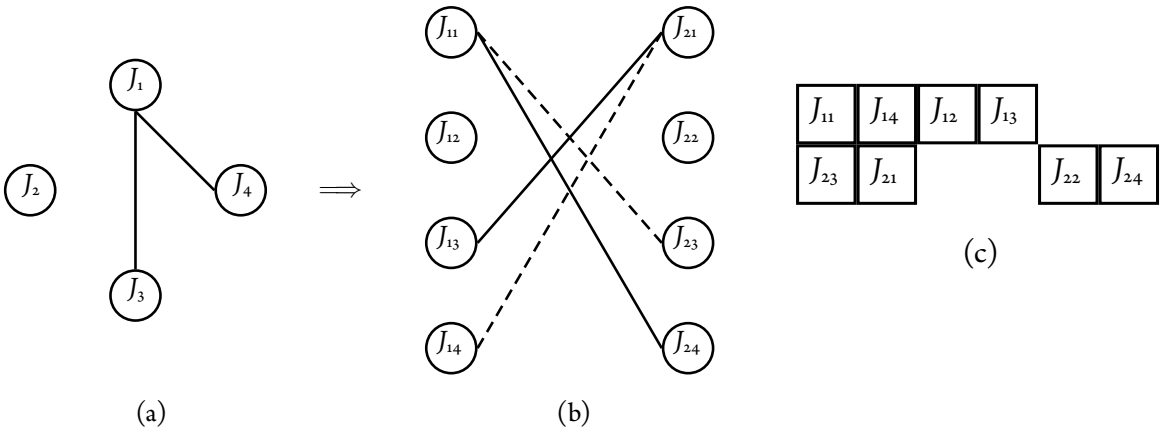


Figure 2.2.12: Application of the algorithm of Theorem 2.11: (a) agreement graph $\overline{G} = (V, \overline{E})$. (b) bipartite graph $R = (V_1 \cup V_2; U)$ associated, where the dashed edges represent a maximum matching. (c) an optimal schedule.

Now, we restrict ourselves to the two-machine OSC problem with unit-time operations, for which we present an algorithm much simpler than the one of Theorem 2.9. This algorithm can be obtained by slightly modifying an approach for solving the problem $O2|res\ldots, p_{ij} = 1|C_{max}$ (see [15]). For a given instance I of the two-machine OSC problem with $p_{ij} = 1$, we construct the bipartite graph $R = (V_1 \cup V_2; U)$ from $\overline{G}$ as follows:

- $V_i = \{J_{i1}, \ldots, J_{in}\}$ represents the operations of the jobs of V to be processed on machine M_i.
- For each two jobs J_l and J_k, two edges $\{J_{2l}, J_{1k}\}$ and $\{J_{1l}, J_{2k}\}$ exist if and only if J_l and J_k are adjacent in $\overline{G}$.

Thus, two operations of I can be processed simultaneously if and only if their corresponding vertices are adjacent in R.

Theorem 2.11 ([148]). *For an arbitrary conflict graph G, the two-machine OSC problem with $p_{ij} = 1$ is solvable in $O(n^{2.5})$ time and the optimal makespan value equals $2n - |M|$, where M is a maximum matching in R.*

Proof. Let M be a maximum matching in R. A feasible schedule σ for I can be obtained by scheduling matched operations in M simultaneously. This requires $|M|$ time units to schedule all the operations incident to the matching M. It remains $2n - 2|M|$ operations that have to be processed in disjoint time intervals (otherwise the matching M is not maximum). Thus, $C_{max}(\sigma) = |M| + (2n - 2|M|) = 2n - |M|$. Clearly, this schedule is optimal since a maximum number of operations are processed simultaneously on the two machines. The graph R is constructed in $O(n^2)$ and a maximum matching in this graph can be found in $O(n^{2.5})$. Therefore, the time complexity of the above algorithm is $O(n^{2.5})$ and the theorem follows. The example of Figure 2.2.12 illustrates this algorithm. □

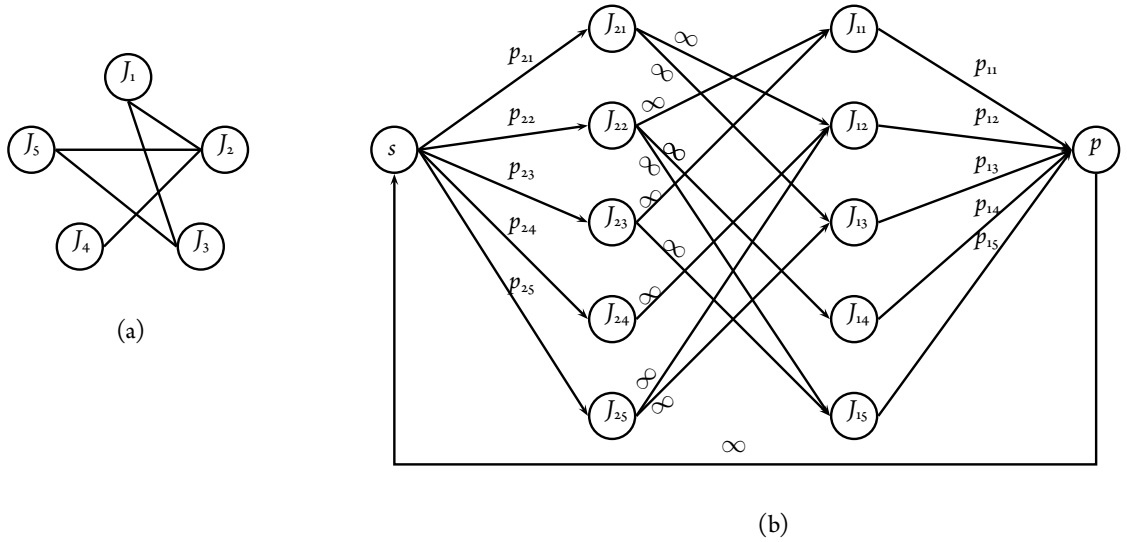


Figure 2.2.13: Construction of the network graph of Theorem 2.12: (a) agreement graph $\overline{G} = (V, \overline{E})$. (b) network $N = (V, U_c, C)$ associated.

In what follows, we show that by allowing preemptions, we make the two-machine OSC problem easy to solve. An exact algorithm is then presented to solve this version in cubic time. This algorithm is based on a network graph. The maximum flow in this network is computed using an algorithm due to Ford and Fulkerson [55]. We refer the reader to [18], where a detailed analysis of the networks and the algorithm of Ford and Fulkerson is presented.

Theorem 2.12 ([148]). *The preemptive version of the two-machine OSC problem can be solved in $O(n^3)$ time for an arbitrary conflict graph.*

Proof. From Proposition 2.1 and the result 3 of Table 2.1.1, it follows that the preemptive version of the two-machine OSC problem can be solved in polynomial time. The approach proposed in [90] to solve problem $O2|res..., pmtn|C_{max}$ can be adapted to solve our problem. Given an instance I of the preemptive version of the two-machine OSC problem, we consider the network $N = (V, U_c, C)$ defined as follows (see Figure 2.2.13 for illustration):

- $V = V_1 \cup V_2 \cup \{s, p\}$, where $V_i = \{J_{i1}, \ldots, J_{in}\}$ represents the operations of the jobs of V to be processed on machine M_i, s is the source vertex and p is the sink vertex.

The set of directed arcs U_c is constructed as follows:

- For each vertex J_{2j} of V_2, an arc (s, J_{2j}) exists with a limited capacity $C(s, J_{2j}) = p_{2j}$.
- For each vertex J_{1j} of V_1, an arc (J_{1j}, p) exists with a limited capacity $C(J_{1j}, p) = p_{1j}$.
- For each couple of jobs J_l and J_k, two arcs (J_{2k}, J_{1l}) and (J_{2l}, J_{1k}) exist if J_k and J_j are adjacent in $\overline{G}$. These arcs have an unlimited capacity.

We consider the problem of finding a maximum flow in N. An optimal schedule for I can be obtained from an optimal solution for the network flow problem as follows.

Algorithm 3 Algorithm for the preemptive version of the two-machine OSC problem.

Input: the problem data.

1: Find a maximum flow f in N.
2: Process operations J_{1j}, $j = 1, \ldots, n$ in any order.
3: Process $f(J_{2l}, J_{1k})$ time units of operation J_{2l} concurrently with operation J_{1k}.
4: For each non saturated arc (s, J_{2l}), process on M_2 the remaining $p_{2l} - f(s, J_{2l})$ time units of operation J_{2l} after completing the last operation on M_1.

Output: The optimal schedule.

From Algorithm 3, any optimal solution of the network flow problem can be converted into a feasible schedule σ of I with $C_{max} = \sum_{j=1}^{n} p_{1j} + \sum_{j=1}^{n}(p_{2j} - f(s, J_{2j}))$. $C_{max}(\sigma)$ is at its minimum value when $\sum_{j=1}^{n} f(s, J_{2j})$ is maximal. This sum corresponds to the optimal value of the network flow problem. Thus, the obtained schedule is optimal. Since a maximum flow can be found in $O(n^3)$ time, the theorem follows. □

To illustrate this algorithm let us consider an example of five jobs whose processing times are given in Figure 2.2.14.(a). The conflict constraints are specified in Figure 2.2.13.(a). The computation of a maximum flow in the network graph N of Figure 2.2.13.(b) and the construction of the optimal schedule are given in Figure 2.2.14.

We have seen in Theorem 2.4 that the three-machine OSC problem with $p_{ij} = 1$ is NP-hard in the strong sense. In what follows, we present a special case that can be solved efficiently. Let I be an instance of the three-machine OSC problem with $p_{ij} = 1$, where $\overline{G} = (V, \overline{E})$ is a triangle-free graph. We consider the 3-partite graph $R = (V_1 \cup V_2 \cup V_3, U)$ defined as follows:

- $V_i = \{J_{i1}, \ldots, J_{in}\}$ represents the operations of the jobs of V to be processed on machine M_i.

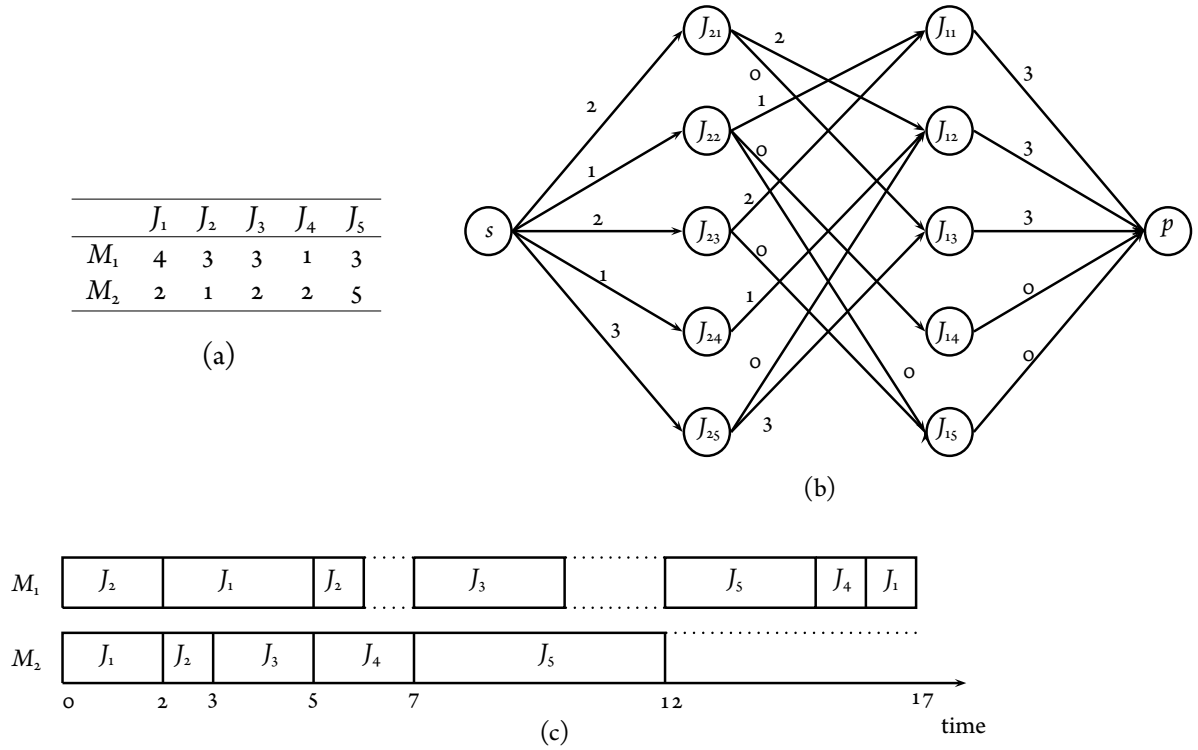


Figure 2.2.14: Application of the algorithm of Theorem 2.12: (a) processing times. (b) a solution of the maximum flow problem of the network $N = (V, U_c, C)$, where the value of the maximum flow is equal to 9. (c) an optimal schedule.

- For each two jobs J_l and J_k, six edges $\{\{J_{ul}, J_{vk}\} | u \neq v, u, v = \overline{1,3}\}$ exist if and only if $\{J_l, J_k\} \in \overline{E}$.

Thus, two operations of I can be processed simultaneously if and only if their corresponding vertices are adjacent in R.

Theorem 2.13 ([148]). *The three-machine OSC problem with $p_{ij} = 1$ is solvable in $O(n^{2.5})$ time for G being a complement of a triangle-free graph and the optimal makespan value equals $3n - |M|$, where M is a maximum matching in R.*

Proof. By construction of the 3-partite graph R, an edge exists between two operations of R if their corresponding jobs are adjacent in $\overline{G}$. Therefore, R cannot contain triangles since $\overline{G}$ does not contain any. This implies that, at any time, at most two operations can be processed simultaneously. Let M be a maximum matching in R. Clearly, an optimal schedule σ for I can be found by scheduling matched operations in M simultaneously. This requires $|M|$ time units to schedule all the operations incident to the matching M. It remains $3n - 2|M|$ operations that have to be processed in disjoint time intervals. Thus, $C_{max}(\sigma) = |M| + (3n - 2|M|) = 3n - |M|$. The graph R is constructed in $O(n^2)$ and a maximum matching in this graph can be found in $O(n^{2.5})$. Therefore, the time complexity of the above algorithm is $O(n^{2.5})$ and the theorem follows. The algorithm of this theorem is illustrated by the example of Figure 2.2.15. □

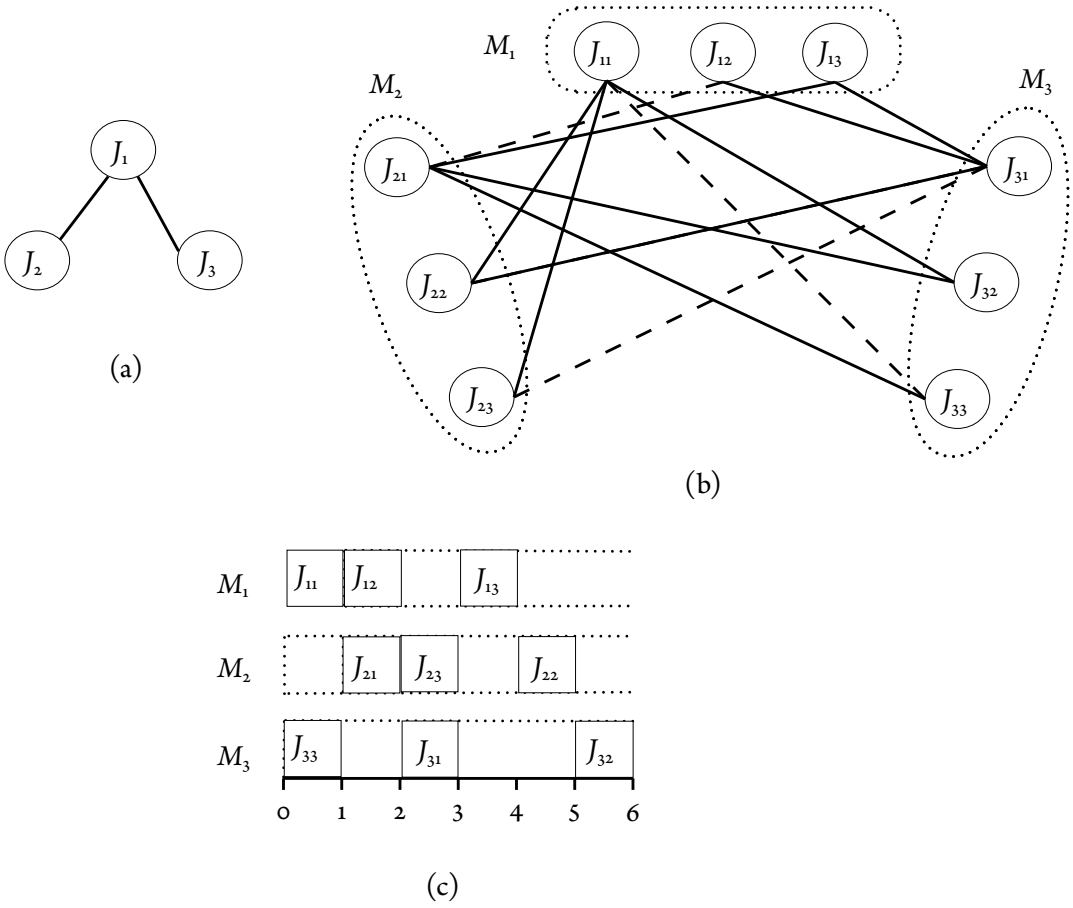


Figure 2.2.15: Application of the algorithm of Theorem 2.13: (a) agreement graph $\overline{G} = (V, \overline{E})$. (b) 3-partite graph $R = (V_1 \cup V_2 \cup V_3, U)$ associated, where the dashed edges represent a maximum matching. (c) an optimal schedule.

2.3 Lower bounds

In what follows, we present three lower bounds on the makespan. These lower bounds are used, in the next sections, to discard nodes in a beam search technique and for benchmarking heuristic solutions.

2.3.1 Weighted agreement graph-based lower bound [148]

This lower bound is obtained by considering the weighted agreement graph $\overline{G}_w = (V, \overline{E}, W)$ defined as follows: let $\overline{G} = (V, \overline{E})$ be the agreement graph and W be the vector representing the processing times of the jobs of V. If we regard these processing times as weights in $\overline{G}$ (i.e. each vertex is associated with a weight that equals the processing time of the corresponding job), we obtain the weighted graph $\overline{G}_w = (V, \overline{E}, W)$. Since nonadjacent jobs in $\overline{G}_w$ cannot be processed simultaneously for any feasible schedule, then the weight of an independent set of maximum weight in $\overline{G}_w$ represents a lower bound on the makespan. The maximum weighted independent set problem is NP-hard, thus two greedy algorithms, namely GWMIN and GWMIN2, presented in Sakai et al. [136] have been considered. The

resulting lower bounds are denoted LB_1 and LB_2 respectively.

2.3.2 Conflict constraints relaxation-based lower bound [148]

This lower bound can be derived by relaxing the conflict constraints between the jobs. The problem is then reduced to the basic m-machine open shop problem. Therefore, the maximum of job durations and machine loads (which is a classical lower bound for shop problems), i.e.

$$LB_3 = \max\{\max_{0\leqslant j\leqslant n}\{\sum_{i=1}^{m} p_{ij}\}, \max_{0\leqslant i\leqslant m}\{\sum_{j=1}^{n} p_{ij}\}\} \tag{2.1}$$

is a valid lower bound on the makespan.

An illustration of the above lower bounds will be given in Example 2.1 while applying the second phase of the heuristic approach of the next section.

2.4 Two-phase heuristic approach

We present in this section a heuristic approach for the m-machine OSC problem. This approach is mainly composed of two phases. In the first phase, the set of operations of all the jobs is partitioned into a minimum number of subsets, called schedule slices, such that the operations of each slice can be processed simultaneously. In the second phase, the resulting schedule slices are inserted one by one to make a complete schedule by using a beam search technique. This approach is proposed in [148].

2.4.1 First phase

The goal of this phase is to generate a minimum number of schedule slices with a minimum total idle time on each slice. To avoid having too many idle times in one schedule slice, the idea is to compute a subset of operations as balanced as possible. To do so, we partition the conflict graph G into a minimum number of independent sets using the greedy algorithm GMIN (see Algorithm 2) as follows: find an independent set in G using GMIN, remove the vertices of this independent set from the graph G and iterate this process on the remaining graph until no vertex remains. For each independent set, the problem is reduced to the basic m-machine open shop problem, so the slices can be computed using a matching algorithm that has been applied in [27], [29] and [70].

For each independent set S, the matching algorithm constructs first a weighted bipartite graph $B_S = (J_S \cup M, O_S, W_S)$ as follows. J_S is the set of the jobs of S and M is the set of the m machines. Each operation J_{ij} of a job $J_j \in J_S$ is represented in the edge set O_S by an edge between J_j and M_i of weight

p_{ij}. Observe that a matching in B_S represents a set of operations that can be processed simultaneously, i.e. a schedule slice. Usually, there are many matchings of maximal cardinality in B_S. Therefore, the matching algorithm adds a secondary objective function and consecutively solves weighted bipartite maximum cardinality matching problems. A number of matching algorithms have been proposed in the literature depending on how to determine a weighted maximum cardinality matching. From the objective functions used, we cite:

$$\text{minimize} \sum \{p_{ij} | J_{ij} \in M_S, M_S \text{ is a matching of maximal cardinality}\} \; [27] \tag{2.2}$$

$$\text{maximize} \sum \{p_{ij} | J_{ij} \in M_S, M_S \text{ is a matching of maximal cardinality}\} \; [27] \tag{2.3}$$

$$\text{minimize } \max\{p_{ij} | J_{ij} \in M_S, M_S \text{ is a matching of maximal cardinality}\} \; [27] \tag{2.4}$$

$$\text{maximize } \min\{p_{ij} | J_{ij} \in M_S, M_S \text{ is a matching of maximal cardinality}\} \; [27] \tag{2.5}$$

$$\text{minimize } \{\max\{p_{ij}\} - \min\{p_{ij}\} | J_{ij} \in M_S, M_S \text{ is a matching of maximal cardinality}\} \; [70] \tag{2.6}$$

Guéret and Prins have tested in [70] the above matching algorithms on randomly generated instances of different sizes, with integer processing times uniformly distributed in $[1, 99]$. For each instance, they constructed a solution by concatenating the schedule slices corresponding to the obtained matchings. The results showed that Function 2.4, i.e. a matching algorithm that computes maximum matchings in which the length of the longest operation is minimized (called min-max matching), gives the best schedules on average. Therefore, we apply in this phase only the min-max matching algorithm.

Algorithm 4 Variant of Dijsktra's algorithm for min-max path

Input: $G = (V, E)$, vertex $x \in V$, non-negative edge lengths $l\{.,.\}$.

```
1: dist(v) = ∞, ∀ v ∈ V − {x};
2: dist(x) = 0;
3: Insert all vertices in the set Q;
4: while Q ≠ ∅ do
5:    Remove from Q the vertex u whose value dist(u) is smallest among Q vertices;
6:    for all v such that {u, v} ∈ E do
7:       if dist(v) > max(dist(u), l{u, v}) then
8:          dist(v) = max(dist(u), l{u, v});
9:       end if
10:   end for
11: end while
```

Output: min-max paths from x to all the vertices of the graph.

To compute a maximum matching in which the length of the longest operation is minimized (see [125]), one tries to find from any min-max matching of cardinality k (eventually null!) an augmenting path in which the length of the longest edge is minimized. If augmentation is possible, a min-max matching of cardinality $(k+1)$ results. Recall that an augmenting path is a path that starts from and ends on free (unmatched) vertices and the edges of this path belong alternatively to the matching and not to the matching. A path in which the length of the longest edge is minimized (called min-max path) can be computed using a variant of Dijsktra's algorithm [65] by basically changing sums to max. This variant is summarized in Algorithm 4, where $dist(v)$ will eventually contain the length of the longest edge of the path from x to v.

The time complexity of this phase is $O(n^4 d)$, such that d is the maximum degree of the vertices of all the graphs B_S. Note that if $p_{ij} > 0$ for all $i = 1, \ldots, m$, $j = 1, \ldots, n$, then $d = max\{m, n\}$ and the complexity becomes $O(n^4 max\{m, n\})$.

2.4.2 Second phase

We start by sorting the schedule slices generated in phase one according to a priority rule PR_k, $(k = 1, \ldots, 8)$. The priority rules are based on three indicators that are computed for every slice C_l. These are:

- A length $L_l = max\{p_{ij} | J_{ij} \in C_l\}$.
- A conflict degree $Conf_l = \sum_{J_{ij} \in C_l} c_j$, where c_j is the degree of J_j in G.
- An agreement degree $Agree_l = \sum_{J_{ij} \in C_l} a_j$, where a_j is the degree of J_j in $\overline{G}$.

Regarding these indicators, the slices are sorted in a list L according to: (PR_1) decreasing order of $Conf_l/L_l$; (PR_2) increasing order of $Conf_l/L_l$; (PR_3) decreasing order of $Agree_l/L_l$; (PR_4) increasing order of $Agree_l/L_l$; (PR_5) decreasing order of L_l; (PR_6) increasing order of L_l; (PR_7) decreasing order of $Conf_l$; (PR_8) increasing order of $Conf_l$.

To construct a complete schedule we use a beam search technique that pursues a breadth first strategy to build its search tree. With each node, at level l of the search tree, is associated a partial schedule defined as a permutation of the first l slices of L. This node has a predecessor and $(l+1)$ successors, these successors are generated by inserting the slice C_{l+1} at the $(l+1)$ possible positions. The sequence of the slices of each node is scheduled as follows: the operations of the first slice start at time 0, then we insert the operations of the next slice to start as early as possible subject to the constraints that the operations of the same job and the operations of two conflicting jobs do not overlap. At each level of the search tree, this technique expands only the w most promising nodes, and discards the rest, where the integer w is called the beam width. We consider the following three variants of beam search.

- Beam1: At each level of the search tree we select the best w nodes, such that the selected nodes have the same predecessor.
- Beam2: For each of the w predecessors we select the best node, i.e. all nodes have different predecessors (as long as we do not have w predecessors the first variant is applied).
- Beam3: At each level of the search tree, we select the best w nodes from the whole set.

We still have to decide which nodes are the best at each level. We assign to each node a cost that is equal to the makespan of the partial schedule associated. If there exist ties, the leftmost node is selected. Note that the nodes are generated from left to right by inserting the slice consecutively starting from the first position.

Figure 2.4.1 shows an insertion of five slices using the three variants of beam search for $w = 2$. The numbers denote the costs of the corresponding nodes of a fictitious example.

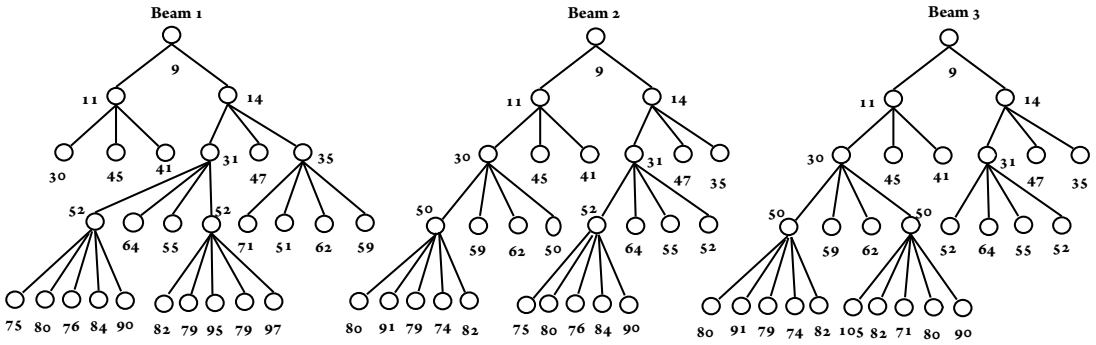


Figure 2.4.1: Variants of beam search.

Below, we describe a pruning rule based on the lower bounds discussed in Section 2.3. Let us consider a level l of the search tree, recall that the nodes of this level correspond to the same set of slices, that is the l first slices of L. Let $J(l)$ be the set of jobs of these slices and $G[J(l)]$ be the conflict graph induced by the jobs of $J(l)$. The processing time of each job of $G[J(l)]$ is equal to the sum of the processing times of the operations of this job belonging to the l first slices of L. A lower bound on the makespan of the jobs of $J(l)$ can now be computed using the lower bounds of Section 2.3, let LB^l be the best obtained lower bound. It is clear that, for any node at level l of the search tree, LB^l is a lower bound on the makespan of the partial schedule associated. For each set of nodes that belong to the same branch, let $w^{'}$ be the number of nodes of this branch to be selected for further branching at level $(l+1)$ ($w^{'}$ depends on the variant of beam search, e.g. $w^{'} = 1$ if Beam2 is selected). Therefore, if $w^{'}$ nodes of cost equals LB^l are evaluated, all the remaining nodes of this branch are discarded from the search tree without computing their costs.

The time complexity of this phase is $O(m|L|^3)$. However, we have at most $(m.n)$ slices. Thus, the total complexity is $O(n^3m^4)$, if all the processing times p_{ij} are positive.

The heuristic approach is summarized is Algorithm 5, where $SF = \{F_1^l, \ldots, F_w^l\}$ is the set of nodes selected for further branching after inserting the slice C_l. For each node F_u^l, the set SF_u^l represents the nodes generated by inserting the slice C_{l+1}. The time complexity of HmOSCn is $O(max\{n^3m^4, n^4max\{m,n\}\})$.

Algorithm 5 HmOSCn: heuristic approach for the m-machine OSC problem.

Input: Data of the OSC problem, PR_k, variant of beam search, w.

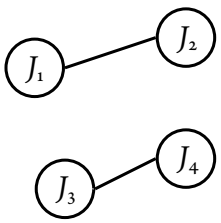
1: Partition the conflict graph G into a minimum number of independent sets using iteratively Algorithm 2 until no vertex remains;
2: **for** each independent set S **do**
3: Construct the weighted bipartite graph B_S;
4: Apply the min-max matching algorithm on B_S iteratively until no edge remains;
5: **end for**
6: Sort the resulting schedule slices in a list L according to the priority rule PR_k;
7: $SF = \{F_1^1\}$, $w^* = 1$;
8: **for** $l = 2$ to $|L|$ **do**
9: Compute the lower bound LB^l;
10: **for** each node $F_u^{l-1} \in SF$ such that $u = 1, \ldots, w^*$ **do**
11: update $w^{'}$ regarding the beam search variant chosen, $k = 1$;
12: **while** $k \leq l$ and $w^{'} \neq 0$ **do**
13: Insert the slice C_l in the position k;
14: Include the generated node in SF_u^{l-1} and compute its cost;
15: Call the pruning rule and update $w^{'}$ if necessarily; $k = k + 1$;
16: **end while**
17: **end for**
18: $N^* = \bigcup_{u=1}^{w^*} SF_u^{l-1}$, w^*=min$\{w, |N^*|\}$;
19: Select w^* nodes from N^* following the beam search variant chosen, which form the set SF;
20: **end for**
21: Choose a solution from SF with the best C_{max}, denoted by $F_{|L|}^*$;

Output: The final schedule $F_{|L|}^*$ and the corresponding C_{max}.

Example 2.1. *We illustrate Algorithm 5 by an example of 4 jobs to be processed on 4 machines, the processing times are given in Figure 2.4.2.(a) (Taillard instance [144]). The jobs are subject to the conflict constraints depicted in Figure 2.4.2.(b).*

	J_1	J_2	J_3	J_4
M_1	54	34	61	2
M_2	9	15	89	70
M_3	38	19	28	87
M_4	95	34	7	29

(a)

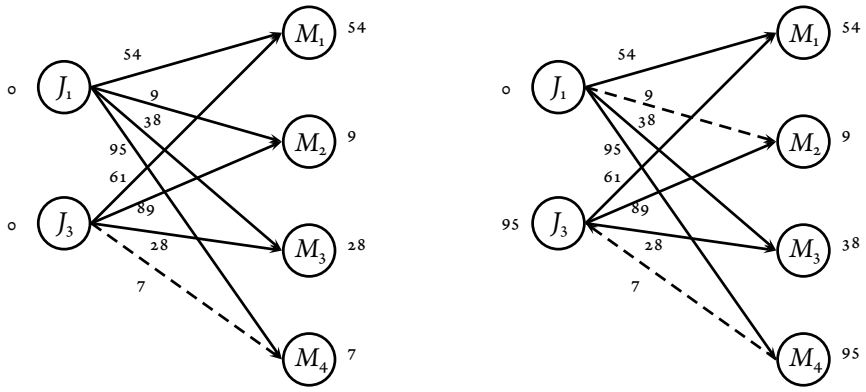


(b)

Figure 2.4.2: Example 2.1: (a) processing times. (b) conflict graph $G = (V, E)$.

Phase 1 *The decomposition of G gives two independent sets $S_1 = \{J_1, J_3\}$ and $S_2 = \{J_2, J_4\}$. We consider first the independent set S_1 and we construct the weighted bipartite graph B_{S_1} associated. We partition B_{S_1} into min-max matchings. Figure 2.4.3 shows how to compute the first min-max matching. The resulting schedule slice is C_{11}={(J_3, M_4),(J_1, M_2)} (dashed edges in Figure 2.4.3). We delete the edges of this matching from B_{S_1} and we iterate this process until no edge remains. The obtained schedule slices are: C_{11}={(J_3, M_4),(J_1, M_2)}, C_{12}={(J_3, M_3),(J_1, M_1)}, C_{13}={(J_3, M_1),(J_1, M_3)}, C_{14}={(J_3, M_2),(J_1, M_4)}.*

We repeat the same steps for S_2 and the resulting schedule slices are: C_{21}={(J_2, M_2),(J_4, M_1)}, C_{22}={(J_2, M_3),(J_4, M_4)}, C_{23}={(J_2, M_1),(J_4, M_2)}, C_{24}={(J_2, M_4),(J_4, M_3)}.



First iteration. **Second iteration.**

Figure 2.4.3: Example 2.1: computing a min-max matching in B_{S_1}.

Phase 2 *We use for this example the priority rule PR_1: $PR_1(C_{11}) = \frac{2}{9} = 0.222$, $PR_1(C_{12}) = \frac{2}{54} = 0.037$, $PR_1(C_{13}) = \frac{2}{61} = 0.032$, $PR_1(C_{14}) = \frac{2}{95} = 0.021$, $PR_1(C_{21}) = \frac{2}{15} = 0.133$, $PR_1(C_{22}) = \frac{2}{29} = 0.069$, $PR_1(C_{23}) = \frac{2}{70} = 0.028$, $PR_1(C_{24}) = \frac{2}{87} = 0.023$. Thus, the insertion order is: C_{11}, C_{21}, C_{22}, C_{12}, C_{13}, C_{23}, C_{24}, C_{14}.*

Let us consider the Beam2 variant with $w = 2$. We denote by $SF_u^l = \{uF_1^{l+1}, \ldots, uF_{l+1}^{l+1}\}$ the set of nodes generated at level $l+1$ from node F_u^l and $C_{max}(F_u^l)$ is the makespan of the partial schedule associated with F_u^l, i.e. the cost of node F_u^l, $(u = 1, \ldots, w)$. Before we start the construction of the schedule, let us compute the lower bound LB of the problem of this example. We first apply the first lower bound of Section 2.3. In the weighted agreement graph associated with G, we have two independent sets. The independent set with the maximum weight is composed of J_3 and J_4 and the weight of this set is 373. Regarding the lower bound LB_3 of Section 2.3, we have $LB_3 = \max\{\max_{0 \leqslant j \leqslant n}\{\sum_{i=1}^{m} p_{ij}\}, \max_{0 \leqslant i \leqslant m}\{\sum_{j=1}^{n} p_{ij}\}\} = \max\{151, 183, 172, 165, 196, 102, 185, 188\} = 196$. Then, $LB = \max\{373, 196\} = 373$. Now, we insert the schedule slices one by one as follows:

- $SF = \{F_1^1\} = (C_{11})$, $C_{max}(F_1^1) = 9$, $SF_1^1 = \{1F_1^2, 1F_2^2\}$.

- $1F_1^2 = (C_{21}C_{11})$, $1F_2^2 = (C_{11}C_{21})$, $C_{max}(1F_1^2) = C_{max}(1F_2^2) = 24$. *Since* $w = 2$, *then* $F_1^2 = 1F_1^2$, $F_2^2 = 1F_2^2$ *and* $SF = \{F_1^2, F_2^2\}$.

- $LB^3 = 43$. *We start with* $SF_1^2 = \{1F_1^3, 1F_2^3, 1F_3^3\}$. *We have* $C_{max}(1F_1^3) = 43 = LB^3$, *then the nodes* $1F_2^3$ *and* $1F_3^3$ *are discarded and* $F_1^3 = 1F_1^3$. *Return to* $SF_2^2 = \{2F_1^3, 2F_2^3, 2F_3^3\}$. *We have* $C_{max}(2F_1^3) = 43$ *so* $2F_2^3$ *and* $2F_3^3$ *are discarded and* $F_1^3 = 2F_2^3$. $SF = \{F_1^3, F_2^3\}$. *Recall that* $F_1^3 = (C_{22}C_{21}C_{11})$ *and* $F_2^3 = (C_{22}C_{11}C_{21})$

- *Iterate this process until the last schedule slice.* $SF_1^7 = \{1F_1^8, 1F_2^8, 1F_3^8, 1F_4^8, 1F_5^8, 1F_6^8, 1F_7^8, 1F_8^8\}$, $C_{max}(1F_1^8) = 373 = LB^8 = LB$. *Therefore, the schedule associated with node* $1F_1^8$ *is optimal and all the remaining nodes of* SF_1^7 *and* SF_2^7 *are discarded.*

Figure 2.4.5 shows the beam search tree (where "D" indicates a discarded node), and the obtained schedule which is, in this case, optimal is given in Figure 2.4.4.

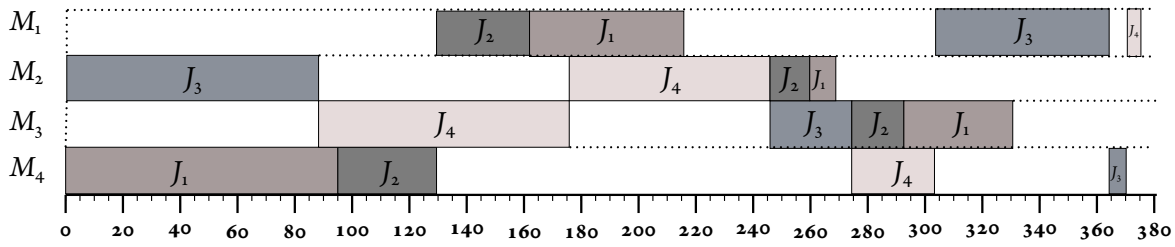


Figure 2.4.4: Example 2.1: an "optimal" schedule with $C_{max} = 373$.

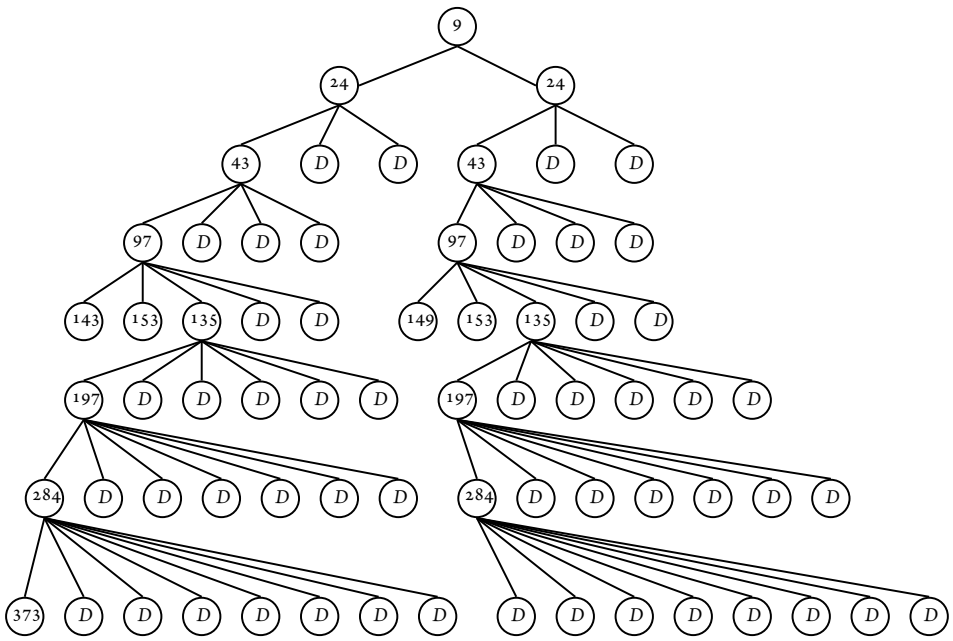


Figure 2.4.5: Example 2.1: beam search tree.

2.5 Computational experiments

We conducted a computational experiment where the above heuristic approach and lower bounds were coded in C++ language (using Microsoft Visual Studio 2012) and executed on a Pentium(R) Dual-Core PC with CPU@3.00 GHz and 1,00 Go RAM. All experiments are carried out on 3600 instances, which are constructed as follows. We consider the benchmark instances of the open shop problem given by Taillard [144], for each instance a set of conflict graphs is associated. The conflict graphs are generated using the $G(n,p)$ Erdős Rényi method (see Algorithm 6), which is the most intuitive and the most widely utilized graph generation method for scheduling problems (see e.g. [40]). Given n vertices, the $G(n,p)$ method generates a graph where each element of the $\binom{n}{2}$ possible edges is present with probability p. Note that the density of a graph $G = (V,E)$ measures how many edges are in set E compared to the maximum possible number of edges between vertices in set V. Thus, when the value of p increases, the graph density increases too.

Algorithm 6 $G(n,p)$ Erdős Rényi method [52]

Input: $n \in N, p \in [0,1]$.

```
1:  Let A be an adjacency matrix n × n initialized as the zero matrix ;
2:  for all i = 1 to n. do
3:     for all j = 1 to i. do
4:        if Random()<p then
5:           A[i][j] = 1;
6:        else
7:           A[i][j] = 0;
8:        end if
9:     end for
10: end for
```

Output: The graph represented by A.

Taillard instances [144] can be classified into six classes for each value of n: $n \in \{4,5,7,10,15,20\}$ with $m = n$. Each class contains 10 particularly hard instances. For each instance, 20 conflict graphs are generated for each value of p, $p \in \{0.2, 0.5, 0.8\}$. All the generated instances are classified into six classes and they are referred to as: 4OSC4, 5OSC5, 7OSC7, 10OSC10, 15OSC15 and 20OSC20, where mOSCn represents the class of instances with m machines and n jobs. The performance of the proposed heuristic variants is evaluated using the following criteria.

- APD(%): average percentage deviation from the best lower bound. The percentage deviation is obtained by the following formula: $\frac{C_{max} - LB}{LB} \times 100$, where C_{max} is the makespan obtained by a given heuristic variant and LB is the best lower bound.
- Nopt: number of times a given heuristic variant coincides with the best lower bound, i.e. with the optimal solution value.

- CPU: average CPU time in seconds.

The results of the simulation are summarized in Tables 2.5.1-2.5.6. The bold values in these tables indicate the best results obtained for each set mOSCn with fixed : p, beam search variant and w. Whereas, the underlined values represent the best results obtained for each set mOSCn with a fixed p. Figure 2.5.1 shows the best results obtained by Beam2 and Beam3 according to APD and Table 2.5.7 recapitulates the best results obtained by the 24 heuristic variants for all the instance classes.

Table 2.5.1: HmOSCn: experimental results for 4OSC4.

4OSC4				Heuristics							
				PR_1	PR_2	PR_3	PR_4	PR_5	PR_6	PR_7	PR_8
$p = 0.2$	Beam2		APD(%)	1.465	**1.177**	1.591	1.311	**1.177**	1.479	1.400	1.432
		$w = 8$	Nopt	156	**162**	157	154	**162**	154	156	157
			CPU(s)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
			APD(%)	1.402	**1.177**	1.551	1.213	**1.177**	1.403	1.249	1.268
		$w = 20$	Nopt	157	**162**	158	154	**162**	156	156	157
			CPU(s)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
	Beam3		APD(%)	1.400	**1.177**	1.277	1.398	**1.177**	1.420	1.571	1.603
		$w = 8$	Nopt	154	**162**	157	158	**162**	154	153	152
			CPU(s)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
			APD(%)	1.371	**1.177**	1.198	1.207	**1.177**	1.273	1.230	1.328
		$w = 20$	Nopt	158	**162**	161	161	**162**	158	160	160
			CPU(s)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
$p = 0.5$	Beam2		APD(%)	0.536	**0.153**	0.437	0.320	0.159	0.449	0.768	0.648
		$w = 8$	Nopt	174	**187**	177	184	**187**	174	176	181
			CPU(s)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
			APD(%)	0.353	**0.112**	0.321	0.243	0.135	0.355	0.613	0.577
		$w = 20$	Nopt	181	**191**	180	185	190	177	182	183
			CPU(s)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
	Beam3		APD(%)	0.382	0.102	0.280	0.179	**0.079**	0.273	0.651	0.623
		$w = 8$	Nopt	179	189	184	189	**191**	180	180	180
			CPU(s)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
			APD(%)	0.199	0.089	0.271	0.126	**0.067**	0.244	0.331	0.270
		$w = 20$	Nopt	186	191	185	190	**192**	184	188	187
			CPU(s)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
$p = 0.8$	Beam2		APD(%)	0.304	0.129	0.176	0.191	0.175	0.196	0.194	**0.082**
		$w = 8$	Nopt	185	194	192	189	192	185	193	**196**
			CPU(s)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
			APD(%)	0.245	0.129	0.101	0.191	0.175	0.182	0.099	**0.067**
		$w = 20$	Nopt	186	194	194	189	192	186	**197**	**197**
			CPU(s)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
	Beam3		APD(%)	0.133	0.089	0.149	0.107	0.095	0.155	0.181	**0.084**
		$w = 8$	Nopt	187	194	195	191	189	192	194	**196**
			CPU(s)	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
			APD(%)	0.133	0.089	0.107	0.107	0.091	0.161	0.177	**0.084**
		$w = 20$	Nopt	187	194	**196**	191	195	192	**196**	**196**
			CPU(s)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001

Table 2.5.2: HmOSCn: experimental results for 5OSC5.

5OSC5				Heuristics							
				PR_1	PR_2	PR_3	PR_4	PR_5	PR_6	PR_7	PR_8
$p = 0.2$	Beam1		APD(%)	3.090	2.606	3.225	2.882	**2.601**	2.967	3.641	3.599
		$w = 8$	Nopt	140	**147**	142	142	**147**	139	146	142
			CPU(s)	0.001	0.001	0.002	0.001	0.001	0.001	0.001	0.002
			APD(%)	2.777	**2.605**	3.110	2.784	2.634	2.827	3.320	3.283
		$w = 20$	Nopt	**151**	150	145	148	150	146	147	143
			CPU(s)	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
	Beam2		APD(%)	2.989	**2.546**	3.113	2.850	2.557	2.822	3.338	3.428
		$w = 8$	Nopt	145	**161**	148	150	159	149	150	146
			CPU(s)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
			APD(%)	2.875	2.484	2.955	2.733	**2.481**	2.700	3.104	3.188
		$w = 20$	Nopt	149	**165**	153	151	164	156	153	149
			CPU(s)	0.003	0.003	0.004	0.002	0.003	0.003	0.003	0.004
	Beam3		APD(%)	2.833	2.567	2.972	2.662	**2.538**	2.805	3.725	3.551
		$w = 8$	Nopt	154	156	151	155	**159**	152	149	145
			CPU(s)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
			APD(%)	2.648	**2.518**	2.796	2.600	2.519	2.619	3.443	3.352
		$w = 20$	Nopt	158	162	156	159	**164**	158	150	145
			CPU(s)	0.003	0.003	0.003	0.002	0.003	0.003	0.003	0.003
$p = 0.5$	Beam1		APD(%)	1.231	0.892	1.442	0.959	**0.765**	0.987	2.516	1.819
		$w = 8$	Nopt	149	142	142	143	**151**	146	147	141
			CPU(s)	0.003	0.003	0.003	0.003	0.003	0.003	0.003	0.003
			APD(%)	0.470	0.265	0.600	0.349	**0.247**	0.577	0.608	0.734
		$w = 20$	Nopt	178	**180**	169	177	**180**	173	172	161
			CPU(s)	0.006	0.005	0.006	0.005	0.005	0.005	0.006	0.005
	Beam2		APD(%)	1.130	**0.605**	1.292	0.730	0.630	0.984	2.205	1.978
		$w = 8$	Nopt	158	174	162	166	**177**	159	162	164
			CPU(s)	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
			APD(%)	1.020	**0.533**	1.031	0.679	0.560	0.838	1.975	1.893
		$w = 20$	Nopt	164	179	170	168	**180**	165	166	165
			CPU(s)	0.006	0.005	0.006	0.004	0.005	0.005	0.006	0.005
	Beam3		APD(%)	1.097	**0.554**	1.159	0.654	0.611	1.015	2.331	1.714
		$w = 8$	Nopt	162	170	161	**175**	167	162	153	165
			CPU(s)	0.002	0.002	0.003	0.002	0.002	0.002	0.002	0.002
			APD(%)	1.030	0.507	0.839	0.661	**0.480**	0.741	2.067	1.293
		$w = 20$	Nopt	166	175	165	171	**176**	162	163	166
			CPU(s)	0.005	0.004	0.005	0.004	0.004	0.005	0.005	0.005
$p = 0.8$	Beam1		APD(%)	0.464	0.283	0.387	0.261	**0.171**	0.348	0.650	0.183
		$w = 8$	Nopt	162	143	170	153	165	163	183	**186**
			CPU(s)	0.003	0.003	0.003	0.003	0.003	0.003	0.002	0.003
			APD(%)	0.126	0.092	0.128	0.097	**0.064**	0.158	0.100	0.147
		$w = 20$	Nopt	188	188	187	188	188	184	**196**	192
			CPU(s)	0.008	0.007	0.007	0.006	0.007	0.007	0.007	0.006
	Beam2		APD(%)	0.381	**0.045**	0.202	0.086	0.088	0.169	0.596	0.234
		$w = 8$	Nopt	179	**190**	189	189	**190**	185	183	**190**
			CPU(s)	0.002	0.002	0.002	0.001	0.002	0.002	0.001	0.002
			APD(%)	0.366	**0.040**	0.121	0.086	0.071	0.127	0.506	0.232
		$w = 20$	Nopt	181	**191**	**191**	189	**191**	185	189	190
			CPU(s)	0.005	0.005	0.005	0.004	0.005	0.005	0.004	0.005
	Beam3		APD(%)	0.202	0.080	0.103	0.095	**0.079**	0.148	0.453	0.113
		$w = 8$	Nopt	184	187	194	194	188	186	189	**195**
			CPU(s)	0.002	0.002	0.001	0.001	0.001	0.002	0.001	0.002
			APD(%)	0.150	0.076	0.079	**0.069**	0.077	0.150	0.382	0.091
		$w = 20$	Nopt	190	188	192	190	190	186	192	**195**
			CPU(s)	0.004	0.004	0.004	0.004	0.004	0.004	0.003	0.005

Table 2.5.3: HmOSCn: experimental results for 7OSC7.

7OSC7				Heuristics							
				PR_1	PR_2	PR_3	PR_4	PR_5	PR_6	PR_7	PR_8
$p = 0.2$	Beam2		APD(%)	5.563	5.299	6.211	6.200	**4.634**	5.323	6.298	7.576
		$w = 8$	Nopt	116	**135**	117	117	134	120	122	118
			CPU(s)	0.011	0.010	0.013	0.009	0.011	0.011	0.013	0.011
			APD(%)	5.335	5.154	5.884	6.120	**4.544**	5.124	6.022	7.406
		$w = 20$	Nopt	120	138	125	118	**142**	122	125	126
			CPU(s)	0.028	0.027	0.033	0.023	0.028	0.027	0.032	0.028
	Beam3		APD(%)	5.455	4.738	6.315	5.855	**4.532**	5.276	6.756	7.003
		$w = 8$	Nopt	125	133	112	122	**138**	122	119	118
			CPU(s)	0.011	0.009	0.012	0.008	0.010	0.011	0.011	0.010
			APD(%)	5.204	4.655	5.924	5.538	**4.529**	5.008	6.209	6.451
		$w = 20$	Nopt	132	132	124	134	**142**	129	123	123
			CPU(s)	0.025	0.023	0.029	0.020	0.024	0.025	0.028	0.025
$p = 0.5$	Beam2		APD(%)	2.327	1.833	3.173	2.607	**1.352**	1.805	4.766	4.294
		$w = 8$	Nopt	111	**142**	111	124	138	122	101	108
			CPU(s)	0.027	0.020	0.028	0.020	0.023	0.025	0.026	0.020
			APD(%)	2.158	1.740	2.809	2.498	**1.251**	1.681	4.503	4.250
		$w = 20$	Nopt	116	145	114	127	**150**	128	105	111
			CPU(s)	0.067	0.051	0.071	0.050	0.057	0.062	0.064	0.051
	Beam3		APD(%)	2.479	1.522	3.232	2.233	**1.377**	1.811	4.804	3.491
		$w = 8$	Nopt	111	**136**	101	123	134	128	101	114
			CPU(s)	0.025	0.018	0.026	0.019	0.019	0.022	0.023	0.018
			APD(%)	2.188	1.481	2.817	2.031	**1.323**	1.677	3.996	3.211
		$w = 20$	Nopt	116	139	104	133	**146**	133	106	120
			CPU(s)	0.056	0.043	0.063	0.044	0.044	0.054	0.056	0.045
$p = 0.8$	Beam2		APD(%)	0.933	0.484	0.934	0.723	**0.301**	0.573	2.602	1.211
		$w = 8$	Nopt	129	153	143	145	**163**	144	148	156
			CPU(s)	0.037	0.029	0.039	0.026	0.028	0.034	0.027	0.024
			APD(%)	0.856	0.440	0.836	0.698	**0.275**	0.546	2.584	1.173
		$w = 20$	Nopt	132	156	148	146	**165**	144	148	160
			CPU(s)	0.094	0.073	0.096	0.064	0.071	0.088	0.067	0.060
	Beam3		APD(%)	0.966	0.402	0.998	0.558	**0.354**	0.655	2.384	1.031
		$w = 8$	Nopt	134	156	141	145	**157**	138	143	**157**
			CPU(s)	0.036	0.025	0.035	0.024	0.027	0.033	0.023	0.021
			APD(%)	0.921	0.303	0.831	0.494	**0.263**	0.577	2.244	0.897
		$w = 20$	Nopt	136	161	146	149	**164**	141	146	162
			CPU(s)	0.084	0.060	0.080	0.059	0.060	0.079	0.057	0.050

2.5.1 Discussion

We study in this section the performance of the heuristic approach and we discuss the impact of the priority rules and the beam search variants.

We observed from the implementation that Beam1 is weaker compared to Beam2 and Beam3 except for instance class 5OSC5 (Table 2.5.2) for which Beam1 yields good results in particular for high and medium densities. Thus, the results corresponding to Beam1 have been removed except from Table 2.5.2. Beam2 and Beam3 are closely competitive according to APD and Nopt. However, Beam2 seems

Table 2.5.4: HmOSCn: experimental results for 10OSC10.

10OSC10				Heuristics							
				PR_1	PR_2	PR_3	PR_4	PR_5	PR_6	PR_7	PR_8
$p = 0.2$	Beam2		APD(%)	6.523	5.864	7.310	7.015	**4.936**	6.086	8.183	9.067
		$w = 8$	Nopt	75	101	76	82	**110**	80	68	75
			CPU(s)	0.128	0.099	0.130	0.090	0.110	0.112	0.131	0.101
			APD(%)	6.175	5.680	6.991	6.817	**4.811**	5.810	7.783	8.710
		$w = 20$	Nopt	82	111	83	84	**116**	82	75	86
			CPU(s)	0.314	0.248	0.324	0.225	0.273	0.279	0.324	0.252
	Beam3		APD(%)	6.643	5.261	7.858	6.377	**4.931**	6.221	8.567	8.345
		$w = 8$	Nopt	78	101	61	89	**103**	82	67	80
			CPU(s)	0.116	0.088	0.121	0.080	0.097	0.104	0.115	0.091
			APD(%)	6.277	5.060	7.486	6.007	**4.706**	5.724	8.099	8.076
		$w = 20$	Nopt	84	103	65	97	**107**	88	71	86
			CPU(s)	0.282	0.216	0.291	0.195	0.230	0.253	0.287	0.216
$p = 0.5$	Beam2		APD(%)	6.107	4.849	6.536	5.859	**3.772**	4.903	9.290	8.338
		$w = 8$	Nopt	53	78	53	65	**91**	61	41	54
			CPU(s)	0.385	0.297	0.392	0.282	0.315	0.345	0.376	0.264
			APD(%)	5.704	4.752	6.119	5.789	**3.537**	4.707	8.915	8.243
		$w = 20$	Nopt	58	81	57	65	**100**	66	43	56
			CPU(s)	0.968	0.747	0.982	0.708	0.792	0.870	0.947	0.663
	Beam3		APD(%)	6.370	4.291	7.058	5.245	**3.868**	5.129	9.751	7.556
		$w = 8$	Nopt	54	82	45	69	**87**	61	42	61
			CPU(s)	0.350	0.274	0.362	0.270	0.297	0.323	0.344	0.239
			APD(%)	5.991	3.975	6.731	4.777	**3.562**	4.768	9.256	6.916
		$w = 20$	Nopt	54	86	46	75	**91**	67	46	60
			CPU(s)	0.852	0.669	0.879	0.656	0.695	0.768	0.845	0.595
$p = 0.8$	Beam2		APD(%)	1.919	1.272	2.076	1.510	**1.053**	1.546	4.418	2.413
		$w = 8$	Nopt	88	111	86	102	**122**	84	88	96
			CPU(s)	0.674	0.470	0.670	0.442	0.515	0.626	0.466	0.388
			APD(%)	1.814	1.249	1.905	1.501	**0.943**	1.495	4.382	2.347
		$w = 20$	Nopt	92	114	92	103	**130**	86	88	98
			CPU(s)	1.665	1.161	1.654	1.098	1.261	1.551	1.157	0.958
	Beam3		APD(%)	2.009	1.157	2.379	1.363	**1.140**	1.584	4.348	2.200
		$w = 8$	Nopt	77	109	72	105	**117**	93	90	107
			CPU(s)	0.639	0.445	0.658	0.403	0.478	0.566	0.411	0.346
			APD(%)	1.858	1.118	2.254	1.276	**1.032**	1.481	4.201	2.150
		$w = 20$	Nopt	85	116	80	111	**119**	93	89	111
			CPU(s)	1.517	1.075	1.513	0.971	1.143	1.382	1.021	0.838

to work slightly better than Beam3 especially for Nopt and for instances with medium and high densities (see Figures 2.5.1 and 2.5.2). Therefore, it is more efficient to consider w nodes at each level of the search tree having different predecessors, than the best ones of the same predecessor. As expected, for larger beam width w the results are better in general except for some instance classes.

By looking at the results of the priority rules, PR_5 seems to be more efficient than the remaining priority rules especially for large size instances. For smaller instances, PR_5, PR_2 and PR_8 perform well. However, PR_6, PR_4, PR_1, PR_3 and PR_7 have low performance since they give few good results especially with Beam1 and for relatively small instances. Thus, a slice with the longest length should be given a higher priority

Table 2.5.5: HmOSCn: experimental results for 15OSC15.

15OSC15				Heuristics							
				PR_1	PR_2	PR_3	PR_4	PR_5	PR_6	PR_7	PR_8
$p = 0.2$	Beam2		APD(%)	11.880	10.577	13.027	12.944	**8.628**	10.587	14.962	16.681
		$w = 8$	Nopt	33	47	33	39	**62**	41	29	35
			CPU(s)	1.852	1.557	1.917	1.414	1.726	1.724	1.936	1.420
			APD(%)	11.446	10.337	12.559	12.633	**8.336**	10.289	14.322	16.350
		$w = 20$	Nopt	35	50	34	45	**68**	44	31	36
			CPU(s)	4.620	3.893	4.791	3.529	4.309	4.302	4.833	3.536
	Beam3		APD(%)	12.458	9.698	13.819	11.729	**8.673**	10.456	15.753	15.460
		$w = 8$	Nopt	32	53	33	35	**58**	44	28	38
			CPU(s)	1.734	1.459	1.799	1.382	1.591	1.597	1.800	1.325
			APD(%)	11.954	9.477	13.198	11.142	**8.197**	10.113	15.189	14.620
		$w = 20$	Nopt	37	53	32	40	**63**	48	27	39
			CPU(s)	4.159	3.613	4.404	3.362	3.848	3.961	4.470	3.247
$p = 0.5$	Beam2		APD(%)	11.500	9.774	12.265	11.601	**7.509**	9.845	16.590	15.950
		$w = 8$	Nopt	13	21	9	12	**27**	17	9	10
			CPU(s)	6.728	5.465	6.826	5.148	6.116	6.410	6.663	4.596
			APD(%)	11.043	9.497	11.768	11.498	**7.173**	9.520	16.147	15.733
		$w = 20$	Nopt	14	23	11	13	**30**	18	9	10
			CPU(s)	16.838	13.629	17.046	12.867	15.332	16.075	16.660	11.496
	Beam3		APD(%)	12.249	9.106	12.977	10.708	**7.645**	10.035	17.117	15.257
		$w = 8$	Nopt	14	20	8	15	**21**	12	7	10
			CPU(s)	6.552	5.457	6.616	5.129	6.071	6.158	6.493	4.494
			APD(%)	11.715	8.508	12.689	9.940	**7.265**	9.537	16.163	14.261
		$w = 20$	Nopt	13	18	11	16	**27**	15	6	12
			CPU(s)	15.848	13.948	15.936	12.918	14.683	14.943	16.056	11.184
$p = 0.8$	Beam2		APD(%)	3.953	2.951	4.452	3.832	**2.142**	3.061	8.853	5.932
		$w = 8$	Nopt	23	44	22	34	**65**	30	21	33
			CPU(s)	16.700	13.170	17.274	12.340	13.882	16.712	13.497	9.935
			APD(%)	3.820	2.889	4.287	3.826	**2.034**	3.002	8.726	5.902
		$w = 20$	Nopt	25	50	24	34	**67**	30	21	33
			CPU(s)	42.929	33.782	44.314	31.738	36.063	42.955	34.652	25.609
	Beam3		APD(%)	4.281	2.763	4.898	3.581	**2.415**	3.206	8.937	5.609
		$w = 8$	Nopt	22	42	19	33	**58**	35	20	36
			CPU(s)	16.147	12.970	16.799	12.167	13.712	16.294	13.080	9.397
			APD(%)	4.125	2.491	4.893	3.266	**2.258**	3.038	8.725	5.362
		$w = 20$	Nopt	20	47	17	35	**56**	35	19	36
			CPU(s)	40.679	32.259	41.277	30.588	34.050	39.586	33.197	23.027

for large instances. For relatively small instances, the longest slice with the lowest conflict degree should be selected first.

As it is observed from Table 2.5.7, while increasing the density of the conflict graph, the APD decreases significantly. This emerges mainly from the facts that: (a) the lower bounds are tight; (b) the number of slices computed in phase one increases which increases the number of partial schedules evaluated; (c) the set of feasible schedules is reduced.

From Table 2.5.7, the overall results were promising. For instances with at most 7 jobs and 7 machines, we have on average less than 0.123% deviation from the best lower bound and more than 93% of the instances were solved optimally for $p = 0.8$. Whereas for $p = 0.5$, we have less than 0.521% deviation

Table 2.5.6: HmOSCn: experimental results for 20OSC20.

20OSC20				Heuristics							
				PR_1	PR_2	PR_3	PR_4	PR_5	PR_6	PR_7	PR_8
$p = 0.2$	Beam2		APD(%)	17.294	16.320	18.261	19.033	**12.584**	15.756	21.013	24.145
		$w = 8$	Nopt	19	21	15	17	**38**	19	14	10
			CPU(s)	13.238	10.524	13.326	9.607	11.871	12.005	13.890	9.310
			APD(%)	16.835	16.035	17.792	18.638	**12.216**	15.330	20.533	23.827
		$w = 20$	Nopt	19	22	16	18	**39**	23	15	13
			CPU(s)	32.826	26.130	33.179	23.811	29.497	29.792	34.449	23.124
	Beam3		APD(%)	18.123	15.319	19.127	18.002	**12.474**	15.557	21.833	23.151
		$w = 8$	Nopt	13	20	10	19	**30**	21	12	11
			CPU(s)	12.754	10.274	12.814	9.278	11.500	11.442	13.403	8.862
			APD(%)	17.781	14.691	18.837	17.055	**11.855**	14.948	21.213	21.987
		$w = 20$	Nopt	14	24	12	19	**35**	23	10	13
			CPU(s)	31.415	25.447	31.783	23.104	28.344	27.734	33.289	21.508
$p = 0.5$	Beam2		APD(%)	16.840	14.394	17.448	16.449	**11.437**	14.923	21.358	21.721
		$w = 8$	Nopt	4	6	4	4	**9**	3	2	4
			CPU(s)	49.316	40.102	49.572	37.825	44.698	46.692	49.419	34.056
			APD(%)	16.718	14.355	17.361	16.425	**11.363**	14.857	21.214	21.673
		$w = 10$	Nopt	4	6	4	4	**9**	3	2	4
			CPU(s)	62.102	50.414	62.380	47.591	56.290	58.861	62.257	42.893
	Beam3		APD(%)	17.471	13.854	18.232	15.532	**11.789**	15.067	21.882	21.145
		$w = 8$	Nopt	3	5	4	4	**7**	5	1	3
			CPU(s)	48.152	40.491	48.439	38.290	45.400	45.791	48.574	33.524
			APD(%)	17.265	13.646	18.144	15.515	**11.513**	14.846	21.799	20.838
		$w = 10$	Nopt	3	5	3	4	**10**	5	1	3
			CPU(s)	60.394	50.603	60.345	47.802	55.390	56.982	60.778	42.139
$p = 0.8$	Beam2		APD(%)	7.887	6.168	8.528	7.525	**4.787**	6.232	14.601	11.286
		$w = 3$	Nopt	4	13	3	5	**16**	7	4	4
			CPU(s)	55.384	45.321	54.677	43.025	49.393	56.506	47.719	35.854
			APD(%)	7.773	6.146	8.443	7.523	**4.739**	6.190	14.540	11.270
		$w = 4$	Nopt	4	13	3	5	**16**	7	4	4
			CPU(s)	75.198	61.185	73.930	58.130	66.815	76.436	64.492	47.562
	Beam3		APD(%)	8.027	6.096	8.925	7.391	**5.019**	6.219	14.617	11.079
		$w = 3$	Nopt	4	11	3	5	**14**	8	4	3
			CPU(s)	55.237	45.824	54.509	42.862	49.105	56.123	47.308	34.827
			APD(%)	8.081	6.005	8.898	7.334	**4.949**	6.251	14.673	11.100
		$w = 4$	Nopt	4	10	3	5	**16**	7	4	4
			CPU(s)	73.128	60.873	72.331	57.001	65.252	74.867	62.770	46.048

from the best lower bound and 87% of the instances were solved optimally. For $p = 0.2$, the average makespan is still quite good: less than 2.729% deviation from the best lower bound and about 78% of the instances were solved optimally. Considering all the instance classes, on average more than 55% of the instances were solved optimally. Furthermore, the average deviation from the best lower bound is about 1.347% for $p = 0.8$, 3.93% for $p = 0.5$ and 4.706% for $p = 0.2$. The average CPU time required is about 5.408 seconds for $p = 0.2$, 12.079 seconds for $p = 0.5$ and 17.367 seconds for $p = 0.8$. As it is observed, the density of the conflict graph and the CPU time are directly proportional. This can be explained by the fact that, when the density of the conflict graph increases, the number of slices computed in phase one increases too which increases the complexity of the heuristic approach.

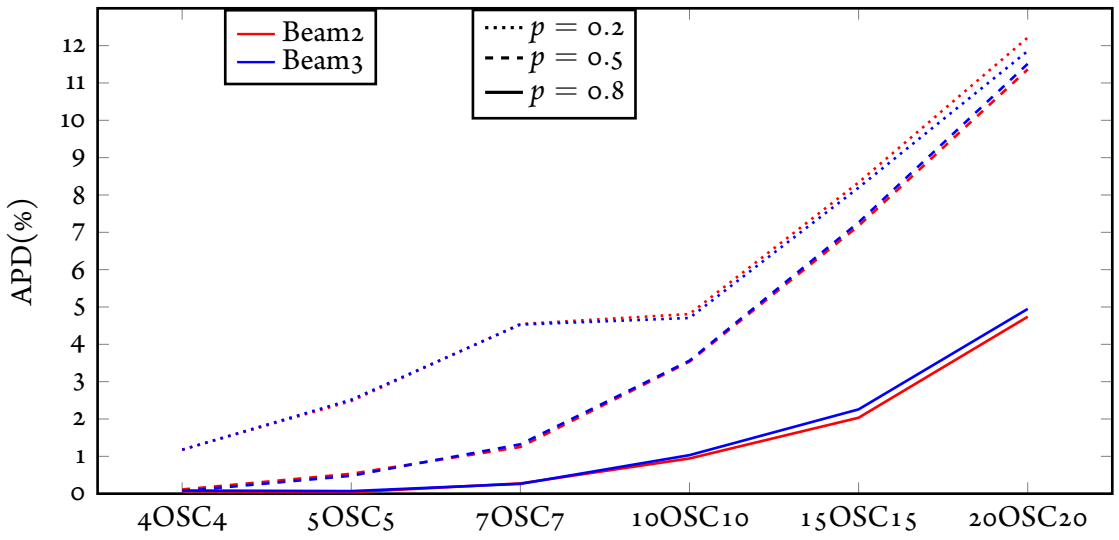


Figure 2.5.1: Comparison: Beam2 vs Beam3 with respect to APD(%).

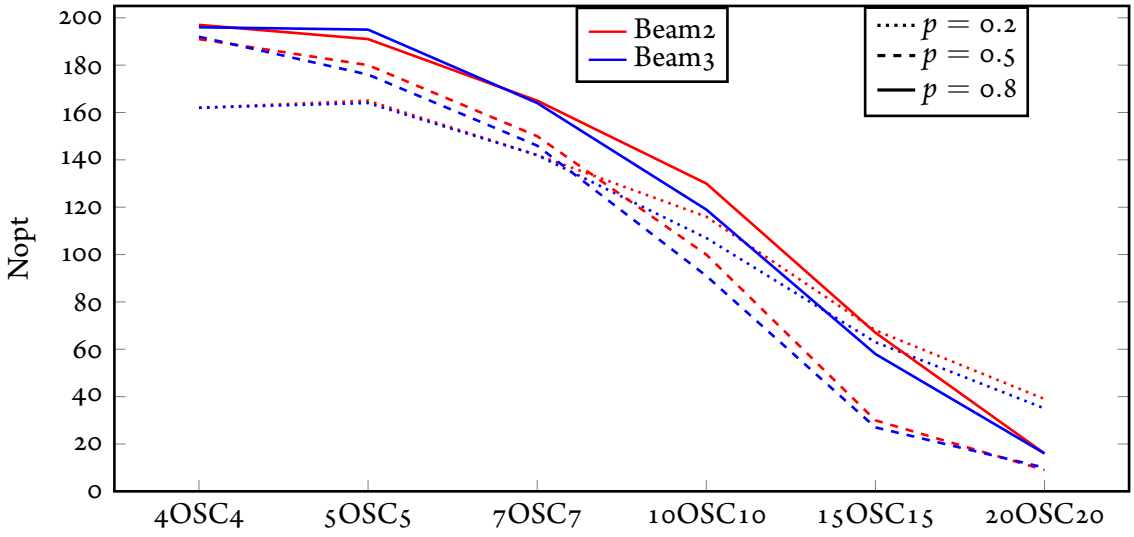


Figure 2.5.2: Comparison: Beam2 vs Beam3 with respect to Nopt.

2.6 Conclusion

This chapter considered the open shop scheduling problem with a conflict graph G under minimization of the makespan, such that adjacent jobs in G cannot be processed simultaneously on different machines. We began our theoretical study by proving that the OSC problem is polynomially equivalent to $O|res^t.11|C_{max}$. Then, two important negative results were discussed. We showed that the two-machine OSC problem with $p_{ij} \in \{1, 2, 3\}$ is NP-hard in the strong sense even when restricted to complements of bipartite graphs. The same result holds for the three-machine OSC problem with $p_{ij} = 1$. The reduction used is from the well known NP-hard problem of 3-DM [60]. From the later result, we proved that the partition into triangles in a particular 3-partite graph in which the set of edges has a partition into cycles of length 6 is NP-hard. After that, we shifted to a more positive vein. In particular, efficient algorithms were proposed for the two-machine OSC problem with $p_{ij} \in \{0, 1, 2\}$, and for the three-

Table 2.5.7: HmOSCn: best results of the benchmark problems.

Density		Instances					
		4OSC4	5OSC5	7OSC7	10OSC10	15OSC15	20OSC20
$p = 0.2$	APD(%)	1.177	2.481	4.529	4.706	8.197	11.855
	Nopt	162	165	142	116	68	39
	CPU(s)	0.000	0.003	0.024	0.230	3.848	28.344
$p = 0.5$	APD(%)	0.067	0.247	1.251	3.537	7.173	11.363
	Nopt	192	180	150	100	30	10
	CPU(s)	0.001	0.005	0.057	0.792	15.332	56.290
$p = 0.8$	APD(%)	0.067	0.040	0.263	0.943	2.034	4.739
	Nopt	197	196	165	130	67	16
	CPU(s)	0.001	0.005	0.060	1.261	36.063	66.815

machine OSC problem with $p_{ij} = 1$ and G being a complement of triangle-free graph. We also proved that by allowing preemption we make the two-machine OSC problem easy to solve. Furthermore, new complexity results of the open shop problem under resource constraints were established. Tables 2.6.1 and 2.6.2 summarize the results presented throughout this chapter about the complexity of the OSC problem and the open shop problem under resource constraints.

Table 2.6.1: Complexity results of the OSC problem.

OSC problem	Complexity	Reference
$m = 2, p_{1j} = p_{2j}$ and $p_{ij} \in \{1, 2, 3\}$, G a complement of a bipartite graph	strongly NP-Hard	Theorem 2.2
$m = 2, p_{ij} \in \{0, 1, 2\}$, $G = (V, E)$	$O(n^{2.5})$	Theorem 2.9
$m = 2, p_{ij} = 1, G = (V, E)$	$O(n^{2.5})$	Theorem 2.11
$m = 2$, preemption, $G = (V, E)$	$O(n^3)$	Theorem 2.12
$m = 3, p_{ij} = 1, G = (V, E)$	strongly NP-Hard	Theorem 2.4
$m = 3, p_{ij} = 1$, G a complement of a triangle-free graph	$O(n^{2.5})$	Theorem 2.13

Table 2.6.2: New complexity results of the open shop problem under resource constraints.

Problem	Complexity	Reference
$O2\|res^t.11, p_{ij} \in \{1, 2, 3\}\|C_{max}$	sNP-Hard	Corollary 2.3
$O2\|res^t.11, p_{ij} \in \{0, a, 2a\}\|C_{max}, a \geq 1$	polynomial	Corollary 2.10
$O3\|res^t.11, p_{ij} = 1\|C_{max}$	sNP-Hard	Corollary 2.7

On the other hand, a two-phase heuristic approach was proposed for the general m-machine OSC problem. In the first phase, the set of operations is partitioned into a minimum number of slices, where each slice corresponds to a matching in a weighted bipartite graph, which is constructed from the conflict graph. In the second phase, the resulting slices are inserted one by one to make a complete schedule using a beam search technique. The quality of the obtained solutions is compared to three lower bounds. The experimental study, we conducted, showed that the overall results were promising even for difficult problems.

The greatest challenge to any thinker is stating the problem in a way that will allow a solution.

Bertrand Russell

3

Flow shop problem

In the previous chapter, we considered the problem of scheduling with conflict graph in the open shop system, we now shift our attention to the flow shop system and study it in a similar fashion. Recall that, besides the conflict constraints between the jobs, the operations of each job need to be processed in order, i.e. one may not begin processing operation J_{ij} until operation $J_{i-1,j}$ has been completed for $1 < i \leq m$. While in the open shop problem the order in which operations are processed is immaterial.

Thus, the flow shop scheduling problem with conflict graph is formulated as follows. Given are a set $\{J_j, j = 1, \ldots, n\}$ of n jobs, a set $\{M_i, i = 1, \ldots, m\}$ of m machines and a conflict graph $G = (V, E)$ over the jobs. Each job J_j consists of m operations J_{ij} $(i = 1, \ldots, m)$, where J_{ij} has to be processed on machine M_i for p_{ij} time units. The processing times p_{ij} are nonnegative numbers. All jobs have the same processing order through the machines, i.e. each job is processed first on M_1, next on M_2, and so on, and lastly on M_m. Each vertex in G represents a job and jobs that are represented by adjacent vertices in G are conflicting and cannot be processed simultaneously on different machines. A machine may process only one job at a time and a job may be processed by only one machine at a time. The objective is to find a schedule that minimises the maximum completion time (makespan). This problem is referred to as Flow Shop scheduling with Conflict graph (FSC for short) and can be solved in polynomial time if G is a clique (all the operations are processed in disjoint time intervals and $C_{max} = \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij}$) or if G is an independent set and the jobs are processed on two machines or even three if some conditions are satisfied (using the Johnson's rule [88], see Section 1.1.2 for the basic flow shop problem).

As before, our approach is to first theoretically study the problem to answer the main question: *can we give an efficient algorithm that produces an optimal schedule for the FSC problem*? This question is treated in three sections. Since conflicts between jobs may arise from shared resources, we start in Section 3.1 by studying the relation between the FSC problem and the flow shop problem under resource constraints. The previous complexity results of this later problem are also included in this section. Then, the complexity of the FSC problem is discussed in Section 3.2 for various relevant restricted cases: non-preemptive and preemptive cases, unit-time operations and arbitrary processing times, split graphs, complements of bipartite graphs, arbitrary conflict graphs, etc. Furthermore, new complexity results of the flow shop problem under resource constraints are derived. As mentioned in the first chapter, most of the literature on flow shop scheduling is limited to permutation schedules, which maintain the same job order on all machines. It is well known that, for $m \leqslant 3$, the search for the optimal schedule can be restricted to permutation schedules [88], but this restriction can be costly when $m \geqslant 4$ (see e.g. [129]). It is of interest then, to consider the question: *What can be said about the permutation schedules when introducing the conflict graph*? The answer to this question is discussed in Section 3.3.

Now, we complement our theoretical study by a description and analysis of algorithms for restricted cases as well as the general case. Section 3.5 presents several mathematical models for different version of the problem under study. A branch and bound algorithm for the case of unit-time operations is given in Section 3.6. In Section 3.7, two heuristic approaches are proposed for the general problem. The solutions they produce are compared to five lower bounds, that will be presented in Section 3.4, for large instances and to the optimal solutions obtained by the mathematical models for small instances. Finally, extensive computational experiments performed on a wide range of test problems are provided in Section 3.8 to measure the performance of the proposed algorithms.

3.1 Related works

To study the complexity of the FSC problem, we first establish a relation between this problem and the resource-constrained scheduling problem. For the differences between shop scheduling with conflict graph and shop scheduling under resource constraints, the reader is referred to Section 1.4.

Proposition 3.1 ([147]). *Problem $F|ConfG = (V, E)|C_{max}$ is polynomially equivalent to $F|res^t.11|C_{max}$.*

Proof. The proof is similar to that of Proposition 2.1. □

In Table 3.1.1, we summarize the previous complexity results of the flow shop problem under resource constraints.

Table 3.1.1: Previous complexity results of the flow shop under resource constraints.

Problem	Complexity	Reference
$F2\|res111\|C_{max}$	strongly NP-hard	[15]
$F2\|res111, p_{ij}=1\|C_{max}$	$O(n)$	[15]
$F2\|res122, p_{ij}=1\|C_{max}$	$O(n)$	[141]
$F2\|res1.., p_{ij}=1\|C_{max}$	$O(nlogn)$	[134]
$F2\|res2.., p_{ij}=1\|C_{max}$	strongly NP-hard	[133]
$F2\|res.11, p_{ij}=1\|C_{max}$	strongly NP-hard	[16]
$F3\|res111, p_{ij}=1\|C_{max}$	$O(n)$	[141]
$F3\|res1.., p_{ij}=1\|C_{max}$	strongly NP-hard	[134]
$Fm\|res1.1, p_{ij}=1\|C_{max}$	$O(n^c logn)$, where $c=2^{m+1}$	[17]
$F\|res111, p_{ij}=1\|C_{max}$	strongly NP-hard	[17]
$F2\|pmtn, res1.., p_{ij}=1\|C_{max}$	$O(nlogn)$	[134]
$F2\|pmtn, res2.., p_{ij}=1\|C_{max}$	strongly NP-hard	[133]
$F2\|pmtn, res111\|C_{max}$	open	-
$F3\|pmtn, res1.., p_{ij}=1\|C_{max}$	strongly NP-hard	[134]
$Fm\|pmtn, res1.1, p_{ij}=1\|C_{max}$	open	-

3.2 Complexity issues

We have seen in Section 1.1.2 that the basic flow shop problem is solvable in $O(nlogn)$-time for $m=2$ and also for $m=3$ under certain conditions [88], but it is NP-hard in the strong sense for $m \geq 3$ [61]. This leads us to the question: *will the flow shop problem with $m=2$ remain polynomial when we introduce the conflict graph?* Unfortunately, our results in this section show that it is unlikely that an efficient algorithm can be obtained for the two-machine FSC problem even when restricted to interesting graph classes including split graphs and complements of bipartite graphs. Furthermore, we consider the preemptive version in an effort to find polynomial cases, but again the results are not satisfactory. Then, we turn our attention to the case of unit-time operations, for which important results will be given and some polynomial time solvable cases will be presented. On the other hand, new complexity results of the flow shop problem under resource constraints will be derived.

Since the FSC problem is polynomially equivalent to $F|res^t.11|C_{max}$, we start by analysing the results of Table 3.1.1. Blazewicz et al.(1983)[15] proved that the problem $F2|res111|C_{max}$ is NP-hard in the strong sense. In the instance used in the reduction process, the operations of each job have the same requirement of the resource. Thus, the problem $F2|res^t111|C_{max}$ is NP-hard in the strong sense. However, for the problem $F2|res.11, p_{ij}=1|C_{max}$ (resp. $F|res111, p_{ij}=1|C_{max}$), the instance used in the reduction process is not an instance of $F2|res^t.11, p_{ij}=1|C_{max}$ (resp. $F|res^t111, p_{ij}=1|C_{max}$). In the later two instances, we can find two jobs in which two operations belonging to different machines are in conflict and the two others are not. Then, we cannot deduce complexity results of the FSC problem from these two later problems.

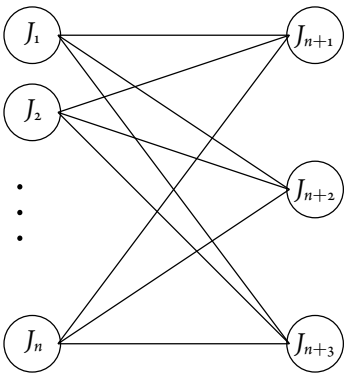
Now, we return to the problem $F2|res^t111|C_{max}$ and we construct the conflict graph $G = (V, E)$ associated. Let R_s be the required resource. To each job is associated a vertex in V, let $S = \{J_j \in V | R_s(J_j) = 0\}$ and $C = \{J_j \in V | R_s(J_j) = 1\}$. Clearly, $V = S \cup C$. The conflict edges are defined by the relation: $\{J_l, J_k\} \in E$ if and only if $R_s(J_k) + R_s(J_l) > 1$. Therefore, the subgraph $G[S]$ induced by the vertices of S is an independent set and the subgraph $G[C]$ induced by the vertices of C is a clique, and no edge exists between the sets S and C. Then, the obtained conflict graph is a complement of a complete split graph (that is a split graph). It follows that, for this class of conflict graphs, the two-machine FSC problem is NP-hard in the strong sense. In the next theorem, we show that, when the conflict graph is a complement of a bipartite graph, the two-machine FSC problem remains NP-hard.

Theorem 3.2 ([147]). *Problem $F2|ConfG = (V, E)|C_{max}$ is NP-hard in the ordinary sense for G being a complement of a complete bipartite graph.*

Proof. We prove this result by means of a reduction from the 2-Partition problem, which is known to be NP-complete in the ordinary sense (see for e.g. [60]). It is defined as follows: given a set $N = \{a_1, \ldots, a_n\}$ of n positive integers with $\sum_{j=1}^{n} a_j = 2B$ for some integer B, does there exist a partition of N into two subsets N_1 and N_2, such that $\sum_{a_j \in N_1} a_j = \sum_{a_j \in N_2} a_j = B$?

	M_1	M_2
$J_j, j = 1, \ldots, n$	a_j	a_j
J_{n+1}	B	B
J_{n+2}	0	B
J_{n+3}	B	0

(a)



(b)

Figure 3.2.1: Instance I of Theorem 3.2. (a) processing times. (b) agreement graph $\overline{G} = (V, \overline{E})$.

Given an arbitrary instance of 2-Partition, we construct an instance I of the two-machine FSC problem as follows: the set of jobs is composed by the partition jobs $J_1, \ldots, J_n$ denoted by P-jobs and three additional jobs J_{n+1}, J_{n+2} and J_{n+3}. Figure 3.2.1.(a) summarizes the processing time of each job J_j, $j = 1, \ldots, n+3$, while Figure 3.2.1.(b) presents the agreement graph $\overline{G}$ associated which is a complete bipartite graph.

We show that 2-Partition has a solution if and only if there exists a feasible schedule σ for I with $C_{max}(\sigma) \leqslant 4B$.

Suppose that 2-Partition has a solution and let N_1 and N_2 be the required subsets of N. We construct a feasible schedule σ for I as follows (see Figure 3.2.2). Machine M_1 starts with the P-jobs corresponding

to N_1, that are processed within $[0, B]$, followed by job J_{n+1}, which in turn is followed by the P-jobs corresponding to N_2, that are processed within $[2B, 3B]$, and finally M_1 finishes with job J_{n+3}. Machine M_2 starts with job J_{n+2}, followed by the P-jobs corresponding to N_1, that are processed within $[B, 2B]$, then followed by job J_{n+1} and finally finishes with the P-jobs corresponding to N_2, that are processed within $[3B, 4B]$. Observe that there is no idle time in the schedule σ, which satisfies the precedence constraints between operations and the conflict constraints between jobs, and $C_{max}(\sigma) = 4B$.

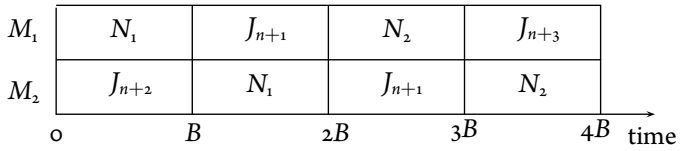


Figure 3.2.2: 2-Partition reduces to $F2|ConfG = (V, E)|C_{max}$.

Conversely, suppose there exists a feasible schedule σ for I with $C_{max}(\sigma) \leqslant 4B$. Since the total processing time of jobs on both machines is $4B$, then there is no idle time on machines M_1 and M_2 whatsoever, and $C_{max}(\sigma) = 4B$. We consider now the conflict constraints between the jobs of I. Since J_{n+1} agrees only with the P-jobs and since there is no idle time on both machines, it follows that J_{n+1} has to be processed simultaneously with a subset of P-jobs for B time units on each machine. Let O_1 and O_2 be the subsets of P-jobs to be processed simultaneously with J_{n+1} on M_1 and M_2 respectively. Similarly, the jobs J_{n+2} and J_{n+3} agree only with the P-jobs, the only way to avoid idle times is to process the unique operation of J_{n+2} (resp. J_{n+3}) simultaneously with a subset of P-jobs for B time units on M_1 (resp. M_2), this subset is denoted by O_3 (resp. O_4). Observe that the jobs of P-jobs are mutually in conflict and have to be processed in an interval of length $4B$. Thus, at any time a job of P-jobs is processed on one of the machines.

Let us introduce the following notations.

- BL_1 denotes the block of schedule σ composed by the operations of O_1 on M_1 in parallel with the operation of J_{n+1} on M_2.
- BL_2 denotes the block of schedule σ composed by the operations of O_2 on M_2 in parallel with the operation of J_{n+1} on M_1.
- BL_3 denotes the block of schedule σ composed by the operations of O_3 on M_1 in parallel with the operation of J_{n+2} on M_2.
- BL_4 denotes the block of schedule σ composed by the operations of O_4 on M_2 in parallel with the operation of J_{n+3} on M_1.

Because of the precedence constraints between the operations of J_{n+1}, BL_2 should be scheduled before BL_1. Moreover, the jobs of O_2 cannot be started on M_2 before finishing their processing on M_1. Thus,

BL_3 should be scheduled before BL_2. Finally, BL_4 has to be scheduled after BL_1 so that the jobs of O_1 can pass to M_2 to finish processing. Hence, the only possible sequence of blocks is (BL_3, BL_2, BL_1, BL_4). Let N_1 and N_2 be two subsets of N corresponding to the jobs of O_3 and O_1 respectively. The sets O_3 and O_1 are disjoint, then N_1 and N_2 are necessarily disjoint, we have $\sum_{a_j \in N_1} a_j = \sum_{a_j \in N_2} a_j = B$, and a partition of N follows. Observe that a P-job can be processed on M_1 in O_3 and on M_2 in O_4. However, this case cannot occur. Indeed, we have $T[O_2] = B$ (where $T[O_k]$ is the length of the time interval assigned to O_k) and for each P-job J_j we have $p_{1j} = p_{2j}$. This implies that $T[O_3] \geq B$. We already have that $T[O_3] = B$. Then, O_3 and O_2 correspond to the same set of jobs. As a consequence, O_1 and O_4 correspond also to the same set of jobs. Therefore, O_2 and O_4 yield the same partition of N obtained from O_3 and O_1. □

Now, we consider the preemptive version of the FSC problem to see if by allowing preemption we make the problem easy to solve. Unfortunately, the next theorem shows that the problem remains NP-hard even when restricted to a complement of a complete bipartite graph.

Theorem 3.3 ([147]). *Problem $F2|ConfG = (V, E), pmtn|C_{max}$ is NP-hard in the ordinary sense for G being a complement of a complete bipartite graph.*

Proof. The candidate used in the reduction process is the 2-Partition problem (see the proof of Theorem 3.2). We consider the instance I constructed in Theorem 3.2, which is a valid instance for the preemptive version. In what follows, we prove that 2-Partition has a solution if, and only if, our scheduling problem has a solution σ with $C_{max}(\sigma) \leqslant 4B$.

Suppose that 2-Partition has a solution and let N_1 and N_2 be the required subsets of N. Since all non preemptive schedules are also preemptive, the schedule σ of I obtained in Theorem 3.2 (see Figure 3.2.2) is also feasible for the preemptive version and has $C_{max}(\sigma) = 4B$.

Conversely, assume there exists a feasible schedule σ for I with $C_{max}(\sigma) \leqslant 4B$. Following the proof of Theorem 3.2, we construct the blocs BL_1, BL_2, BL_3 and BL_4 (Figure 3.2.3. Schedule 1). Since preemption is allowed, we distinguish the following cases:

- Preemption between BL_3 and BL_2, and between BL_1 and BL_4 (see Figure 3.2.3.Schedule 2).
- Preemption between BL_3 and BL_4 (see Figure 3.2.3.Schedule 3). In this schedule we have $T[O_3^1] < T[O_4^1] + T[O_2]$. However, for each P-job J_j we have $p_{1j} = p_{2j}$ and operation J_{2j} cannot be started before operation J_{1j} is completed. Thus, this case cannot occur.
- The other cases of preemption between the blocks are immediately excluded due to the precedence constraints between the operations of J_{n+1}.

Now, we consider the feasible schedules of I (Schedules 1 and 2 of Figure 3.2.3). Observe that O_1 and

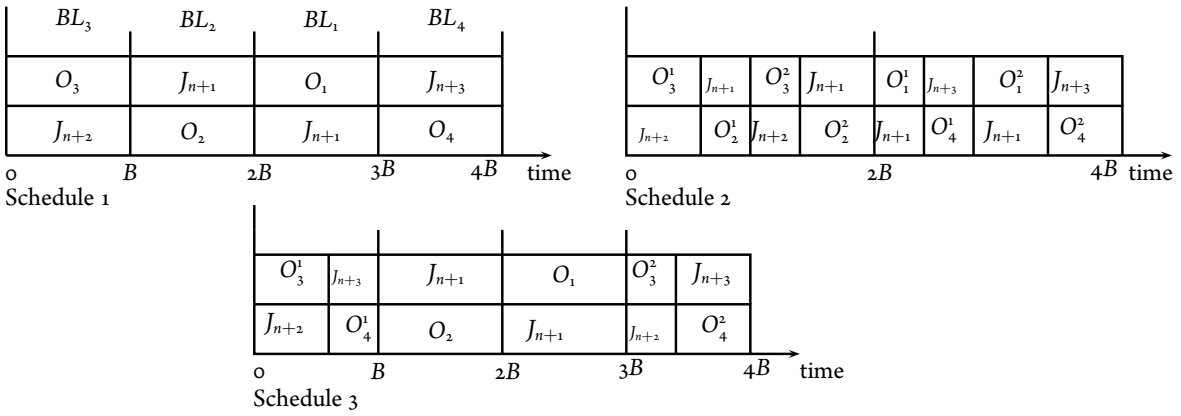


Figure 3.2.3: 2-Partition reduces to $F2|ConfG = (V, E), pmtn|C_{max}$.

O_3 (resp. O_2 and O_4) are not necessarily disjoint subsets i.e., a job of O_3 (resp. O_2) may be interrupted and restarted later in O_1 (resp. O_4). The following lemma shows that this case cannot occur.

Lemma 3.4. *If there exists a feasible schedule σ for I with $C_{max}(\sigma) = 4B$, then the subsets O_1 and O_3 are disjoint, and so are O_2 and O_4.*

Proof. We proceed by contradiction. Suppose there exists at least one job interrupted in O_3 and restarted later in O_1, then the operation of this job on M_2 is processed necessarily in O_4. Recall that for each P-job J_j we have $p_{1j} = p_{2j}$ and operation J_{2j} cannot be started before operation J_{1j} is completed. In the most favorable situation, where none of the jobs of O_2 is interrupted and restarted later in O_4, $T[O_3] - T[O_2]$ is equal to the processing time of the parts processed in O_3 of the interrupted jobs. Thus $T[O_3] > T[O_2]$, which contradicts the assumption that $T[O_3] = T[O_2] = B$.

Consider now O_2, suppose there exists at least one job interrupted in O_2 and restarted at a later time in O_4. It is clear that the processing of this job in O_3 has been already finished. Recall that $p_{1j} = p_{2j}$ for each P-job J_j and the jobs of O_2 cannot be started before finishing their processing in O_3. Thus, we have $T[O_3] > T[O_2]$, which contradicts the assumption that $T[O_3] = T[O_2] = B$.

Therefore, O_3 and O_1 are disjoint subsets and so are O_4 and O_2. □

Similarly to Theorem 3.2, we can show that O_3 and O_2 are of the same jobs, and so are O_1 and O_4. Let N_1 and N_2 be two subsets of N corresponding to the jobs of O_3 and O_1 respectively (or to O_2 and O_4 respectively). From Lemma 3.4, the sets O_3 and O_1 are disjoint and so are O_2 and O_4. Then, N_1 and N_2 are necessarily disjoint, we have $\sum_{a_j \in N_1} a_j = \sum_{a_j \in N_2} a_j = B$, and a partition of N is established. □

As far as we know, the complexity of $F2|pmtn, res.11|C_{max}$ is not known (see Table 3.1.1). The following corollary shows that this problem is NP-hard even when the operations of each job have the same requirement of each resource.

Corollary 3.5 ([147]). *Problem $F2|pmtn, res^t.11|C_{max}$ is NP-hard.*

Proof. Consequence of Proposition 3.1 and Theorem 3.3. □

The next theorem shows that even when all the operations have the same processing time, the two-machine FSC problem remains NP-hard.

Theorem 3.6 ([147]). *Problem $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ is NP-hard in the strong sense.*

Proof. We will use in the reduction process the Hamiltonian path problem, which is known to be NP-complete in the strong sense (see for e.g. [60]).

For an arbitrary instance of the Hamiltonian path problem, given by a graph $\overline{G} = (V, \overline{E})$ with n vertices, we build an instance I of the scheduling problem described above (in polynomial time) as follows: to each vertex $v_j \in V$ corresponds a job J_j of I with $p_{ij} = 1$ $(i = 1, 2)$. Two jobs of I are in conflict if, and only if, their corresponding vertices are not adjacent in $\overline{G}$. Observe that $\overline{G}$ is the agreement graph associated to I. In what follows we show that there exists a feasible schedule σ for I with $C_{max}(\sigma) \leq n+1$ if, and only if, $\overline{G}$ has a Hamiltonian path.

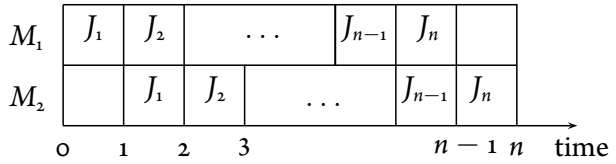


Figure 3.2.4: Hamiltonian path problem reduces to $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$.

Assume that $\overline{G}$ has a Hamiltonian path and let $p = \{v_1, e_1, v_2, \ldots, e_{n-1}, v_n\}$ be this path where e_k is an edge incident to the vertices v_k and v_{k+1} $(k = 1, \ldots, n-1)$. The desired schedule σ is constructed by processing the jobs $J_1, J_2, \ldots, J_n$, corresponding to $v_1, v_2, \ldots, v_n$ respectively, as shown in Figure 3.2.4. There is no interior idle times in the schedule σ, which satisfies the precedence constraints between the operations and the conflict constraints between the jobs. Therefore, the obtained schedule is feasible and $C_{max}(\sigma) = n + 1$.

Conversely, suppose there exists a feasible schedule σ for I with $C_{max}(\sigma) \leqslant n + 1$. The processing times of all the jobs sum up to n on each machine. Since all the jobs have unit-time operations and have to be processed first on M_1 and next on M_2, there must be an idle time of one unit on M_2 at the beginning and an idle time of one unit on M_1 at the end. Therefore, we have $n + 1 \leq C_{max}(\sigma)$ but $C_{max}(\sigma) \leqslant n + 1$ so $C_{max}(\sigma) = n + 1$. It follows that the job sequence on each machine is the same, otherwise additional idle times will occur in the schedule σ. By construction of the schedule, each job has to be processed simultaneously with one job on M_1 and another job on M_2, except the first and the last jobs of the sequence that have to be processed simultaneously with only one job. Suppose without loss of generality that the job sequence of σ is $J_1, J_2, \ldots, J_n$ corresponding to $v_1, v_2, \ldots, v_n$ respectively. This schedule must have the form of Figure 3.2.4. Since J_j and J_{j+1} are not in conflict, the vertices v_j and

v_{j+1} are adjacent ($j = 1, \ldots, n-1$). So, there exists a path that visits the vertices $v_1, v_2, \ldots,$ and v_n in this order exactly once. Consequently, $\overline{G}$ has a Hamiltonian path. □

We have seen that $F2|res.11, p_{ij} = 1|C_{max}$ is NP-hard in the strong sense (see Table 3.1.1). The following corollary shows that this problem remains NP-hard even if the operations of each job have the same requirement of each resource.

Corollary 3.7 ([147]). *Problem $F2|res^t.11, p_{ij} = 1|C_{max}$ is NP-hard in the strong sense.*

Proof. Consequence of Proposition 3.1 and Theorem 3.6. □

Remark 3.1 ([147]). *Following the proof of Theorem 3.6, we can observe that $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ is polynomially equivalent to the Minimum Path Cover problem (MPC) of the agreement graph $\overline{G}$. Recall that a path cover of a graph is a family of vertex-disjoint paths that covers all the vertices of this graph. The MPC problem is to find a path cover of minimum cardinality (see Section 1.2 for more details). Thus, our scheduling problem is NP-hard on all graph classes on which the MPC problem is NP-hard, including planar graphs, bipartite graphs, chordal graphs, chordal bipartite graphs and strongly chordal graphs. However, polynomial-time algorithms exist for trees, block graphs, interval graphs, cographs, bipartite distance-hereditary graphs, bipartite permutation graphs, cocomparability graphs and distance-hereditary graphs.*

Remark 3.2 ([147]). *Note that the proof of Theorem 3.6 remains valid when the processing times of the operations are equal to p (identical processing times).*

The NP-hardness of $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ can be proved using a different idea as follows.

Theorem 3.8 ([154]). *A feasible schedule for $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ with I idle times on each machine exists if and only if $\overline{G}$ has a path cover of cardinality I.*

Proof. First, observe that in any feasible schedule for $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$, the number of idle times on both machines is the same. We refer to a pair of operations scheduled simultaneously as a slice. Suppose there exists a feasible schedule σ for an instance of $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ with I idle times on each machine. It is clear that there exists $(n - I)$ slices in σ, this set is denoted by S. Let x be the number of jobs that belong to two different slices of S. Then, the number of jobs not in the slices of S is: $n - (2|S| - x)$, each one is considered as a path. Furthermore, the jobs in the slices of S can be partitioned into $(|S| - x)$ vertex-disjoint paths. In summary, this means that $\overline{G}$ has a path cover of $(|S| - x)$+ $(n - (2|S| - x))$=$n - |S|$=I paths.

Conversely, suppose that $\overline{G}$ has a path cover of I paths. Let $p = \{v_1, e_1, v_2, \ldots, e_{l-1}, v_l\}$ be one of these paths where e_i is an edge incident to the vertices v_i and v_{i+1} ($i = 1, \ldots, l-1$). Suppose without loose of generality that the job J_i corresponds to the vertex v_i. Then, the jobs J_1,. . ., J_l can be scheduled in this order without interior idle times similarly to the jobs of Figure 3.2.4. By concatenating all the partial

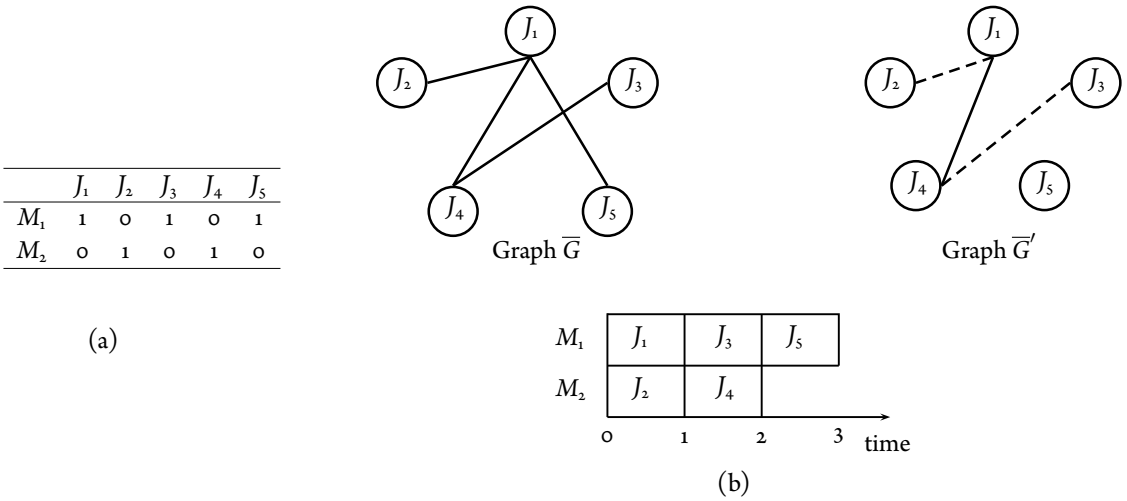
schedules of the I paths, with an idle time of one unit between two paths scheduled successively, we obtain a complete schedule with I idles times on each machine. □

Obviously, an optimal schedule for $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ with I idle times on each machine exists if and only if the minimum cardinality of a path cover of $\overline{G}$ is I. Since the MPC problem is NP-hard, our scheduling problem is also NP-hard.

Now, we discuss some special cases that can be solved in polynomial time. Blazewicz et al.(1983)[15] showed that the problem $F2|res111, p_{ij} = 1|C_{max}$ can be solved in $O(n)$-time. An algorithm much simpler can be proposed for $F2|res^t111, p_{ij} = 1|C_{max}$. The conflict graph associated to this later problem is similar to the one constructed for $F2|res^t111|C_{max}$ in the beginning of this section. An optimal schedule σ is obtained by scheduling the jobs of C successively in disjoint time intervals, which requires $2|C|$ time units. Then, we place the jobs of S inside the $|C|-1$ idle times induced by the jobs of C. If $|S| > |C|-1$, then the remaining jobs (i.e. $|S| - (|C| - 1)$ jobs) are scheduled starting from time $2C - 1$ without inducing interior idle times. Therefore

$$C_{max}(\sigma) = \begin{cases} 2|C| & \text{if } |S| < |C|, \\ |C| + |S| + 1 & \text{if } |S| \geq |C|. \end{cases}$$

In what follows, we discuss another special case that can be solved efficiently. Let I be an instance of the two-machine FSC problem in which each job has only one operation of one unit of processing time and let $\overline{G}$ be the agreement graph associated. $\overline{G}'$ is the partial graph where two vertices are adjacent if, and only if, their corresponding jobs are adjacent in $\overline{G}$ and they are of different machines. Note that the graph $\overline{G}'$ is bipartite since it can be written $\overline{G}' = (S_1 \cup S_2; \overline{E}')$, where the independent set S_i represents the jobs to be processed on machine M_i, $i = 1, \ldots, 2$.



	J_1	J_2	J_3	J_4	J_5
M_1	1	0	1	0	1
M_2	0	1	0	1	0

(a)

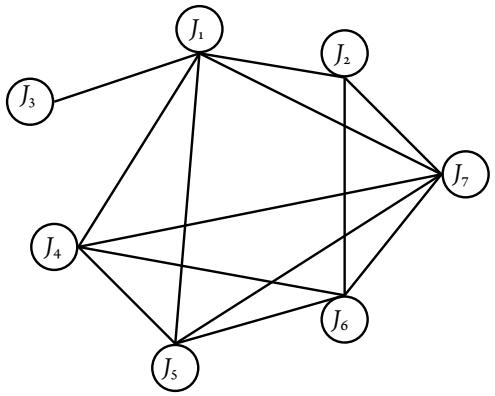
(b)

Figure 3.2.5: Application of the algorithm of Theorem 3.9. (a) processing times. (b) construction of $\overline{G}'$ and the optimal schedule.

Theorem 3.9 ([147]). *For an arbitrary conflict graph G, the two-machine FSC problem with $p_{ij} \in \{0,1\}$ and $p_{1j} + p_{2j} = 1$ is solvable in $O(n^{2.5})$-time and the optimal makespan is $(n - |M|)$, where M is a maximum matching in $\overline{G}'$.*

Proof. We claim that a maximum matching in $\overline{G}'$ yields an optimal schedule for I. Since $\overline{G}'$ is constructed in $O(n^2)$-time, and a maximal matching can be found in $O(n^{2.5})$, the theorem follows.

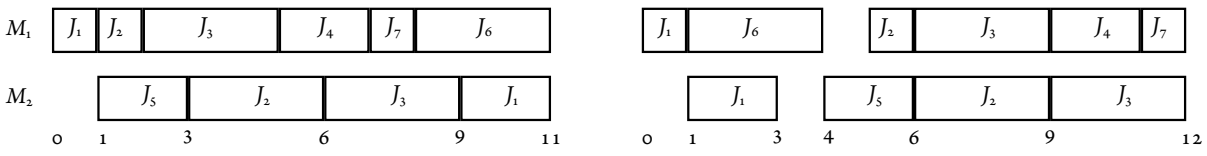
Let M be a maximum matching in $\overline{G}'$ (dashed edges in Figure 3.2.5). Each edge in M induces a schedule such that jobs incident to this edge are processed simultaneously, this requires $|M|$ time units to process all the jobs incident to the matching M. We are left with $(n - 2|M|)$ jobs that are mutually in conflict or of the same machine, otherwise the matching M is not maximal. These jobs have be processed independently in disjoint time intervals. Therefore, the optimal makespan value equals $|M| + (n - 2|M|) = (n - |M|)$. □



(a)

	J_1	J_2	J_3	J_4	J_5	J_6	J_7
M_1	1	1	3	2	0	3	1
M_2	2	3	3	0	2	0	0

(b)



(c) (d)

Figure 3.2.6: Instance of $F2|ConfG = (V,E)|C_{max}$. (a) conflict graph $G = (V,E)$. (b) processing times. (c) an optimal schedule. (d) an optimal permutation schedule.

3.3 Permutation vs. non permutation schedules

When searching for an optimal schedule of a set of jobs in a flow shop system, the question arises whether it suffices merely to determine a permutation in which the jobs traverse the entire system. For

$Fm||C_{max}$ with $m \leq 3$, the search for an optimal schedule can be restricted to permutation schedules [88], but this result can be very costly when $m \geq 4$ (see e.g. [129] and Section 1.1.2). Clearly, the conflict constraints make the problem much more difficult, so our emphasis, in this section, is put on the following question: *are the permutation schedules still dominant for $m \leq 3$ when considering the conflict constraints?*

For the case of arbitrary processing times, the permutation schedules are not dominant even for $m = 2$. Indeed, consider the following example (given in [154]) of 7 jobs that need to be processed on 2 machines, the processing times are given in Figure 3.2.6.(b). The jobs are subject to the conflict constraints depicted in Figure 3.2.6.(a). The optimal schedule is given, on M_1, by the sequence J_1, J_2, J_3, J_4, J_7 and J_6 (where no idle time occurs), and on M_2 by the sequence J_5, J_2, J_3 and J_1 for a makespan of value 11 (see Figure 3.2.6.(c)). The optimal permutation is given by J_1, J_6, J_5, J_2, J_3, J_4 and J_7 for a makespan of value 12 (see Figure 3.2.6.(d)), this schedule is obtained from the mathematical model (P_2) that will be discussed in Section 3.5.1.

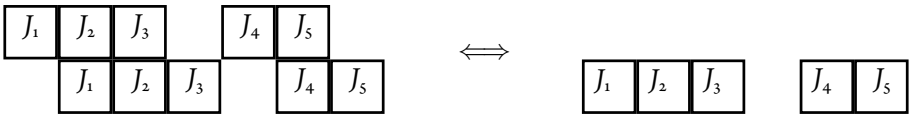


Figure 3.3.1: Transformation of an optimal schedule of $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$.

In the restricted case of unit-time operations, it can be observed from the proof of Theorem 3.6 that there exists an optimal no-wait permutation schedule for $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$. This schedule is obtained by processing each path of the minimum path cover of $\overline{G}$ as depicted in Figure 3.2.4, with an idle time of one unit between two successive paths. Therefore, an optimal schedule for $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ corresponds exactly to an optimal schedule of n unit-time jobs on one machine such that an idle time occurs between two conflicting jobs processed successively (see Figure 3.3.1).

3.4 Lower bounds

We present in this section five lower bounds on the makespan for $Fm|ConfG = (V, E), prmu|C_{max}$. Note that these lower bounds are useful for benchmarking heuristic solutions.

3.4.1 Precedence constraints relaxation-based lower bound [147]

The lower bounds of this section are obtained by relaxing the precedence constraints between operations. The FSC problem is then reduced to the OSC problem. The lower bounds LB_1, LB_2 and LB_3

proposed in Section 2.3 for the OSC problem are valid lower bounds on the makespan for the FSC problem.

3.4.2 Binary variables relaxation-based lower bound [147]

This lower bound is derived by relaxing in the interval $[0,1]$ the binary variables D_{jk} and y_{ri}^{kj} ($i \neq r, r, i = \overline{1,m}, 1 \leq j < k \leq n.$) of the MILP model (P_1) that will be proposed for $Fm|ConfG = (V,E), prmu|C_{max}$ in Section 3.5.1. The resulting lower bound is denoted LB_4.

3.4.3 One bottleneck machine-based lower bound [147]

Still with one bottleneck machine, we improve a lower bound proposed for the permutation flow shop problem (see [95]). This latter bound is found by picking one machine, say M_u, to be the unique bottleneck. The preceding non-bottlenecks merge to form a single non-bottleneck denoted by N_{1u}= $\{M_1, M_2, \ldots, M_{u-1}\}$. On N_{1u}, each job J_j requires $p_{1u,j} = \sum_{i=1}^{u-1} p_{ij}$ time units. Similarly, the succeeding non-bottlenecks combine to become $N_{um} = \{M_{u+1}, M_{u+2}, \ldots, M_m\}$, on which the processing time of each J_j is $p_{um,j} = \sum_{i=u+1}^{m} p_{ij}$. The question is, how long will it take to perform all the jobs on the machines N_{1u}, M_u and N_{um}? Let $l_{1u} = \min_{j=1,\ldots,n}\{p_{1u,j}\}$ be the minimum amount of time before M_u starts to work and $l_{um} = \min_{j=1,\ldots,n}\{p_{um,j}\}$ be the minimum amount of time that it remains inactive after its work up to the end of the operations. Since all the jobs are processed in parallel on N_{1u}, each J_j is available for processing on M_u at time $p_{1u,j}$, which can be viewed as an ordinary ready time r_j. Scheduling the jobs on M_u in non-decreasing order of r_j minimizes the makespan on M_u (adapted Jackson's rule). Thus, the lower bound can be defined as follows, where $C_u(S)$ is the makespan on M_u using schedule S when jobs arrive at time r_j.

$$LB = \max_{u=1,\ldots,m}\{C_u(\nearrow r_j) + l_{um}\}, \text{where } r_j = p_{1u,j}$$

The role of N_{1u} and N_{um} can be interchanged and another similar bound can be found.

$$LB^{'} = \max_{u=1,\ldots,m}\{l_{1u} + C_u(\nearrow r_j)\}, \text{where } r_j = p_{um,j}$$

Tighter lower bound can be calculated for the permutation FSC problem by considering the conflict constraints between the jobs. Let $\overline{G} = (V, \overline{E})$ be the agreement graph and W_u be the vector representing the processing times of the jobs of V on N_{1u}. If we look at these processing times as weights in $\overline{G}$, we obtain the weighted graph $\overline{G}_u = (V, \overline{E}, W_u)$. Since the jobs of an independent set in $\overline{G}_u$ cannot be processed simultaneously on N_{1u}, the ready times of the jobs of V can be improved. Let IS be an independent set of maximum weight in $\overline{G}_u$. If we sort the jobs of IS in non-decreasing order of their weights in a list L, and we suppose without loss of generality that the obtained job ordering is: $J_1, \ldots, J_{|SI|}$, then new ready times $r_j^{'}$ can be defined as follows: $r_j^{'} = \begin{cases} \sum_{k=1}^{j} r_k, & \text{if } J_j \in IS \\ r_j, & \text{otherwise} \end{cases}$

Proposition 3.10 ([147]). *For the permutations FSC problem,*

$$LB_5^{'} = \max_{u=1,\ldots,m} \{C_u(\nearrow r_j^{'}) + l_{um}\}$$

is a valid lower bound for the makespan. The role of N_{1u} and N_{um} can be interchanged and another similar bound can be found, such that W_u will be the vector representing the processing times of the jobs of V on N_{um}.

$$LB_5^{''} = \max_{u=1,\ldots,m} \{l_{1u} + C_u(\nearrow r_j^{'})\}$$

We therefore have an overall lower bound of LB_5=max$\{LB_5^{'}, LB_5^{''}\}$. Note that the set *IS* is computed using the greedy algorithms GWMIN and GWMIN2 proposed in [136].

3.5 On the mathematical model

In this section, we will be discussing mixed-integer linear programming (MILP) models that summarize in mathematical form the essential components of different versions of our problem. We start first by the general case in which we study the FSC and the permutation FSC problems. Then, we restrict our attention to the two-machine FSC problem with unit-time operations.

3.5.1 Scheduling on m machines

The following model is proposed for $Fm|ConfG = (V, E), prmu|C_{max}$ and it is based on the Liao-You model ([99, 156], see "Exact methods" of the basic flow shop problem in Section 1.1.2 for a comparison between the MILP models proposed for $F2||C_{max}$). The parameters used are:

- a_{jk}: 1 if J_j and J_k are in conflict, 0 otherwise.
- M: big constant.

The decision variables are used to assign jobs to various positions in the sequence and to plan all the operations. They are defined as follows.

- D_{jk}: 1 if job J_j is scheduled any time before J_k, 0 otherwise.
- S_{ij}: start time of job J_j on machine M_i.
- C_{max}: the maximum completion time determined by the completion time of the job in the last position.

- y_{ri}^{kj}: when $a_{jk} = 1$, this variable is equal to 1 if the operation J_{rk} is scheduled any time before J_{ij}, 0 otherwise.
- q_{rjk}: surplus variable that accounts for the precedence relationship of jobs J_j and J_k on machine M_r.
- R_{ri}^{kj}: surplus variable that accounts for the precedence relationship of operations J_{rk} and J_{ij} when $a_{jk} = 1$.

The MILP model can be formulated as follows:

$$
(P_1)\quad
\begin{array}{lllll}
\min & C_{max} & & & \\
\text{S.C} & S_{ij} + p_{ij} \leq S_{i+1,j}; & i = \overline{1, m-1}, & j = \overline{1, n}, & (1)\\
& S_{ij} - S_{ik} + M.D_{jk} - p_{ik} = q_{ijk}; & i = \overline{1, m}, & 1 \leq j < k \leq n, & (2)\\
& a_{jk}(S_{rk} - S_{ij} - p_{ij} + M.y_{ri}^{kj}) = R_{ri}^{kj}.a_{jk}; & (r \neq i), r, i = \overline{1, m} & 1 \leq j < k \leq n, & (3)\\
& M - p_{ij} - p_{ik} \geq q_{ijk}; & i = \overline{1, m} & 1 \leq j < k \leq n, & (4)\\
& M - p_{ij} - p_{rk} \geq R_{ri}^{kj}; & (i \neq r), i, r = \overline{1, m} & 1 \leq j < k \leq n, & (5)\\
& C_{max} \geq S_{mj} + p_{mj}; & j = \overline{1, n}, & & (6)\\
& D_{jk} \in \{0, 1\}, y_{ri}^{kj} \in \{0, 1\}; & (i \neq r), r, i = \overline{1, m}, & 1 \leq j < k \leq n, & (7)\\
& q_{ijk} \geq 0; & i = \overline{1, m} & 1 \leq j < k \leq n, & (8)\\
& R_{ri}^{kj} \geq 0; & (r \neq i), r, i = \overline{1, m}, & 1 \leq j < k \leq n, & (9)\\
& S_{ij} \geq 0; & i = \overline{1, m}, & j = \overline{1, n}. & (10)
\end{array}
$$

The objective is to minimize the makespan C_{max} i.e. to minimize the completion time of the last job in the sequence on the last position. Constraint (1) specifies that job J_j cannot begin processing on machine M_{i+1} until it has completed its processing on machine M_i. Constraint (2) ensures that job J_k either precedes job J_j in the sequence (i.e., $S_{ij} - S_{ik} - p_{ik} \geq 0, \forall\, i = 1 \ldots, m$), or follows it (i.e., $S_{ik} - S_{ij} - p_{ij} \geq 0, \forall\, i = 1 \ldots, m$), but not both. Liao-You [99] algebraically combined these pairs of inequality dichotomous constraints into a single equality constraint that they set equal to a surplus variable q_{ijk}, with M defined as a very large constant. Constraint (3) enforces the conflict constraints between jobs J_k and J_j on M_r and M_i respectively ($r \neq i$ and $1 \leq j < k \leq n$). When $a_{jk} = 1$, operation J_{rk} either precedes operation J_{ij} (i.e., $S_{rk} \geqslant S_{ij} + p_{ij}$), or follows it (i.e., $(S_{ij} \geqslant S_{rk} + p_{rk})$, but not both. We model these two disjoint constraints using the binary variable y_{ri}^{kj}. Then, we combine the resulting inequalities into a single equality constraint that we set equal to a surplus variable R_{rk}^{ij}. Again, M is a very large constant. Constraints (4) and (5) ensure the feasibility of constraints (2) and (3) respectively. Constraint (6) equates the makespan to the maximum completion time of all the jobs on the last machine. Constraint (7) sets up the binary restrictions for D_{jk} and y_{ri}^{kj}. Constraints (8), (9) and (10) define the surplus variables and the start time variables as non-negative continuous variables.

This model requires $nm(m(n-1)+1)$ constraints, $\frac{n(n-1)}{2}(m(m-1)+1)$ binary variables and $(\frac{nm^2(n-1)}{2} + nm + 1)$ continuous variables.

However, one can find examples in which the optimal schedule does require sequence changes between machines (see e.g. Figure 3.2.6). To model the situation when sequence changes are allowed, we associate to each machine a set of variables that defines the sequence of jobs on this machine. If we consider the previous model, it is sufficient to replace the variable D_{jk} by the variable D_{ijk} where:

- D_{ijk}: 1 if job J_j is scheduled any time before J_k on machine M_i, 0 otherwise.

Therefore, a MILP model for the problem $Fm|ConfG = (V, E)|C_{max}$ (denoted by (P_2)) is obtained by changing only constraint (2) in (P_1) as follows:

$$S_{ij} - S_{ik} + M.D_{ijk} - p_{ik} = q_{ijk} \quad \forall i = \overline{1, m}, \quad \forall 1 \leq j < k \leq n.$$

This model requires the same number of constraints and continuous variables than (P_1), but at a cost of an additional $\frac{n(n-1)}{2}(m-1)$ binary variables.

3.5.2 Scheduling on two machines with unit-time operations

We present in this section three MILP models for $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$. These models are based on the fact that, there exists an optimal no-wait permutation schedule for this restricted problem. The parameters used are:

- a_{jk}: 1 if J_j and J_k are in conflict, 0 otherwise.
- $U = min\{H_{11}, H_{12}\} - 1$, where $min\{H_{11}, H_{12}\}$ is an upper bound on the makespan obtained by the heuristics H_{11} and H_{12} that will be discussed in Section 3.6.4.
- M: big constant.

3.5.2.1 First MILP model (P_3) [152]

The present MILP model relies on an assignment of jobs to various positions in the schedule. The efficiency of this model depends on the number of positions considered. Recall that we have at most $2n-1$ possible positions, this number can be reduced by computing an upper bound on the makespan. This upper bound is obtained by applying the heuristics H_{11} and H_{12} proposed for $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ in Section 3.6.4. The objective is to minimize the position of the last job in the sequence. The decision variables used are:

- x_{jr}: 1 if job J_j is assigned to position r, 0 otherwise ($j = \overline{1, n}; r = \overline{1, U}$).

- y: the maximum completion time determined by the completion time of the job in the last position.

The proposed mathematical model is defined as follows:

$$
(P_3)\left|\begin{array}{lll}
\min & y & \\
\text{S.C} & \sum_{j=1}^{n} x_{jr} \leqslant 1; & r = \overline{1, U}, \quad (1)\\
& \sum_{r=1}^{U} x_{jr} = 1; & j = \overline{1, n}, \quad (2)\\
& x_{jr} + x_{k,r+1} \leqslant 2 - a_{jk}; & j, k = \overline{1, n}, (j \neq k), r = \overline{1, U-1}, \quad (3)\\
& \sum_{j=1}^{n} x_{jr} * r + 1 \leqslant y; & r = \overline{1, U}, \quad (4)\\
& x_{jr} \in \{0, 1\}, y \geqslant 0; & j = \overline{1, n}, r = \overline{1, U}. \quad (5)
\end{array}\right.
$$

Constraint (1) insures that each position is filled with at most one job, while constraint (2) insures that each job is assigned to just one position. Constraint (3) specifies that two conflicting jobs cannot be processed in two successive positions. Constraint (4) equates makespan to the maximum completion time of all the jobs. Constraint (5) sets up the binary restriction for x_{jr} and defines y as a nonnegative continuous variable that takes integer value since processing times are given integer values.

3.5.2.2 Second MILP model (P_4) [154]

The idea of this model is to minimise the total number of idle times generated. To do so, we define the following decision variables:

- x_{jk}: 1 if job J_k is scheduled immediately after J_j, 0 otherwise.
- z_k: the position of job J_k in the optimal sequence.

The developed mathematical model is as follows:

$$
(P_4)\left|\begin{array}{lll}
\min & \sum_{j=1}^{n}\sum_{k=1}^{n} x_{jk} a_{jk} & \quad (1)\\
\text{S.C} & \sum_{j=1}^{n}\sum_{k=1}^{n} x_{jk} = n - 1, & \quad (2)\\
& \sum_{j=1}^{n} x_{jk} \leqslant 1; \quad k = \overline{1, n}, & \quad (3)\\
& \sum_{k=1}^{n} x_{jk} \leqslant 1; \quad j = \overline{1, n}, & \quad (4)\\
& z_j - z_k \leqslant n(1 - x_{jk}) - 1; \quad (j \neq k), j, k = \overline{1, n}, & \quad (5)\\
& x_{jj} = 0; \quad j = \overline{1, n}, & \quad (6)\\
& x_{jk} \in \{0, 1\}, 1 \leqslant z_k \leqslant n; \quad j, k = \overline{1, n}. & \quad (7)
\end{array}\right.
$$

The objective function is given by (1): minimizing the total number of idle times. Constraint (2) assures that the sum of precedence relationships of the n jobs is $(n-1)$. Constraint (3) ensures that job J_k has at most one immediate predecessor. Constraint (4) ensures that job J_j has at most one immediate successor. Constraint (5) specifies that each job has to be assigned to exactly one position. Constraint (6) guarantees that there is no precedence relationship between a job and itself. Constraint (7) sets up the binary restriction for x_{jk} and defines z_k as a nonnegative continuous variable bounded by 1 and n, this variable takes an integer value since we have n positions from 1 to n and the difference between two successive positions is equal to one (which is ensured by constraint (5)).

3.5.2.3 Third MILP model (P_5) [154]

This model is obtained by adapting (P_1) to the case of two machines with unit-time operations, then the decision variables used are:

- D_{jk}: 1 if job J_j is scheduled any time before J_k, 0 otherwise.
- S_j: start time of job J_j.
- q_{jk}: surplus variable that accounts for the precedence relationship of jobs J_j and J_k.
- y: the maximum completion time determined by the completion time of the job in the last position.

The mathematical model is formulated as follows:

$$
(P_5)\quad
\begin{array}{lll}
\min & y & \\
\text{S.C} & S_j - S_k + M.D_{jk} - 1 - a_{jk} = q_{jk}; & 1 \le j < k \le n, \quad (1)\\
& M - 2 - 2a_{jk} \ge q_{jk}; & 1 \le j < k \le n, \quad (2)\\
& y \ge S_j + 2; & j = \overline{1,n}, \quad (3)\\
& D_{jk} \in \{0,1\}, q_{jk} \ge 0; & 1 \le j < k \le n, \quad (4)\\
& S_j \ge 0; & j = \overline{1,n}. \quad (5)
\end{array}
$$

Constraint (1) enforces the conflict constraints and ensures that job J_j either precedes job J_k (i.e. $S_k - S_j \ge 1 + a_{jk}$) or follows it (i.e. $S_j - S_k \ge 1 + a_{jk}$), but not both. We model these two disjoint constraints using the binary variable D_{jk}. Then, we combine the resulting inequalities into a single equality constraint that we set equal to a surplus variable q_{jk}, with M defined as a very large constant. Constraint (3) ensures the feasibility of constraint (1). Constraint (3) equates makespan to the maximum completion time of all the jobs.

The size complexity of each of the three MILP models is presented in Table 3.5.1.

Table 3.5.1: Size complexity of the MILP models of $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$

Model	Binary variables	Constraints	Continuous variables
(P_3)	$n(2n-1)$	$(n^2-n)(2n-2)+(5n-2)$	1
(P_4)	n^2	$n(n+3)+1$	n
(P_5)	$\frac{1}{2}n(n-1)$	n^2	$\frac{1}{2}n(n+1)$

3.6 Branch and bound algorithm

We describe in this section a branch and bound algorithm that has been proposed in [152] for $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$. This algorithm constructs a permutation π of n jobs that minimizes the total number of idle times. It is characterized in terms of the following procedures.

3.6.1 Branching rule

With each node, at level k of the search tree, is associated a partial permutation σ in which the jobs in the first k positions have been already fixed. The number of branches departing from this node equals to the number of jobs not in σ, i.e. for each job J_j with $J_j \notin \sigma$ a branch is considered extending σ by one additional position to a new partial permutation (σJ_j).

3.6.2 Selecting rule

The algorithm pursues a depth first strategy. That is, movement down a branch of the search tree, and therefore having more jobs fixed in σ. The jobs are selected according to a branching priority. In this work, we test four heuristics for setting the branching priority.

- Nonincreasing order of the conflict degrees (Decrease_conflict).
- Nondecreasing order of the conflict degrees (Increase_conflict).
- Order of the schedule obtained by the best heuristic when evaluating the root node (Best_heuristic).
- No branching strategy: select the jobs as they were inserted by the user (No_sort).

The conflict degree of a job is the degree of the corresponding vertex in the conflict graph.

3.6.3 Lower bounds

We present in this section lower bounds that apply in order to estimate the quality of all possible completions τ of partial permutations σ to a complete permutation $(\sigma\tau)$. A compromise is required between the tightness of the lower bounds and the running time to compute them. In what follows, $C(\sigma)$ denotes the completion time of the last job in σ.

3.6.3.1 Agreement graph-based lower bound (lb_1, lb_2)

These lower bounds are obtained by adapting LB_1 and LB_2 to the case of two machines with unit-time operations (the weights are equal to one) and by adding some improvements. Let $V_\tau \subset V$ be a subset of vertices corresponding to the jobs of τ, $\overline{G}[V_\tau]$ be the agreement graph induced by the vertices of V_τ and IS_τ be a maximum independent set in $\overline{G}[V_\tau]$.

Theorem 3.11.

$$lb = C(\sigma) + lb(IS_\tau) + (u - 1)$$

where

$$lb(IS_\tau) = \begin{cases} 2|IS_\tau| & \text{if } |\tau| < 2|IS_\tau| \\ |\tau| + 1 & \text{if } |\tau| \geq 2|IS_\tau| \end{cases} \quad \text{and } u = \begin{cases} 1 & \text{if the last job in } \sigma \text{ is in conflict with} \\ & \text{all the jobs of } \tau, \text{ or if } \sigma \text{ is empty.} \\ 0 & \text{otherwise,} \end{cases}$$

is a lower bound on the makespan of all schedules that begin with σ.

Proof. First consider the term $lb(IS_\tau)$ and the jobs of τ. Since non adjacent jobs in $\overline{G}[V_\tau]$ must be scheduled in disjoint time intervals for any feasible schedule of τ, then $2|IS_\tau|$ represents a lower bound on the makespan of the jobs of τ. This lower bound can be improved by considering the remaining jobs of τ. Note that there exist $|IS_\tau| - 1$ interior idle times in the schedule of the jobs of IS_τ. In the most favorable situation, the jobs of $\tau \setminus IS_\tau$ are placed in these idles times without inducing any additional interior idle times. If $|\tau \setminus IS_\tau| > |IS_\tau| - 1$ (i.e., $|\tau| \geq 2|IS_\tau|$), then $|\tau \setminus IS_\tau| - (|IS_\tau| - 1)$ jobs have to be scheduled after the last job of IS_τ. Thus, we have at least $|\tau \setminus IS_\tau| - (|IS_\tau| - 1) = |\tau| - |IS_\tau| - |IS_\tau| + 1 = |\tau| - 2|IS_\tau| + 1$ additional time units. The lower bound in this case is $lb(IS_\tau) = 2|IS_\tau| + (|\tau| - 2|IS_\tau| + 1) = |\tau| + 1$.

Next observe that an idle time occurs when the last job in σ conflicts with all the jobs of τ or when σ is empty (an idle time on M_2 in the beginning of the schedule) in which case $u = 1$, otherwise $u = 0$. Therefore, the result is established. □

The two greedy algorithms, GMIN (see Algorithm 2) and GMAX, presented in [136] have been considered to compute IS_τ. The resulting lower bounds are denoted lb_1 and lb_2 respectively. These lower bounds are expected to perform well when the conflict graph is dense.

3.6.3.2 Conflict constraints relaxation-based lower bound (lb_3)

This lower bound can be derived by first relaxing the constraints of conflict between the jobs of τ (similarly to LB_3). The problem is then reduced to the basic two-machine flow shop problem with unit-time operations. In this case:

$$lb_3^{'} = C(\sigma) + |\tau| + u.$$

is a valid lower bound for the makespan. Observe that an idle time occurs when the last job in σ conflicts with all the jobs of τ or when σ is empty (an idle time on M_2 in the beginning of the schedule) in which case $u = 1$, otherwise $u = 0$.

Let $S = \{J_j \in \tau : d(J_j)_{G[V_\tau]} = |\tau| - 1\}$ and $L = \{J_j \in \tau : d(J_j)_{G[V_\tau]} = |\tau| - 2\}$, where $d(J_j)_{G[V_\tau]}$ is the degree of J_j in the conflict graph induced by the vertices of V_τ.

Theorem 3.12.

$$lb_3 = lb_3^{'} + s + l.$$

where

$$s = \begin{cases} |S| & \text{if } |S| < |\tau|. \\ |S| - 1 & \text{if } |S| = |\tau|. \end{cases} \quad \text{and } l = \begin{cases} \lfloor |L|/2 \rfloor - 1 & \text{if } |L| \text{ is even.} \\ \lfloor |L|/2 \rfloor & \text{if } |L| \text{ is odd.} \end{cases}$$

is a lower bound on the makespan of all schedules that begin with σ.

Proof. Let us consider the term s and the jobs of τ. If $|S| = |\tau|$ ($G[V_\tau]$ is a clique), the number of idle times obtained by scheduling the jobs of τ, in the most favorable situation, is $|S| - 1$. Now, suppose that $\tau \backslash S \neq \emptyset$, then there exists at least one job of $\tau \backslash S$ that will be scheduled near a job of S, which will induce at least one idle time (since the jobs of S are in conflict with all the jobs of τ). In addition, each of the remaining jobs of S will induce at least one idle time for all the possible positions. Hence the total number of idle times is $|S|$.

We now examine the term l. Observe that the jobs of L corresponds to leaves (vertices of degree 1) in $\overline{G}[V_\tau]$. In the most favorable situation, we need one path to cover two leaves. Thus, we need at least $(\lfloor |L|/2 \rfloor + |L| \bmod 2)$ vertex-disjoint paths to cover all the leaves of $\overline{G}[V_\tau]$ which will induce $\lfloor |L|/2 \rfloor +$ $(|L| \bmod 2)$-1 interior idle times, and the result follows. □

3.6.3.3 Maximum matching-based lower bound (lb_4)

This lower bound is obtained by relaxing the precedence constraints between the operations of each job of τ. The relaxed sub-problem, of scheduling the jobs of τ, is then reduced to the two-machine open shop problem with unit-time operations and conflict graph and might be viewed as a maximum matching problem in a bipartite graph (see Theorem 2.11). Following the proof of Theorem 2.11, we

construct the bipartite graph R_τ from the conflict graph $G[V_\tau]$ induced by the vertices of τ (see Figure 2.2.12). Let M be a maximum matching in R_τ. Observe that an idle time occurs when the last job in σ conflicts with all the jobs of τ or when σ is empty (an idle time on M_2 in the beginning of the schedule) in which case $u = 1$, otherwise $u = 0$. Hence $lb_4 = C(\sigma) + 2|\tau| - |M| + (u - 1)$ is a lower bound on the makespan.

3.6.3.4 Binary variables relaxation-based lower bound (lb_5, lb_6, lb_7)

Let us consider the sub-problem of scheduling the jobs of τ. Three lower bounds can be derived by relaxing the binary variables of the MILP models (P_3), (P_4) and (P_5) in the interval $[0, 1]$. The resulting lower bounds are denoted by lb_5', lb_6' and lb_7' respectively. Similarly to the previous lower bounds, an idle time occurs when the last job in σ conflicts with all the jobs of τ or when σ is empty (an idle time on M_2 in the beginning of the schedule) in which case $u = 1$, otherwise $u = 0$. Therefore

$$lb_i = C(\sigma) + lb_i' + (u - 1), i = \overline{5, 7}.$$

are valid lower bounds on the makespan of the original scheduling problem.

3.6.4 Upper bounds (heuristics)

We develop in this section four heuristics in an effort to find tight upper bounds for the makespan. Each heuristic initially builds a priority list, then a strategy is applied to schedule the constructed list.

Step 1. The priority list is constructed any of the following rules.

1. Nonincreasing order of the conflict degrees.
2. Nondecreasing order of the conflict degrees.

Step 2. The priority list is then scheduled with either of the following strategies.

1. Place the next job of the priority list on the first available position that generates a minimum total idle time.
2. Schedule the job with the highest priority value (from the jobs that have not been assigned yet) which is not in conflict with the job already processed. If such a job does not exist, then select the first job of the priority list.

The second strategy of Step 2 is inspired from the following Lemma.

Lemma 3.13 ([57]). *A path cover has the minimum number of paths if, and only if, it has the maximum number of edges.*

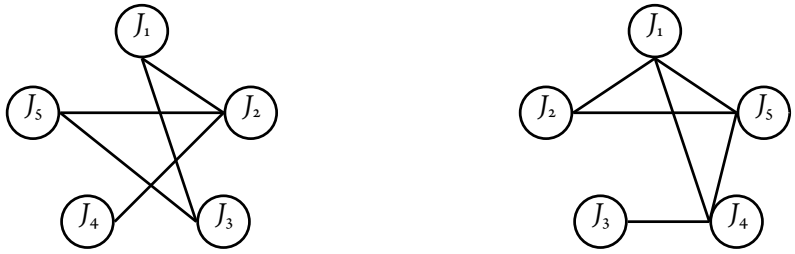


Figure 3.6.1: Conflict graph G. (a) Example 3.1. (b) Example 3.2.

We denote by H_{ij} the corresponding heuristic in which priority rule i of Step 1 is applied to strategy j of Step 2.

Let ω be a partial permutation computed by applying one of the heuristics described above on the sub-problem induced by the jobs of τ and let $\omega^{'}$ be a partial permutation obtained by reversing the jobs of ω. It is clear that both of the partial permutations have the same makespan. Observe that if the last job in σ conflicts with the first job of ω and it is not in conflict with the last job of this permutation, a tighter upper bound can be obtained by concatenating the partial permutations σ and $\omega^{'}$ to a complete permutation $(\sigma\omega^{'})$.

Example 3.1. *Let us consider the following instance with $n = 5$. The jobs are subject to the conflict constraints depicted in Figure 3.6.1.(a). We first apply priority rules of Step 1 on this instance. We derive the following lists: $\{J_2, J_1, J_3, J_5, J_4\}$ and $\{J_4, J_1, J_3, J_5, J_2\}$ corresponding respectively to priority rules 1 and 2. Then, we apply the two strategies of Step 2, we get the following schedules: H_{11}: $(J_2, J_3, J_4, J_5, J_1)$ with makespan=6, H_{12}: $(J_2, J_3, J_4, J_1, J_5)$ with makespan=6, H_{21}: $(J_4, J_1, J_5, J_2, J_3)$ with makespan= 7 and H_{22}: $(J_4, J_1, J_5, J_3, J_2)$ with makespan=7.*

Let us now examine the time complexity of the above heuristics. Step 1 can be executed in $O(nlogn)$-time. However, the two strategies of Step 2 run in $O(n^2)$-time. Thus, the overall time complexity of each heuristic is $O(n^2)$.

3.6.5 Algorithm scheme

In the branch and bound algorithm we implemented, we keep track of the best permutation schedule that we found so far; this is called the incumbent. Initially, it is the best permutation schedule obtained when evaluating the root of the search tree. We only invoke H_{11} and H_{12} when computing the upper bound of a node of the branch and bound tree. This choice is justified by the good performance of these heuristics that we will discuss in Section 3.8.1.

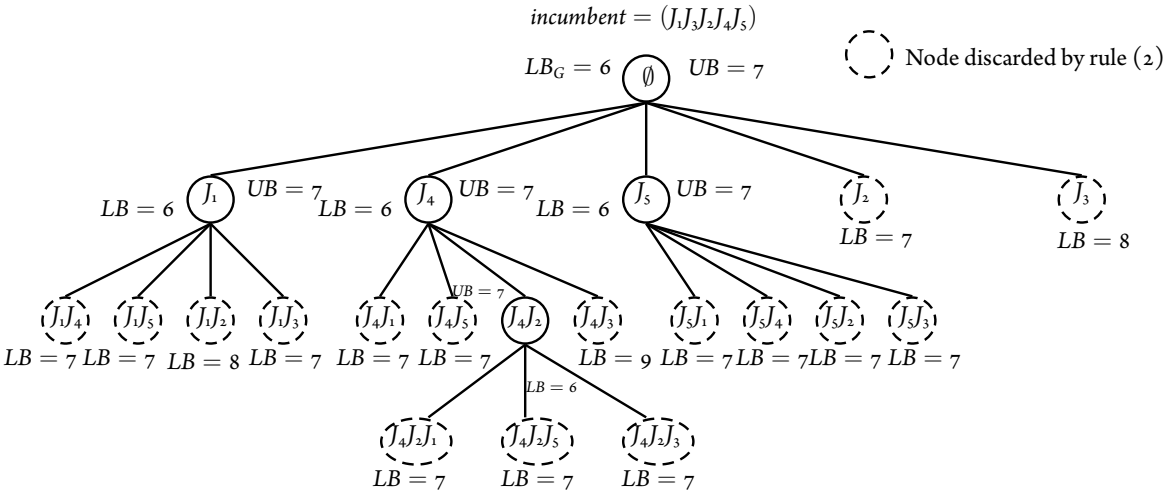


Figure 3.6.2: Example 3.2: search tree of the branch and bound algorithm.

The lower bound of the root node is called the general lower bound (LB_G). A node of the search tree is discarded in four cases:

1. This node has scheduled all the n jobs.
2. The lower bound of this node is greater than or equal to the makespan of the incumbent.
3. The lower bound of this node is equal to its upper bound.
4. the upper bound of this node is equal to the general lower bound.

The branching is ended until all nodes have been discarded or a solution with a makespan equals to the general lower bound is found. It should be noticed that, when the discarding rules (1) and (2) are satisfied at a node of the search tree, it is not necessarily to compute the upper bound of this node.

Example 3.2. *We consider the following instance with $n = 5$. Figure 3.6.1.(b) presents the conflict constraints between the jobs. The lower bound used is $lb_3' + s$ (Theorem 3.12). The branching priority is $\{J_1, J_4, J_5, J_2, J_3\}$, it is set according to the nonincreasing order of the conflict degrees. Following the iterations along the branch and bound tree of Figure 3.6.2, the optimal permutation schedule is $(J_1J_3J_2J_4J_5)$, with a makespan of 7 units.*

3.7 Heuristic approaches

We present in this section two different heuristic approaches to solve $Fm|ConfG = (V, E), prmu|C_{max}$ [147]. These approaches are based on an algorithm that plans all the operations of a given sequence of jobs on the m machines so as to minimize the maximum completion time. We first present this algorithm.

3.7.1 Makespan computation

The algorithm we are presenting is mainly based on the following property.

Property 3.1. *A machine M_u is allowed to remain idle if, and only if,*

- *All the jobs have completed their processing on this machine, and there exists at least one machine still working.*
- *There exists a machine executing a job that is in conflict with the one to be processed on M_u.*
- *There exists at least a machine M_v, with $v < u$, that did not finish the processing of the operation of the job to be executed on M_u.*

The construction of the schedule is maintained following the time axis and using a parameter t. Let O_i be the set of operations to be performed on machine M_i and R_t be the set of operations ready to be processed at time t on the m machines. A general scheme of this algorithm is given in Algorithm 7 that can be implemented in $O(nm^3)$.

Algorithm 7 Makespan computation for $Fm|ConfG = (V, E), prmu|C_{max}$

Input: Job sequence: $J_1, J_2, \ldots, J_n$

1: $M = \{M_1, \ldots, M_m\}$, $t = 0$
2: **while** $M \neq \emptyset$ **do**
3: $R_t = \emptyset$
4: **for** each $M_i \in M$ available at time t **do**
5: Find the next unscheduled operation according to input job sequence order, let J_{ij} be this operation
6: **if** J_{ij} is not in conflict with an operation that is running in one of the machines at time t **then**
7: $R_t = R_t \cup \{J_{ij}\}$
8: **end if**
9: **end for**
10: Sort the operations of R_t according to an order of selection OS_k, $k = 1, \ldots, 8$ (described below)
11: Schedule a subset of R_t (eventually the full set R_t) starting from time t subject to the conflict and precedence constraints and following the order of selection OS_k
12: **for** $i = 1, \ldots, m$ **do**
13: Delete from O_i the scheduled operation (if any)
14: **if** $O_i = \emptyset$ **then**
15: $M = M \setminus \{M_i\}$
16: **end if**
17: **end for**
18: Update t to the least finishing time of the scheduled operations
19: **end while**

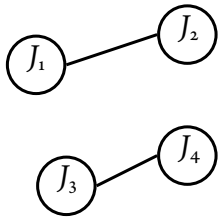
Output: Makespan C_{max} and the planning of all the operations

We denote by c_j (resp. a_j) the degree of job J_j in the conflict graph G (resp. agreement graph $\overline{G}$). Eight orders of selection have been used to sort the operations of R_t: (OS_1) increasing order of c_j/p_{ij}, (OS_2) decreasing order of c_j/p_{ij}, (OS_3) increasing order of a_j/p_{ij}, (OS_4) decreasing order of a_j/p_{ij}, (OS_5) decreasing order of p_{ij}, (OS_6) increasing order of p_{ij}, (OS_7) decreasing order of c_j and (OS_8) increasing order of c_j.

Example 3.3. *To demonstrate how Algorithm 7 works, consider 4 jobs to be processed on 3 machines, whose processing times are given in Figure 3.7.1.(a). The jobs are subject to the conflict constraints depicted in Figure 3.7.1.(b).*

	J_1	J_2	J_3	J_4
M_1	6	0	3	4
M_2	5	1	5	4
M_3	4	4	4	0

(a)



(b)

Figure 3.7.1: Example 3.3: (a) processing times. (b) conflict graph $G = (V, E)$.

The job sequence is given by: $J_2\ J_3\ J_1\ J_4$. *Initially* $t = 0$, $M = \{M_1, M_2, M_3\}$, $O_1 = \{J_{13}, J_{11}, J_{14}\}$, $O_2 = \{J_{22}, J_{23}, J_{21}, J_{24}\}$ *et* $O_3 = \{J_{32}, J_{33}, J_{31},\}$. *The order of selection used is* OS_7.

1. $M \neq \emptyset$. *We start by* M_1. $p_{12} = 0$ *but* $p_{13} = 3$, *hence* J_{13} *is ready for processing at* $t = 0$ *and* $R_0 = \{J_{13}\}$.
2. *Regarding* M_2, J_{22} *is ready for processing at* $t = 0$, $R_0 = R_0 \cup \{J_{22}\}$.
3. *For* M_3, J_{32} *is not ready for processing at* $t = 0$ *as* J_{22} *has not been processed yet. Thus,* M_3 *remains inactive.* $OS_7(J_{13}) = OS_7(J_{22}) = 1$, *since we have a tie,* J_{13} *is selected first.* J_{13} *and* J_{22} *are not in conflict, so they will be processed starting from* $t = 0$. $O_1 = O_1 \setminus \{J_{13}\}$ *and* $O_2 = O_2 \setminus \{J_{22}\}$.
4. *We update t to the least finishing time of all the scheduled operations which is equal to* 1. *Since* $M \neq \emptyset$, *we start by* M_1. *At* $t = 1$, M_1 *is not available.*
5. *We pass to* M_2, J_{23} *is not ready for execution at* $t = 1$ *since* J_{13} *is still processing.*
6. *For* M_3, *the operation to be processed at* $t = 1$ *is* J_{32}. *In addition,* $\nexists$ *any operation that is running at time t and conflicts with* J_{32} (J_{13} *and* J_{32} *are not in conflict). Thus,* $R_1 = \{J_{32}\}$. *We process* J_{32} *starting from* $t = 1$. $O_3 = O_3 \setminus \{J_{32}\}$.
7. *We set t to 3. Since* $M \neq \emptyset$, *we start by* M_1. *At* $t = 3$, *the operation to be processed is* J_{11}, *but* J_{11} *and* J_{32} *are in conflict. Thus,* M_1 *remains inactive.*
8. *For* M_2, J_{23} *is ready for processing at* $t = 3$, *hence* $R_3 = \{J_{23}\}$.

9. *M_3 is not available. We process J_{23} starting from time t=3. $O_2 = O_2 \setminus \{J_{23}\}$.*

10. *t is set to 5. Analogously, iterate this process until $M = \emptyset$. The resulting schedule has $C_{max} = 20$.*

The resulting schedule with the planning of all the operations is depicted in Figure 3.7.2.

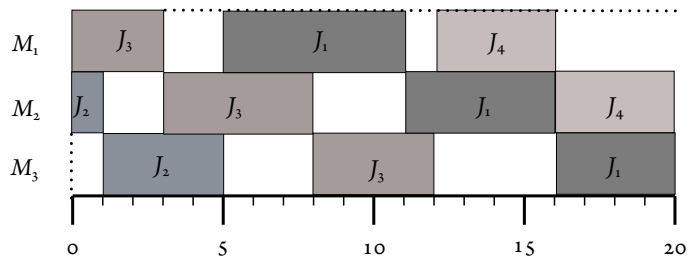


Figure 3.7.2: Example 3.3: schedule of the sequence $J_2\ J_3\ J_1\ J_4$ given by Algorithm 7.

3.7.2 First heuristic approach: H1mFSCn

This heuristic approach starts by scheduling the jobs of an independent set S of maximum cardinality in G, in which the problem is reduced to the basic m-machine flow shop problem. The set S is computed using the greedy algorithm GMIN (see Algorithm 2) and the jobs of S are scheduled using the NEH-T algorithm (the reader is referred to Section 1.1.2 paragraph "Approximation methods" of the flow shop problem for more details about this algorithm). Then, it selects a job according to priority order PO_k (described below), from the jobs that haven't been assigned yet, and places it at all possible positions of the partial sequence under construction. The makespans of the resulting partial sequences are computed by Algorithm 7, and the partial sequence with the best makespan will fix the relative positions of the assigned jobs for the remaining steps. This process is repeated until all jobs are fixed and a complete sequence is found alongside with the planning of all operations. A general scheme of this approach is given in Algorithm 8. Since the NEH-T algorithm runs in $O(n^2m)$-time, the time complexity of H1mFSCn is $O(n^2m^3)$-time.

Algorithm 8 H1mFSCn: first heuristic approach for the m-machine FSC problem.

Input: Data of the FSC problem.

1: Find an independent set S of maximum cardinality in G.
2: Schedule the jobs of S.
3: Arrange the jobs of $V - S$ in a list L according to a priority order PO_k, $k = 1, \ldots, 8$.
4: **for** $l = 1, \ldots, |L|$ **do**
5: Insert the l^{th} job of the generated list L into each of the $(l + |S|)$ possible positions in the partial sequence found in the previous step and keep the partial sequence defining the best makespan.
6: **end for**

Output: The final sequence and the corresponding C_{max}.

The solution produced by H1mFSCn (Algorithm 8) depends on the insertion order of the jobs of $V-S$. In what follows, we consider eight priority orders to generate the list L: (PO_1) increasing order of c_j/p_j, (PO_2) decreasing order of c_j/p_j, (PO_3) increasing order of a_j/p_j, (PO_4) decreasing order of a_j/p_j, (PO_5) decreasing order of p_j, (PO_6) increasing order of p_j, (PO_7) decreasing order of c_j and (PO_8) increasing order of c_j.

Example 3.4. *To illustrate the algorithm outlined above, we consider the 4-job, 3-machine instance of Example 3.3. Let us start by applying the greedy algorithm GMIN (Algorithm 2). The resulting independent set is* $S = \{J_1, J_3\}$. *To schedule the jobs of S, we use the NEH-T algorithm. The best sequence is:* J_3, J_1. *Then, we sort the remaining jobs in increasing order of* c_j/p_j, $j \in \{2, 4\}$. $PO_1(J_2) = \frac{1}{5}$, $PO_1(J_4) = \frac{1}{8}$. J_4 *is selected first and the three partial sequences are tested to choose the best one. The computation of the makespan of the generated sequences is done using Algorithm 7 with* OS_7 *as priority order (calculations are similar to Example 3.3).*

1. J_4 *in the* 1^{st} *position. Sequence:* $J_4, J_3, J_1 \Longrightarrow$ *corresponding* $C_{max} = 25$.
2. J_4 *in the* 2^{nd} *position. Sequence:* $J_3, J_4, J_1 \Longrightarrow$ *corresponding* $C_{max} = 25$.
3. J_4 *in the* 3^{rd} *position. Sequence:* $J_3, J_1, J_4 \Longrightarrow$ *corresponding* $C_{max} = 20$.

Selected sequence: J_3, J_1, J_4.

1. J_2 *in the* 1^{st} *position. Sequence:* $J_2, J_3, J_1, J_4 \Longrightarrow$ *corresponding* $C_{max} = 20$.
2. J_2 *in the* 2^{nd} *position. Sequence:* $J_3, J_2, J_1, J_4 \Longrightarrow$ *corresponding* $C_{max} = 23$.
3. J_2 *in the* 3^{rd} *position. Sequence:* $J_3, J_1, J_2, J_4 \Longrightarrow$ *corresponding* $C_{max} = 23$.
4. J_2 *in the* 4^{th} *position. Sequence:* $J_3, J_1, J_4, J_2 \Longrightarrow$ *corresponding* $C_{max} = 25$.

The best sequence is: J_2, J_3, J_1, J_4 *with* $C_{max} = 20$. *This solution is optimal since the best lower bound is equal to 20. The schedule of this sequence is depicted in Figure 3.7.2.*

3.7.3 Second heuristic approach: H2mFSCn

In this heuristic approach, the conflict graph G is partitioned into a minimum number of independent sets using iteratively the greedy algorithm GMIN (see Algorithm 2) until no vertex remains. The jobs of each set are scheduled using the NEH-T algorithm, we urge the reader to review Section 1.1.2 paragraph "Approximation methods" of the flow shop problem for more details about this algorithm. Then, a set, from the sets that haven't been assigned yet, is selected according to priority order PO'_k (described below) and placed at all possible positions of the partial sequence under construction. The makespans

of the resulting partial sequences are computed by Algorithm 7 and the partial sequence with the best makespan will fix the relative positions of the assigned jobs for the remaining steps. This process is repeated until all sets are fixed and a complete sequence is found alongside with the planning of all operations. The basic algorithm is summarized in Algorithm 9. All the steps of this approach can be executed in $O(n^2m^3)$-time.

Algorithm 9 H2mFSCn: second heuristic approach for the m-machine FSC problem

Input: Data of the FSC problem.

1: Partition the conflict graph G into a minimum number of independent sets.
2: Schedule the jobs of each independent set S_r.
3: Arrange the independent sets in a list L according to a priority order PO'_k, $k = 1, \ldots, 8$.
4: **for** $l = 1, \ldots, |L|$ **do**
5: Insert the l^{th} independent set of the generated list L into each of the $\sum_{r=1}^{l-1} |S_r| + 1$ possible positions in the partial sequence found in the previous step and keep the one defining the best makespan.
6: **end for**

Output: The final sequence and the corresponding C_{max}.

Each independent set S_r is characterised by:

- L_r: makespan computed by using the NEH-T algorithm.
- $C_r = \sum_{J_j \in S_r} c_j$: conflict degree.
- $A_r = \sum_{J_j \in S_r} a_j$: agreement degree.

From these characteristics, eight priority orders have been established: (PO'_1) increasing order of C_r/L_r, (PO'_2) decreasing order of C_j/L_r, (PO'_3) increasing order of A_j/L_r, (PO'_4) decreasing order of A_j/L_r, (PO'_5) decreasing order of L_r, (PO'_6) increasing order of L_r, (PO'_7) decreasing order of C_r and (PO'_8) increasing order of C_r.

Example 3.5. *Consider again the 4-job, 3-machine instance of Example 3.3. First, the conflict graph G is decomposed into a minimum number of independent sets. By applying the greedy algorithm GMIN (Algorithm 2) twice, two independent sets are generated:* $S_1 = \{J_1, J_3\}$ *and* $S_2 = \{J_2, J_4\}$. *We need now to schedule the jobs of each independent set. Using the NEH-T algorithm, the obtained sequence for* S_1 *is:* J_3, J_1 *with* $C_{max} = 18$, *whereas, the best sequence for* S_2 *is:* J_2, J_4 *with* $C_{max} = 8$. *Then, we sort the independent sets in increasing order of* C_j/l_j, $j \in \{1, 2\}$. $PO'_1(S_1) = \frac{2}{18} = 0.111$, $PO'_1(S_2) = \frac{2}{8} = 0.25$. S_1 *is selected first.*

S_2 *is inserted into each of the three possible positions. The computation of the makespan of each of the generated sequences is done using Algorithm 7 with* OS_7 *as priority order (calculations are similar to Example 3.3).*

1. S_2 *in the* 1^{st} *position. Sequence:* $J_2, J_4, J_3, J_1 \Longrightarrow$ *corresponding* $C_{max} = 25$.

2. S_2 *in the* 2^{nd} *position. Sequence:* $J_3, J_2, J_4, J_1 \Longrightarrow$ *corresponding* $C_{max} = 29$.

3. S_2 *in the* 3^{rd} *position. Sequence:* $J_3, J_1, J_2, J_4 \Longrightarrow$ *corresponding* $C_{max} = 23$.

The best sequence is: J_3, J_1, J_2, J_4 *with* $C_{max} = 23$.

3.8 Computational experiments

The runs of the above MILP models, branch and bound algorithm, heuristic approaches and lower bounds were made with Microsoft Visual Studio 2012 (using C++ language) which was executed on a Pentium(R) Dual-Core PC with CPU@3.00 GHz and 1,00 Go RAM. The mathematical models were solved using Cplex 12.6 solver. The conflict graph of each instance is generated using the $G(n, p)$ Erdős Rényi method (see Algorithm 6) similarly to Section 2.5. In all the experiments, we have considered three values of p: $p \in \{0.2, 0.5, 0.8\}$. We begin by the results obtained for the case of unit-time operations, then we pass to the case of arbitrary processing times. Note that the bold values in the tables that follow indicate the best results.

3.8.1 Case of unit-time operations

In this section we will be concerned with the branch and bound algorithm and the MILP models proposed for the problem $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$. The number of jobs n we considered is between 10 to 20000 jobs. For each pair (n, p), a set of 10 instances is generated. The percentage deviation from the optimal solution is computed by the following formulation

$$\frac{|\text{optimal solution} - x|}{\text{optimal solution}} * 100$$

where x is a lower bound or an upper bound.

3.8.1.1 Performance of the lower bounds

In this section, we report the performance of the seven lower bounds presented in Section 3.6.3. We observed from the implementation that lb_3 and lb_6 outperform all the remaining lower bounds for all the instance classes and according to all the criteria. Thus, the results corresponding to these later lower bounds have been removed.

Table 3.8.1: Performance of the lower bounds lb_3 and lb_6.

		(lb_3)					(lb_6)				
p	n	a	b	c	d	e	a	b	c	d	e
0.2	10	100	100	0.000	0.000	-	100	100	0.000	0.000	0.064
	20	100	100	0.000	0.000	-	100	100	0.000	0.000	0.067
	40	100	100	0.000	0.000	-	100	100	0.000	0.000	0.092
	80	100	100	0.000	0.000	-	100	100	0.000	0.000	0.926
	100	100	100	0.000	0.000	-	100	100	0.000	0.000	0.207
	150	100	100	0.000	0.000	-	100	100	0.000	0.000	0.953
	200	100	100	0.000	0.000	-	100	100	0.000	0.000	1.010
	300	100	100	0.000	0.000	-	100	100	0.000	0.000	5.259
	400	100	100	0.000	0.000	-	100	100	0.000	0.000	10.910
	500	100	100	0.000	0.000	-	100	100	0.000	0.000	24.996
	600	100	100	0.000	0.000	-	100	100	0.000	0.000	53.861
	700	100	100	0.000	0.000	-	100	100	0.000	0.000	109.297
	800	100	100	0.000	0.000	-	100	100	0.000	0.000	90.676
0.5	10	90	100	0.000	0.769	-	90	100	0.000	0.769	0.045
	20	100	100	0.000	0.000	-	100	100	0.000	0.000	0.063
	40	100	100	0.000	0.000	-	100	100	0.000	0.000	0.067
	80	100	100	0.000	0.000	-	100	100	0.000	0.000	0.845
	100	100	100	0.000	0.000	-	100	100	0.000	0.000	0.208
	150	100	100	0.000	0.000	-	100	100	0.000	0.000	0.705
	200	100	100	0.000	0.000	-	100	100	0.000	0.000	0.874
	300	100	100	0.000	0.000	-	100	100	0.000	0.000	4.478
	400	100	100	0.000	0.000	-	100	100	0.000	0.000	9.941
	500	100	100	0.000	0.000	-	100	100	0.000	0.000	21.693
	600	100	100	0.000	0.000	-	100	100	0.000	0.000	50.061
	700	100	100	0.000	0.000	-	100	100	0.000	0.000	90.678
	800	100	100	0.000	0.000	-	100	100	0.000	0.000	76.127
0.8	10	80	90	0.1	1.547	-	10	20	0.74	6.326	0.050
	20	80	100	0	0.909	-	50	70	0.285	2.186	0.064
	40	100	100	0.000	0.000	-	100	100	0.000	0.000	0.095
	80	100	100	0.000	0.000	-	100	100	0.000	0.000	0.853
	100	100	100	0.000	0.000	-	100	100	0.000	0.000	0.215
	150	100	100	0.000	0.000	-	100	100	0.000	0.000	0.643
	200	100	100	0.000	0.000	-	100	100	0.000	0.000	0.890
	300	100	100	0.000	0.000	-	100	100	0.000	0.000	3.703
	400	100	100	0.000	0.000	-	100	100	0.000	0.000	7.243
	500	100	100	0.000	0.000	-	100	100	0.000	0.000	18.094
	600	100	100	0.000	0.000	-	100	100	0.000	0.000	32.421
	700	100	100	0.000	0.000	-	100	100	0.000	0.000	59.631
	800	100	100	0.000	0.000	-	100	100	0.000	0.000	57.412

a(%): Number of times in percentage where a given lower bound coincides with the optimal solution.
b(%): Number of times in percentage where a given lower bound yields the best solution.
c: Average gap related to the best lower bound.
d(%): Average percentage deviation from the optimal solution.
e: Average CPU times in seconds. Dash "-" means that the CPU time is less than 0.004 milliseconds.

The results in Table 3.8.1 show that lb_3 and lb_6 are closely competitive. However, lb_3 requires less computer time than lb_6. Therefore, we consider only lb_3 at each node of the search tree.

3.8.1.2 Performance of the heuristic algorithms

We study in this section the performance of the heuristics developed in Section 3.6.4. We observed from the implementation that H_{11} and H_{12} outperform H_{21} and H_{22} for all the instance classes and according to all the performance criteria. This emerges from the fact that jobs with low conflict degree will reduce the idle times that occurred by processing jobs with high conflict degree at the beginning of the schedule. Thus, the results corresponding to H_{21} and H_{22} have been removed.

Table 3.8.2: Performance of the upper bounds (heuristics).

		(H_{11})					(H_{12})				
p	n	a	b	c	d	e	a	b	c	d	e
0.2	10	100	100	0.000	0.000	0.000	100	100	0.000	0.000	0.000
	20	100	100	0.000	0.000	0.000	100	100	0.000	0.000	0.000
	80	100	100	0.000	0.000	0.000	100	100	0.000	0.000	0.000
	100	100	100	0.000	0.000	0.000	100	100	0.000	0.000	0.000
	500	90	90	0.100	0.019	0.000	90	90	0.100	0.019	0.000
	1000	100	100	0.000	0.000	0.000	100	100	0.000	0.000	0.000
	5000	80	80	0.300	0.005	**0.001**	80	80	0.300	0.005	0.002
	10000	70	90	0.100	0.002	**0.002**	70	90	0.100	0.002	0.005
	15000	80	90	0.100	0.0013	**0.003**	**90**	**100**	**0.000**	**0.0006**	0.007
	20000	90	90	0.100	0.0004	**0.005**	**100**	**100**	**0.000**	**0.000**	0.009
0.5	10	80	90	0.100	1.818	0.000	80	90	0.100	1.818	0.000
	20	**80**	**100**	**0.000**	**0.952**	0.000	70	90	0.100	1.428	0.000
	80	**80**	**100**	**0.000**	**0.370**	0.000	70	90	0.100	0.493	0.000
	100	50	80	0.200	0.594	0.000	**60**	**90**	**0.100**	**0.495**	0.000
	500	30	70	0.400	0.159	0.000	30	70	**0.300**	**0.139**	0.000
	1000	30	60	0.400	0.079	0.000	**40**	**70**	**0.300**	**0.069**	0.000
	5000	20	40	0.600	0.019	**0.001**	**30**	**70**	**0.400**	**0.015**	0.002
	10000	50	70	**0.600**	**0.007**	**0.004**	50	70	0.700	0.008	0.006
	15000	20	50	0.600	0.0086	**0.005**	20	**60**	**0.500**	**0.0079**	0.008
	20000	20	50	0.500	0.0044	**0.007**	**40**	**80**	**0.200**	**0.0029**	0.010
0.8	10	30	30	1.100	8.393	0.000	**100**	**100**	**0.000**	**0.000**	0.000
	20	0	20	1.500	11.501	0.000	**30**	**100**	**0.000**	**4.612**	0.000
	80	**10**	60	0.500	2.345	0.000	0	**80**	**0.200**	**1.975**	0.000
	100	**10**	**90**	**0.100**	**1.386**	0.000	0	50	0.500	1.782	0.000
	500	0	50	0.700	0.598	0.000	**10**	**80**	**0.400**	**0.538**	0.000
	1000	0	30	1.100	0.319	0.000	**10**	**80**	**0.400**	**0.249**	0.000
	5000	0	60	1.300	0.073	0.003	0	**70**	**0.500**	**0.057**	**0.002**
	10000	0	30	1.500	0.042	0.007	0	**70**	**0.400**	**0.031**	**0.006**
	15000	0	40	1.000	0.027	0.010	0	**80**	**0.200**	**0.022**	**0.008**
	20000	0	30	1.000	0.021	0.014	0	**60**	**0.600**	**0.019**	**0.011**

a(%): Number of times in percentage a given upper bound coincides with the optimal solution.
b(%): Number of times in percentage a given upper bound yields the best solution.
c: Average gap related to the best lower bound.
d(%): Average percentage deviation from the optimal solution.
e: Average CPU times in seconds.

By looking at the results of Table 3.8.2, H_{12} seems to be more efficient than H_{11} for large instances with high densities, whereas H_{11} performs well for small instances and problems with low densities. Based on these results, we only invoke H_{11} and H_{12} when computing the upper bound of a node of the branch and bound tree.

Table 3.8.3: Performance of B&B1, B&B3 and B&B4.

	$p = 0.2$				$p = 0.5$				$p = 0.8$			
n	a	b	c	d	a	b	c	d	a	b	c	d
B&B1												
10	1	0	100	0.026	81.8	10	90	0.029	32.2	0	100	0.44
20	1	0	100	0.024	4	10	90	0.028	1033.7	2.327	97.673	0.042
40	1.1	0	100	0.022	4.9	10	90	0.023	182	15.936	84.064	0.038
80	1	0	100	0.019	92.3	11.935	88.065	0.024	829.4	27.659	72.341	0.039
100	1	0	100	0.021	254.8	3.034	96.966	0.030	437	32.155	67.845	0.040

(Continued)

Table 3.8.3 . *(Continued)*

150	5.5	0	100	0.038	24.3	2.976	97.024	0.034	2151.3	21.043	78.957	0.052
200	1.3	0	100	0.025	5.2	12.5	87.5	0.034	7679.8	25.166	74.834	0.070
250	1.8	0	100	0.021	2610	21.606	78.394	0.045	9680.6	22.404	77.596	0.074
300	1	0	100	0.028	8.1	12.719	87.281	0.034	15373.4	30.444	69.556	0.128
400	1.1	10	90	0.031	18.3	3.479	96.521	0.040	11496.4	30.976	69.024	0.132
450	2.7	0	100	0.029	52.7	19.042	80.958	0.042	32857.7	24.482	75.518	0.282
500	6.6	0	100	0.028	34.9	13.143	86.857	0.032	73213.2	26.875	73.125	0.634
800	1	0	100	0.020	20.4	7.617	92.383	0.033	192780	29.113	70.887	2.327
1000	1	0	100	0.020	3876.4	12.926	87.074	0.114	110436	22.468	77.532	1.870
2000	2.1	0	100	0.038	21.7	19.968	80.032	0.091	1521630	23.603	76.397	47.806
3000	1	0	100	0.040	36.1	20.932	79.068	0.104	2315670	23.957	76.043	113.488
4000	22.4	0	100	0.075	98464.3	6.552	93.448	8.422	5955040	23.345	76.655	366.228
5000	7.3	0.435	99.565	0.101	24.5	23.130	76.870	0.135	5566960	21.997	78.003	554.378
6000	6.2	0	100	0.109	134	3.328	96.672	0.218	8925250	24.478	75.522	1129.32
B&B3												
10	1	0	100	0.018	81.2	10	90	0.025	32.2	0	100	0.039
20	1	0	100	0.017	2.3	10	90	0.022	667.7	4.348	95.652	0.042
40	1.1	0	100	0.019	1.3	10	90	0.022	9.4	10.873	89.127	0.035
80	1	0	100	0.015	3.2	11.364	88.636	0.020	166.2	29.840	70.160	0.032
100	1	0	100	0.016	7.6	1.923	98.077	0.025	9.4	34.312	65.688	0.035
150	4.6	0	100	0.036	11.9	3.895	96.105	0.030	27.7	19.875	80.125	0.039
200	1.3	0	100	0.020	2.3	16.667	83.333	0.029	42.3	28.851	71.149	0.039
250	1.6	0	100	0.031	16.9	17.067	82.933	0.045	49.2	23.267	76.733	0.049
300	1	0	100	0.020	3.7	5.526	94.474	0.026	59.7	29.504	70.496	0.047
400	1.1	10	90	0.024	4.9	3.553	96.447	0.030	79.4	24.314	75.686	0.051
450	2.4	0	100	0.030	25.6	17.671	82.329	0.040	74.1	21.839	78.161	0.056
500	1.6	1.429	98.571	0.022	17	18.572	81.428	0.031	96.5	20.719	79.281	0.059
800	1	0	100	0.019	10.7	6.319	93.681	0.037	229.8	31.111	68.889	0.110
1000	1	0	100	0.024	28.2	10.417	89.583	0.049	144.2	23.301	76.699	0.103
2000	1.9	0	100	0.049	6.9	18.445	81.555	0.109	426.2	23.131	76.869	0.404
3000	1	0	100	0.031	17.4	22.025	77.975	0.078	599	25.680	74.320	0.765
4000	17.5	0	100	0.086	52.1	8.226	91.774	0.172	506.9	21.445	78.555	1.184
5000	6.2	0.435	99.565	0.069	16.5	12.609	87.391	0.122	745	26.045	73.955	2.285
6000	4.9	0	100	0.069	62.2	7.990	92.010	0.316	1672	26.410	73.590	6.902
7000	55.2	0	100	0.240	188.3	17.978	82.022	0.897	1290.2	23.369	76.631	7.463
8000	16	1	99	0.131	71.7	29.078	70.922	0.511	1390.8	25.420	74.580	9.209
9000	37.2	0	100	0.256	96.1	12.189	87.811	0.616	1285.1	20.804	79.196	9.143
10000	115.3	0.136	99.864	0.575	88.6	27.647	72.353	0.615	1038.6	21.054	78.946	8.284
15000	9.1	0	100	0.264	398.5	16.313	83.687	4.318	2516.4	27.064	72.936	33.771
20000	1	0	100	0.092	101.6	12.006	87.994	1.589	1962.3	21.170	78.830	38.439
B&B4												
10	1	0	100	0.024	81.4	12.5	87.5	0.029	32.2	0	100	0.044
20	1	0	100	0.019	1.3	13.333	86.667	0.024	6626.9	5.538	94.462	0.047
40	1.1	0	100	0.028	1.5	10	90	0.027	19724.2	11.943	88.057	0.036
80	1	0	100	0.019	3.3	10	90	0.024	455.2	32.354	67.646	0.035
100	1	0	100	0.020	12.3	2.667	97.333	0.029	871.1	37.693	62.307	0.044
150	1.6	0	100	0.037	126.5	1.144	98.856	0.035	1967.3	29.544	70.456	0.051
200	1.1	0	100	0.027	1.8	17.5	82.5	0.032	6474.3	23.418	76.582	0.066
250	1.2	0	100	0.022	152.8	12.941	87.059	0.036	3610	18.121	81.879	0.051
300	1	0	100	0.026	6	5.619	94.381	0.034	11372.3	29.234	70.766	0.110
400	1.1	10	90	0.032	12.3	2.957	97.043	0.039	38320.5	22.385	77.615	0.289
450	2.1	0	100	0.023	1293.7	17.751	82.249	0.044	32240.7	18.925	81.075	0.271
500	5.3	0	100	0.025	11.5	12	88	0.034	43940.6	18.432	81.568	0.402
800	1	0	100	0.022	4.1	6.25	93.75	0.034	148890	25.692	74.308	1.869
1000	1	0	100	0.025	3750.9	7.304	92.696	0.116	254519	22.825	77.175	4.293
2000	1.4	0	100	0.083	10.8	19.475	80.525	0.071	763874	20.501	79.499	29.245
3000	1	0	100	0.032	21.6	21.224	78.776	0.098	2650140	22.416	77.584	156.017

(Continued)

Table 3.8.3 . *(Continued)*

4000	17.4	0	100	0.098	94	2.302	97.698	0.217	8238790	26.951	73.049	688.221

a: Average number of generated nodes.
b(%): Average usage of H_{11} in percentage.
c(%): Average usage of H_{12} in percentage.
d: Average CPU times in seconds.

3.8.1.3 Impact of the selecting rule

In what follows, we discuss the empirical performance of different versions of the branch and bound algorithm by varying the selecting rule. Let B&B1, B&B2, B&B3 and B&B4 be the versions of the branch and bound algorithm associated with the selecting rules Decrease_conflict, Increase_conflict, Best_heuristic and No_sort respectively.

We first observed that B&B2 is weaker compared to the others. For only $n = 150$, we obtained an instance that cannot be solved to optimality within 3600 s. Thus the results corresponding to B&B2 have been removed. Table 3.8.3 summarizes the results we obtained on the remaining versions. We stopped the simulation of B&B1 (resp B&B4) at $n = 6000$ (resp $n = 4000$), because from $n = 7000$ (resp $n = 5000$) we started to find instances that reach the time limit of one hour.

The evaluation shows that the selecting rule is essential for the performance of the branch and bound algorithm, as it significantly improves the solution time in comparison to the "No_sort" strategy. Among the four selecting rules, B&B3 seems to work systematically better in most of the cases. This is due to the good performance of the permutation schedules produced by H_{11} and H_{12}. However, B&B1 shows also a good performance. This can be explained by the same fact that makes H_{11} and H_{12} better than H_{21} and H_{22}.

3.8.1.4 Computational results for the MILP models and the branch and bound algorithm

From Tables 3.8.3 and 3.8.4, we report the performance of the MILP models (P_3), (P_4) and (P_5) and the B&B3 version.

By looking at the results of the MILP models, (P_4) appears to be more efficient than (P_3) and (P_5) among all the instance classes, and solves all the instances of size up to 450 jobs. For only $n = 20$ jobs, (P_5) fails to solve all the instances (CPLEX threads: out of memory status). We stopped the simulation for (P_5) at $n = 20$ jobs as we are confident that this model cannot solve larger instances. However, (P_3) generates optimal solutions for instances of size up to 100 jobs. Regarding the size complexity of the MILP models (Table 3.5.1), (P_5) requires $\frac{1}{2}n(n-1)$ binary variables, a number less than half that required by (P_4), but

Table 3.8.4: Performance of the MILP models (P_3), (P_4) and (P_5).

	$p = 0.2$			$p = 0.5$			$p = 0.8$		
n	a	b	c	a	b	c	a	b	c
(P_3)									
10	0	0.168	0	0	0.151	0	0	0.289	0
20	0	0.242	0	0	0.419	0	0	4.418	0
40	0	0.944	0	0	3.939	0	0	36.233	0
60	0	5.954	0	0	91.618	0	0	455.356	0
80	0	19.219	0	0	527.723	0	0	3469.16	4
100	0	104.951	0	0	4218.29	5	0	17142	10
(P_5)									
10	0	30.189	0	0	1.312	0	0	5.578	0
20	10	-	-	10	-	-	10	-	-
(P_4)									
10	0	0.169	0	0	0.208	0	0	0.194	0
20	0	0.255	0	0	0.235	0	0	0.259	0
40	0	0.417	0	0	0.491	0	0	0.496	0
60	0	0.894	0	0	0.800	0	0	0.840	0
80	0	1.305	0	0	1.869	0	0	2.479	0
100	0	2.252	0	0	2.331	0	0	3.491	0
150	0	20.315	0	0	27.026	0	0	26.057	0
200	0	97.207	0	0	65.084	0	0	95.420	0
250	0	182.321	0	0	162.057	0	0	204.708	0
300	0	309.531	0	0	337.549	0	0	262.232	0
400	0	861.103	0	0	594.269	0	0	1070.07	0
450	0	1328.49	0	0	718.971	0	0	1101.42	0

a: Number of unsolved instances (CPLEX threads: out of memory status).
b: Average CPU times in seconds.
c: Number of instances solved after 3600 s.

at a cost of an additional $\frac{1}{2}n(n-1)$ continuous variables. Furthermore, (P_4) requires more constraints than (P_5). Moreover, (P_3) has more binary variables and constraints than (P_5). Thus, the size complexity may not be sufficient to assess the computational complexity of various mathematical models.

Interestingly, the B&B3 version succeed to solve all the instances of size up to 20000 jobs in an acceptable amount of time. From Table 3.8.3, the average CPU time required to solve an instance of 20000 jobs is about 0.092 s for $p = 0.2$, 1.589 s for $p = 0.5$ and 38.439 s for $p = 0.8$. These results clearly indicate that the B&B3 version is more effective than the three MILP models.

When considering the density of the conflict graphs, it seems that as the value of p increases, the CPU time and the number of generated nodes of the branch and bound algorithm increases too. Indeed, when the value of p increases the density of the agreement graph decreases and the problem of finding a minimum number of vertex-disjoint paths that cover all the vertices of the agreement graph becomes harder to solve.

Table 3.8.5: Average CPU times (in seconds) required by B&B3 and (P_4) for $p \geq 0.8$.

	$n = 10$		$n = 20$		$n = 30$		$n = 40$		$n = 50$	
p	B&B3	(P_4)	B&B3	(P_4)	B&B3	(P_4)	B&B3	(P_4)	B&B3	(P_4)
0.8	0.039	0.194	0.042	0.259	0.032	0.476	0.035	0.496	0.033	0.815
0.85	0.032	0.256	0.042	0.303	0.042	0.457	0.036	0.767	0.032	0.874
0.88	0.033	0.249	1.560	0.307	0.085	0.454	8.289	1.418	0.041	1.224
0.89	0.035	0.258	0.038	0.250	5.486	0.633	146.482	1.035	>360.043	>362.382
0.9	0.036	0.213	0.035	0.296	0.055	0.425	>727.017	1.902	1.348	>364.98
0.91	0.036	0.236	4.117	0.321	0.213	0.369	67.440	0.925	>720.109	6.213
0.92	0.036	0.227	>1080.41	0.300	>324.001	0.552	>525.261	1.880	>1650.57	5.513
0.93	0.035	0.267	205.230	0.237	4.299	0.452	>1835.38	1.854	>837.498	4.041
0.94	0.036	0.235	0.028	0.307	>2575.21	0.370	>2375.06	1.543	>2168.61	2.593
0.95	0.036	0.283	0.043	0.280	>2160.02	0.364	>2545.25	1.337	>3186.01	2.758
0.96	0.033	0.248	0.030	0.285	>1800.02	0.383	>3240	1.115	>3600	2.655
0.97	0.036	0.247	0.040	0.233	>2160.01	0.409	>2160.01	1.192	>3600	2.685
0.98	0.037	0.273	0.033	0.330	0.038	0.412	>1080.02	1.192	>2520.01	2.678
0.99	0.038	0.248	0.042	0.233	0.038	0.428	>360.028	1.319	>1080.02	1.818
1	0.037	0.293	0.034	0.236	0.038	0.223	0.040	0.267	0.037	0.842

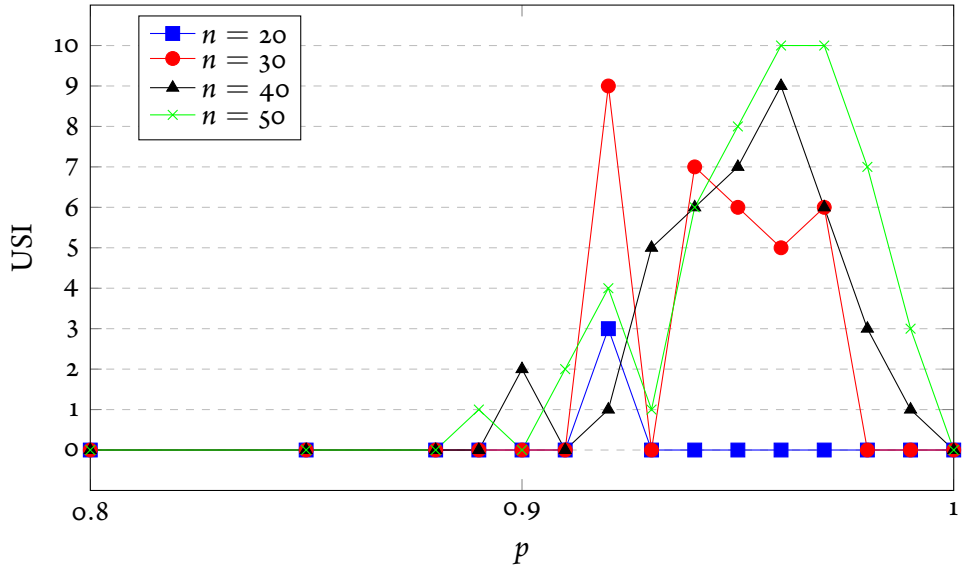


Figure 3.8.1: B&B3: Number of unsolved instances as a function of p.

3.8.1.5 Impact of the density of the conflict graph

We now study the impact of the density of the conflict graph on B&B3 and (P_4) when $p \geq 0.8$. We introduced new instance classes with $p \in \{0.8, 0.85, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93, 0.94, 0.95, 0.96, 0.97, 0.98, 0.99, 1\}$ and $n \in \{10, 20, 30, 40, 50\}$. For each pair (n, p), 10 instances are randomly generated. Each instance is either solved to optimality or stopped due to a CPU time limit of 3600 s. Figure 3.8.1 summarizes the number of unsolved instances after 3600 s (denoted by USI) as a function of the considered values of p for the B&B3. Table 3.8.5 reports the average CPU times required by B&B3 and (P_4). The inequality sign '>' in Table 3.8.5 indicates that the average CPU time needed to find the optimum is bigger than the given value. In fact, for each instance when the time limit of 3600 s is reached the algorithm is terminated and a CPU time of 3600 s is associated to this instance.

From Figure 3.8.1 and Table 3.8.5, we observe that the most difficult value of p tends to one as the value of n increases. For $n = 20$, the difficult instance class is obtained for $p = 0.92$ with 3 unsolved instances, whereas for $n = 50$, the difficult instance classes are obtained for $p = 0.96$ and $p = 0.97$ with 10 unsolved instances. However, for $p = 1$ the branch and bound algorithm performs well and generates only one node since all the jobs are in conflict (the lower bound of the root node is equal to its upper bound). Regarding (P_4), this model has a more stable performance than B&B3 and succeed to solve all the instance classes except for $n = 50$ with $p = 0.89$ and $p = 0.9$, in which (P_4) fails to solve only one instance in less than 3600 s. However, for the instance classes that seem to be easily solved by B&B3, (P_4) requires more computation time than the B&B3 version.

To complete the study, we considered all the instances for which B&B3 reaches the time limit (128 instances), and we compared the best solution found after 3600 s with the optimal solution given by (P_4). Interestingly, the gap between the two values is null in 123 of these instances. Therefore, the decrease in the efficiency of the B&B3 version is due to the lower bounds. Indeed, we observed the existence of jobs that are neither in L nor in S (see Section 3.6.3 for lb_3), these jobs increase the number of idle times in the optimal schedule and they are difficult to detect. This affects the tightness of the lower bounds and slows down the algorithm.

3.8.2 Case of arbitrary processing times

We studied in the previous section the performance of the solution methods proposed for the restricted case of two machines with unit-time operations. Now, our focus is on the general case of arbitrary processing times in which we will discuss the performance of the MILP models (P_1) and (P_2) and the heuristic approaches H1mFSCn and H2mFSCn.

The experiments are carried out on two sets of problems. The first set contains small size instances with known optimal solutions to which the heuristic's approach solutions can be compared. The optimal solutions are obtained from the models (P_1) and (P_2). The instances can be classified into three classes for three values of m: $m \in \{2, 3, 5\}$ ranging in size from 3 job-2 machine to 11 job-2 machine, from 3 job-3 machine to 8 job-3 machine and from 3 job-5 machine to 5 job-5 machine. The processing times of the operations are randomly generated from a uniform distribution. For each triple (n, m, p), 20 instances are built, where the processing times of 10 of these instances are generated from $\{1, \ldots, 10\}$, and from $\{1, \ldots, 20\}$ for the remaining 10 instances. The computational results on this set of instances are summarized in Table 3.8.7. Another group, with small number of instances, has been generated to compare between (P_1) and (P_2). This group contains 270 instances ranging in size from 3 job-2 machine to 11 job-2 machine. The processing times of the operations are generated randomly from a uniform distribution. For each pair (n, p), 10 instances are built, where the processing times of 5 of these instances are generated from $\{0, \ldots, 10\}$, and from $\{0, \ldots, 20\}$ for the remaining 5 instances. The computational results on these instances are summarized in Table 3.8.6.

The second set of problems contains 4500 large instances with unknown optimal solution. In this case, the lower bounds proposed in Section 3.4 are used to evaluate the heuristic approaches. The number of machines and jobs that we have considered are: $m \in \{5, 10, 20\}$, $n \in \{10, 20, 50, 100, 200\}$. For each triple (n, m, p), a set of 100 instances is generated, let $S(n, m, p)$ denotes this set. The processing times of the operations are randomly generated from a uniform distribution ranging from 1 to 10 for 25 instances of $S(n, m, p)$, from 1 to 20 for 25 instances of $S(n, m, p)$, from 1 to 50 for 25 instances of $S(n, m, p)$ and from 50 to 100 for the remaining 25 instances of $S(n, m, p)$. Tables 3.8.8 and 3.8.9 and Figures 3.8.2, 3.8.3, 3.8.4, 3.8.5, 3.8.6 and 3.8.7 summarize the different results we obtained on this set of instances.

Before proceeding to the analysis of the results note that for each heuristic approach, 64 *heuristic variants* are derived, each one represents a combination of a priority order (PO_k used in H1mFSCn and PO'_k used in H2mFSCn) and an order of selection (OS_k used when calling Algorithm 7 in H1mFSCn and H2mFSCn). Thus, each heuristic variant of H1mFSCn (resp. H2mFSCn) is denoted by the couple $(OS_k, PO_{k'})$ (resp.$(OS_k, PO'_{k'})$), $k = \overline{1, 8}$, $k' = \overline{1, 8}$.

We begin with the results obtained on the first set of instances, in which we study the performance of (P_1) and (P_2) and compare their results with the heuristic approaches H1mFSCn and H2mFSCn. Recall that (P_1) generates the optimal permutation schedule whereas (P_2) gives the optimal schedule. Then, we pass to the results of the second set of instances. We first compare between the results obtained by the lower bounds proposed in Section 3.4. Then, we discuss the performance of the two heuristic approaches H1mFSCn and H2mFSCn.

Table 3.8.7: Computational results of (P_1), H1mFSCn and H2mFSCn on small instances.

			H1mFSCn						H2mFSCn				
	m	n	a	b	c	d	e	(P_1)-CPU(s)	a	b	c	d	e
$p = 0.2$	2	3	19	0	1	0.003	0.000	0.117	19	0	1	0.003	0.000
		4	17	2	1	0.012	0.000	0.136	17	2	1	0.012	0.000
		5	**12**	5	3	**0.021**	0.000	0.145	9	6	5	0.029	0.000
		6	**13**	3	4	**0.020**	0.000	0.173	12	2	6	0.025	0.000
		7	**15**	4	1	**0.007**	0.000	0.314	13	3	4	0.017	0.000
		8	**12**	6	2	**0.016**	0.000	1.292	10	7	3	0.020	0.000
		9	**15**	4	1	**0.006**	0.000	11.049	11	6	3	0.014	0.000
		10	**18**	2	0	**0.002**	0.000	84.164	16	4	0	0.005	0.000
		11	**17**	2	1	**0.004**	0.000	819.715	16	3	1	0.006	0.000
$p = 0.5$		3	19	0	1	0.005	0.000	0.129	19	0	1	0.005	0.000
		4	16	3	1	**0.007**	0.000	0.144	16	2	2	0.009	0.000
		5	12	3	5	0.023	0.000	0.164	**13**	3	4	**0.018**	0.000
		6	**13**	5	2	**0.015**	0.000	0.236	10	5	5	0.025	0.000
		7	**7**	10	3	**0.028**	0.000	0.774	4	8	8	0.039	0.000
		8	**9**	9	2	**0.018**	0.000	1.751	8	7	5	0.029	0.000

(Continued)

Table 3.8.7 . *(Continued)*

		9	**8**	11	1	**0.019**	0.000	4.369	5	12	3	0.029	0.000
		10	**8**	10	2	**0.018**	0.000	29.517	3	10	7	0.035	0.000
		11	**6**	9	5	**0.029**	0.000	224.107	5	8	7	0.044	0.000
$p = 0.8$		3	20	0	0	0.000	0.000	0.135	20	0	0	0.000	0.000
		4	19	1	0	0.001	0.000	0.174	19	1	0	0.001	0.000
		5	16	4	0	**0.005**	0.000	0.655	16	3	1	0.008	0.000
		6	**18**	2	0	**0.003**	0.000	2.972	17	3	0	0.005	0.000
		7	**16**	4	0	**0.004**	0.000	12.036	12	8	0	0.009	0.000
		8	**12**	6	2	**0.015**	0.000	49.325	9	10	1	0.017	0.000
		9	**13**	7	0	**0.006**	0.000	527.710	9	10	1	0.016	0.000
		10	**12**	8	0	**0.008**	0.000	691.416	9	10	1	0.012	0.000
		11	**9**	10	1	**0.016**	0.000	8783.3	6	13	1	0.018	0.000
$p = 0.2$	3	3	20	0	0	0.000	0.000	0.126	20	0	0	0.000	0.000
		4	17	3	0	0.004	0.000	0.151	17	3	0	0.004	0.000
		5	**13**	5	2	**0.014**	0.000	0.169	11	5	4	0.019	0.000
		6	**8**	5	7	**0.042**	0.000	0.253	4	6	10	0.063	0.000
		7	**5**	6	9	**0.045**	0.000	0.616	3	2	15	0.071	0.000
		8	6	11	3	**0.023**	0.000	1.430	6	7	7	0.035	0.000
$p = 0.5$		3	20	0	0	0.000	0.000	0.133	**20**	0	0	0.000	0.000
		4	**17**	0	3	**0.010**	0.000	0.188	15	0	5	0.019	0.000
		5	**12**	5	3	**0.024**	0.000	0.390	11	6	3	0.025	0.000
		6	**8**	10	2	**0.022**	0.000	1.452	5	11	4	0.035	0.000
		7	**9**	5	6	**0.024**	0.000	4.878	6	8	6	0.032	0.000
		8	**3**	10	7	**0.047**	0.000	6.960	2	6	12	0.064	0.000
$p = 0.8$		3	19	0	1	0.002	0.000	0.169	19	0	1	0.002	0.000
		4	16	4	0	0.005	0.000	1.021	16	4	0	0.005	0.000
		5	17	2	1	0.008	0.000	32.890	17	2	1	0.008	0.000
		6	**16**	4	0	**0.004**	0.000	171.587	15	5	0	0.006	0.000
		7	**14**	5	1	**0.007**	0.000	1324.55	13	6	1	0.009	0.000
		8	**14**	4	2	**0.012**	0.000	5127.47	9	8	3	0.021	0.000
$p = 0.2$	5	3	18	2	0	0.003	0.000	0.129	18	2	0	0.003	0.000
		4	**15**	4	1	0.008	0.000	0.948	14	5	1	0.008	0.000
		5	**7**	4	9	**0.061**	0.000	1.249	6	4	10	0.066	0.000
$p = 0.5$		3	19	1	0	0.001	0.000	1.262	19	1	0	0.001	0.000
		4	**17**	0	3	**0.010**	0.000	8.386	16	1	3	0.014	0.000
		5	**9**	8	3	**0.029**	0.000	2965.73	8	9	3	0.031	0.000
$p = 0.8$		3	20	0	0	0.000	0.000	4.496	20	0	0	0.000	0.000
		4	18	2	0	0.001	0.000	38.106	18	2	0	0.001	0.000
		5	**18**	2	0	**0.002**	0.000	4697.05	17	3	0	0.003	0.000

(Continued)

Table 3.8.7 . *(Continued)*

a: number of problems with deviation of 0.

b: number of problems with deviation of $0 < to \leq 0.05$.

c: number of problems with deviation of > 0.05.

d: average deviation from the optimal solution obtained by (P_1) of a given heuristic variant.

e: average CPU times in seconds.

3.8.2.1 Performance of (P_1), (P_2) and the heuristic approaches on small instances

We discuss in this section the results of Tables 3.8.6 and 3.8.7 obtained on the first set of instances. Note that, when one of the models fails to solve an instance (CPLEX threads: out of memory status), we replace this instance by another one generated randomly and that can be solved by the two models. Then, we repeat the tests on the same set of instances. Dashes "-" in the cells of Table 3.8.6 mean that one of the two models fails to solve more than five instances. Thus, it is not important to generate other instances.

From Table 3.8.6, we observed that (P_1) outperforms (P_2) according to all the criteria ('a','b' and 'c'), except for $n = 3$, in which (P_2) shows a good performance. Regarding the solutions they produced, the makespan of the optimal permutation schedule coincides with the optimal makespan for all the instance classes. The CPU time required by the two models increases dramatically with the increasing values of n. Thus, it is difficult to obtain exact solutions for large instances (more than 11 jobs and 2 machines) in an acceptable time.

For each triple (n, m, p) in Table 3.8.7, we run all the 64 heuristic variants of a given heuristic approach and we select the heuristic variant that yields to the best average deviation from the optimal solution obtained by (P_1) (i.e. criterion 'd' of Table 3.8.7). Then, we report in this table the results obtained by the selected heuristic variant. By looking at the obtained results, H1mFSCn outperforms H2mFSCn according to all the criteria except the CPU time, and for all the instance classes except for the triple $(5, 2, 0.5)$ in which H2mFSCn outperforms H1mFSCn. This is due to the fact that H1mFSCn computes more partial sequences than H2mFSCn, but this is not always efficient because for larger instances we obtained other results that will be discussed later. The best results found with the two heuristic approaches H1mFSCn and H2mFSCn (most of the results have been obtained by H1mFSCn except for $(5, 2, 0.5)$) can be summarized as follows: the average deviation from the optimal solution of (P_1) is about 0.016 for $p = 0.2$, 0.018 for $p = 0.5$ and 0.005 for $p = 0.8$ with respectively 68.61%, 59.17% and 79.72% of the problems were solved optimally.

Regarding the MILP model (P_1), the results in Table 3.8.7 show that the CPU time required by this model increases dramatically with the increasing values of n, m and p. For only 5 jobs and 5 machines, the average CPU required is about 1.249 s for $p = 0.2$, 2965.73 s for $p = 0.5$ and 4697.05 s for $p = 0.8$.

Table 3.8.6: Comparison between (P_1) and (P_2) for $m = 2$.

		(P_2)				(P_1)		
p	n	a	b	c	d	a	b	c
0.2	3	0	4	0.128	0	0	**6**	**0.115**
	4	0	4	**0.130**	0	0	**6**	0.136
	5	0	4	0.156	0	0	**6**	**0.153**
	6	0	2	0.270	0	0	**8**	**0.175**
	7	0	0	0.432	0	0	**10**	**0.252**
	8	0	0	2.806	0	0	**10**	**0.884**
	9	0	0	29.565	0	0	**10**	**4.020**
	10	0	0	349.012	0	0	**10**	**30.486**
	11	0	0	4066.05	0	0	**10**	**290.498**
0.5	3	0	5	0.119	0	0	5	0.119
	4	0	3	0.151	0	0	**7**	**0.140**
	5	0	1	0.198	0	0	**9**	**0.153**
	6	0	1	0.429	0	0	**9**	**0.243**
	7	0	2	1.941	0	0	**8**	**0.578**
	8	0	0	2.168	0	0	**10**	**0.714**
	9	0	0	38.277	0	0	**10**	**4.755**
	10	0	0	40.157	0	0	**10**	**4.476**
	11	0	0	790.673	0	0	**10**	**22.452**
0.8	3	0	**6**	**0.123**	0	0	4	0.132
	4	0	4	0.206	0	0	**6**	**0.173**
	5	0	1	0.964	0	0	**9**	**0.283**
	6	0	0	4.348	0	0	**10**	**0.748**
	7	0	0	135.473	0	0	**10**	**5.574**
	8	2	0	353.043	0	**0**	**10**	**11.321**
	9	1	0	662.420	0	**0**	**10**	**26.252**
	10	4	0	5281.15	0	**1**	**10**	**232.489**
	11	8	-	-	-	**4**	-	-

a: Number of unsolved instances.
b: Number of times a given model yields the best CPU(s) time.
c: Average CPU time in seconds.
d: Average gap between the solutions of (P_1) and (P_2).

Thus, it is difficult to obtain exact solutions for large instances with high densities in an acceptable time.

3.8.2.2 Performance of the lower bounds

We report in this section the performance of the lower bounds developed in Section 3.4. The results of the simulation are provided in Table 3.8.8 and Figures 3.8.2, 3.8.3 and 3.8.4 where Best_LB is the number of times (in percentage) a given lower bound yields to the best solution among the rest of the lower bounds.

We observed from the implementation that LB_1, LB_2 and LB_5 outperform LB_3 and LB_4 for all the instance classes and according to the two performance criterion ('a' and 'b' of Table 3.8.8), thus the results

Table 3.8.8: Performance of the lower bounds.

			$m = 5$			$m = 10$			$m = 20$		
			$p = 0.2$	$p = 0.5$	$p = 0.8$	$p = 0.2$	$p = 0.5$	$p = 0.8$	$p = 0.2$	$p = 0.5$	$p = 0.8$
$n = 10$	LB_1										
		a	36%	89%	92%	80%	**86%**	90%	88%	86%	89%
		b	**0.1202**	**0.0050**	0.0032	**0.0279**	**0.0154**	0.0021	0.0137	**0.0123**	0.0024
	LB_2										
		a	34%	**91%**	**94%**	**83%**	**86%**	**96%**	**89%**	**87%**	**98%**
		b	0.1244	0.0066	**0.0025**	0.0295	0.0167	**0.0004**	**0.0120**	0.0174	**0.0002**
	LB_5										
		a	**64%**	2%	0%	11%	7%	2%	8%	6%	1%
		b	0.2708	0.1185	0.1423	0.0487	0.0586	0.0719	0.0227	0.0293	0.0348
$n = 20$	LB_1										
		a	2%	62%	75%	77%	82%	79%	80%	74%	**83%**
		b	0.3113	**0.0311**	0.0097	0.0277	0.0115	**0.0081**	**0.0177**	**0.0109**	**0.0034**
	LB_2										
		a	2%	**63%**	**82%**	**82%**	**91%**	**81%**	**85%**	**81%**	80%
		b	0.3110	0.0342	**0.0075**	**0.0245**	**0.0057**	0.0094	0.0201	0.0132	0.0128
	LB_5										
		a	**98%**	27%	0%	16%	1%	1%	12%	13%	4%
		b	**0.0014**	0.0809	0.1540	0.0441	0.0688	0.0778	0.0258	0.0308	0.0396
$n = 50$	LB_1										
		a	0%	0%	**66%**	0%	64%	**74%**	61%	**74%**	**69%**
		b	0.6145	0.3442	**0.0137**	0.3244	0.0224	**0.0073**	0.0425	**0.0140**	**0.0102**
	LB_2										
		a	0%	0%	61%	0%	**74%**	63%	**66%**	69%	62%
		b	0.6140	0.3376	0.0147	0.3224	**0.0151**	0.0159	0.0381	0.0226	0.0131
	LB_5										
		a	**100%**	**100%**	0%	**100%**	12%	3%	30%	12%	10%
		b	**0.0000**	**0.0000**	0.1621	**0.000**	0.0646	0.0731	**0.0242**	0.0338	0.0373
$n = 100$	LB_1										
		a	0%	0%	4%	0%	0%	**63%**	0%	**63%**	54%
		b	0.7737	0.5802	0.1011	0.5846	0.2271	**0.0133**	0.2610	**0.0245**	**0.0182**
	LB_2										
		a	0%	0%	4%	0%	0%	59%	0%	56%	**55%**
		b	0.7729	0.5780	0.0936	0.5778	0.2197	0.0191	0.2533	0.0346	0.0199
	LB_5										
		a	**100%**	**100%**	**93%**	**100%**	**100%**	5%	**100%**	26%	15%
		b	**0.0000**	**0.000**	**0.0018**	**0.0000**	**0.0000**	0.0766	**0.0000**	0.0304	0.0365
$n = 200$	LB_1										
		a	0%	0%	0%	0%	0%	**51%**	0%	5%	**51%**
		b	0.8688	0.7462	0.4251	0.7544	0.5274	0.0231	0.5474	0.1164	**0.0168**
	LB_2										
		a	0%	0%	0%	0%	0%	**51%**	0%	4%	48%
		b	0.8680	0.7463	0.4221	0.7538	0.5219	**0.0163**	0.5437	0.1185	0.0172
	LB_5										
		a	**100%**	**100%**	**100%**	**100%**	**100%**	2%	**100%**	**94%**	15%
		b	**0.0000**	**0.0000**	**0.0000**	**0.0000**	**0.0000**	0.0760	**0.0000**	**0.0010**	0.0350

a: number of times (in percentage) a given lower bound yields to the best solution among all the lower bounds.
b: average deviation from the best lower bound, where the deviation of the lower bound lb from the best lower bound LB is $(LB - lb)/LB$.

corresponding to these later lower bounds have been removed.

The evaluation shows that the lower bounds LB_1 and LB_2 are closely competitive (the curves of the two lower bounds are close in Figures 3.8.2, 3.8.3 and 3.8.4). The average difference between the results obtained by these two lower bounds is 2.75% for the criteria 'a' and 0.0036 for the criteria 'b'. We also observed that LB_5 outperforms LB_1 and LB_2 for instances with large number of jobs and for relatively low densities. This is due to the increase of the new ready times values (r_j') with the increasing number of jobs which in turn increases the value of LB_5. In addition, while decreasing the density, the number of conflicting jobs decreases too which decreases the performance of LB_1 and LB_2.

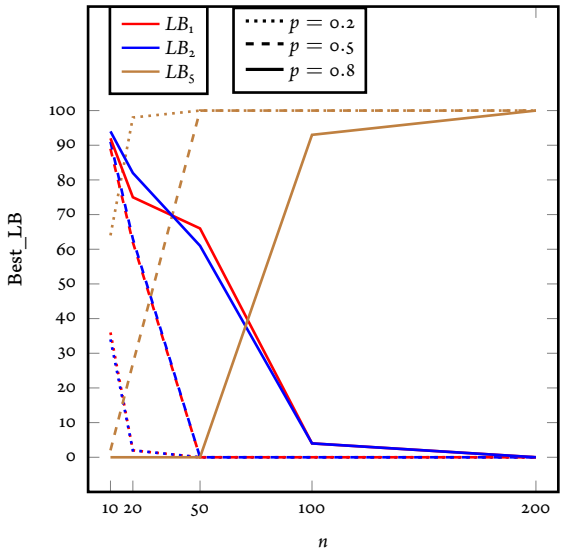


Figure 3.8.2: Performance of the lower bounds for $m = 5$.

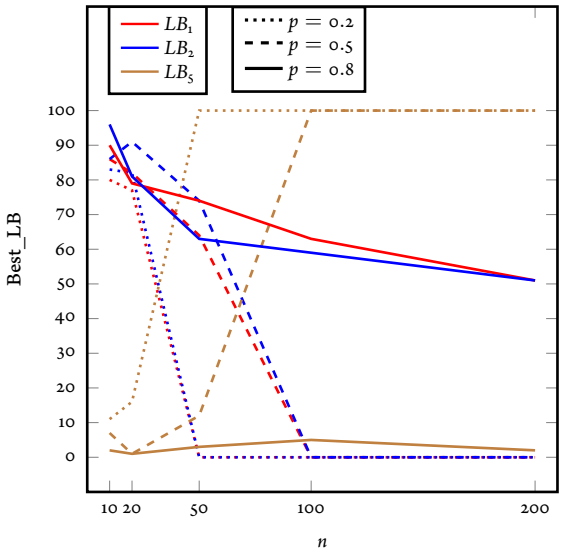


Figure 3.8.3: Performance of the lower bounds for $m = 10$.

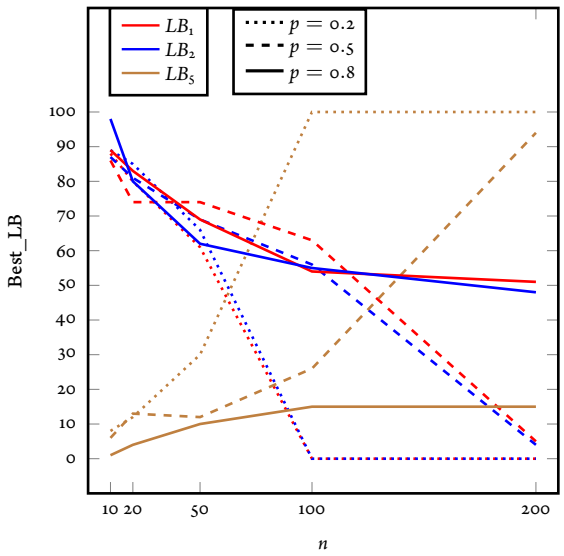


Figure 3.8.4: Performance of the lower bounds for $m = 20$.

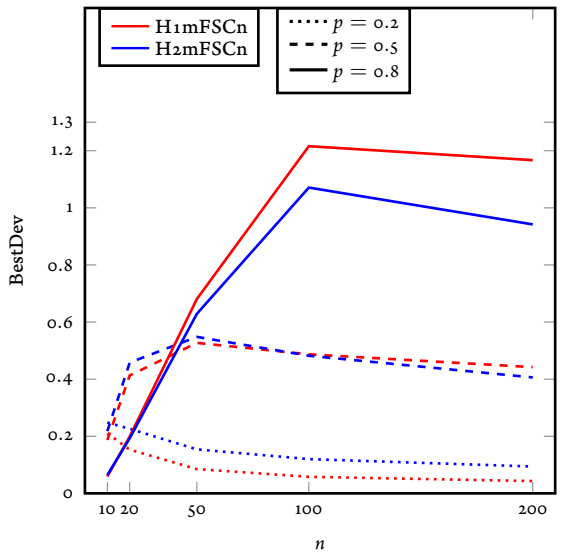


Figure 3.8.5: Comparison: H1mFSCn vs H2mFSCn for $m = 5$.

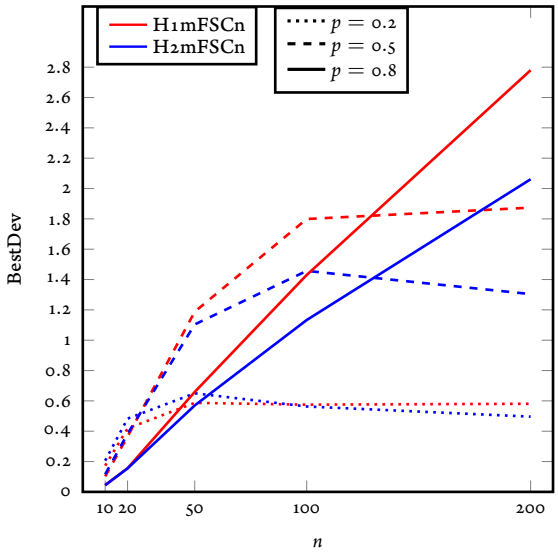


Figure 3.8.6: Comparison: H1mFSCn vs H2mFSCn for $m = 10$.

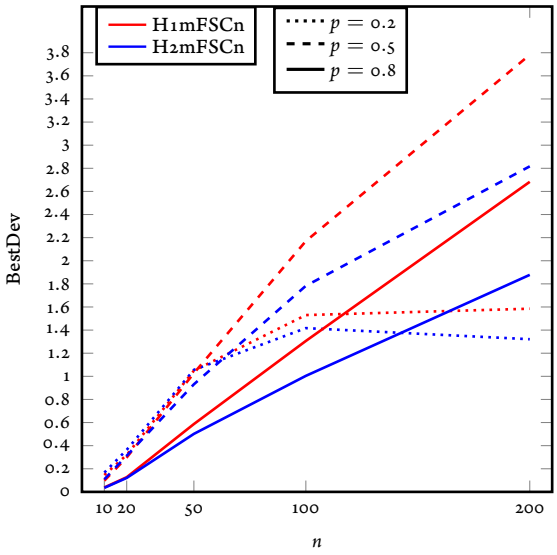


Figure 3.8.7: Comparison: H1mFSCn vs H2mFSCn for $m = 20$.

Table 3.8.9: Computational results of H1mFSCn and H2mFSCn on large instances.

			H1mFSCn			H2mFSCn		
m	n		p=0.2	p=0.5	p=0.8	p=0.2	p=0.5	p=0.8
5	10							
		AD(Nopt)	**0.265**(0)	**0.267**(1)	0.106(4)	0.282(0)	0.273(0)	**0.095**(5)
		CPU(s)	0.000	0.000	0.000	0.000	0.000	0.000
		BestHV	(OS_4, PO_7)	(OS_2, PO_3)	(OS_2, PO_5)	(OS_2, PO_7')	(OS_2, PO_1')	(OS_2, PO_4')
		BestDev	**0.210**	**0.187**	**0.059**	0.248	0.218	0.064
	20							
		AD(Nopt)	**0.221**(1)	**0.529**(0)	0.271(0)	0.268(0)	0.534(0)	**0.249**(0)
		CPU(s)	0.003	0.004	0.004	**0.001**	**0.002**	**0.003**
		BestHV	(OS_4, PO_5)	(OS_2, PO_3)	(OS_7, PO_5)	(OS_2, PO_5')	(OS_2, PO_5')	(OS_2, PO_5')
		BestDev	**0.153**	**0.413**	0.199	0.226	0.458	**0.193**
	50							
		AD(Nopt)	**0.130**(0)	0.654(0)	0.791(0)	0.183(0)	**0.607**(0)	**0.683**(0)
		CPU(s)	0.053	0.064	0.065	**0.012**	**0.022**	**0.036**
		BestHV	(OS_4, PO_3)	(OS_4, PO_5)	(OS_2, PO_5)	(OS_4, PO_4')	(OS_2, PO_4')	(OS_2, PO_5')
		BestDev	**0.085**	**0.527**	0.681	0.154	0.548	**0.629**
	100							
		AD(Nopt)	**0.087**(0)	0.617(0)	1.353(0)	0.137(0)	**0.518**(0)	**1.119**(0)
		CPU(s)	0.404	0.495	0.521	**0.065**	**0.137**	**0.242**
		BestHV	(OS_4, PO_3)	(OS_2, PO_5)	(OS_2, PO_3)	(OS_4, PO_5')	(OS_2, PO_7')	(OS_2, PO_7')
		BestDev	**0.058**	0.487	1.216	0.120	**0.482**	**1.071**
	200							
		AD(Nopt)	**0.060**(2)	0.561(0)	1.314(0)	0.106(0)	**0.428**(0)	**0.971**(0)
		CPU(s)	2.988	3.996	4.262	**0.384**	**0.814**	**1.620**
		BestHV	(OS_6, PO_3)	(OS_2, PO_5)	(OS_2, PO_3)	(OS_4, PO_7')	(OS_4, PO_2')	(OS_2, PO_2')

(Continued)

Table 3.8.9 . *(Continued)*

		BestDev	**0.043**	0.442	1.167	0.094	**0.406**	**0.942**
10	10							
		AD(Nopt)	**0.237**(**5**)	**0.166**(**3**)	0.082(2)	0.252(3)	0.167(2)	**0.071**(**3**)
		CPU(s)	0.001	0.001	0.001	0.000	0.001	0.001
		BestHV	(OS_2, PO_3)	(OS_2, PO_5)	(OS_7, PO_7)	$(OS_2, PO_8^{'})$	$(OS_2, PO_5^{'})$	$(OS_2, PO_4^{'})$
		BestDev	**0.173**	**0.102**	0.045	0.206	0.116	**0.044**
	20							
		AD(Nopt)	**0.514**(0)	0.450(0)	0.216(0)	0.541(0)	**0.442**(0)	**0.196**(0)
		CPU(s)	0.013	0.015	0.015	**0.003**	**0.007**	**0.010**
		BestHV	(OS_4, PO_3)	(OS_2, PO_3)	(OS_7, PO_3)	$(OS_2, PO_2^{'})$	$(OS_2, PO_5^{'})$	$(OS_7, PO_5^{'})$
		BestDev	**0.418**	**0.365**	0.156	0.483	0.383	**0.155**
	50							
		AD(Nopt)	0.718(0)	1.323(0)	0.756(0)	0.718(0)	**1.195**(0)	**0.615**(0)
		CPU(s)	0.214	0.239	0.239	**0.047**	**0.082**	**0.119**
		BestHV	(OS_4, PO_5)	(OS_2, PO_3)	(OS_7, PO_3)	$(OS_2, PO_5^{'})$	$(OS_2, PO_5^{'})$	$(OS_7, PO_5^{'})$
		BestDev	**0.587**	1.188	0.659	0.650	**1.104**	**0.570**
	100							
		AD(Nopt)	0.695(0)	1.936(0)	1.579(0)	**0.625**(0)	**1.540**(0)	**1.185**(0)
		CPU(s)	1.689	1.954	1.970	**0.264**	**0.525**	**0.785**
		BestHV	(OS_6, PO_1)	(OS_2, PO_3)	(OS_7, PO_7)	$(OS_2, PO_4^{'})$	$(OS_3, PO_4^{'})$	$(OS_7, PO_5^{'})$
		BestDev	0.575	1.799	1.430	**0.563**	**1.457**	**1.133**
	200							
		AD(Nopt)	0.697(0)	1.981(0)	3.010(0)	**0.540**(0)	**1.363**(0)	**2.107**(0)
		CPU(s)	14.032	15.766	16.118	**1.472**	**3.289**	**5.588**
		BestHV	(OS_4, PO_3)	(OS_7, PO_7)	(OS_7, PO_7)	$(OS_2, PO_2^{'})$	$(OS_3, PO_4^{'})$	$(OS_3, PO_5^{'})$
		BestDev	0.581	1.874	2.780	**0.496**	**1.304**	**2.061**
20	10							
		AD(Nopt)	**0.196**(**5**)	**0.138**(1)	0.071(**7**)	0.201(4)	0.141(**2**)	**0.057**(6)
		CPU(s)	0.005	0.007	0.007	**0.001**	**0.002**	**0.003**
		BestHV	(OS_2, PO_3)	(OS_2, PO_3)	(OS_7, PO_3)	$(OS_2, PO_6^{'})$	$(OS_7, PO_1^{'})$	$(OS_7, PO_6^{'})$
		BestDev	**0.146**	**0.097**	0.038	0.167	0.109	**0.035**
	20							
		AD(Nopt)	**0.393**(0)	0.369(0)	0.177(0)	0.414(0)	**0.362**(0)	**0.156**(0)
		CPU(s)	0.051	0.055	0.056	**0.018**	**0.013**	**0.021**
		BestHV	(OS_2, PO_3)	(OS_7, PO_7)	(OS_7, PO_3)	$(OS_2, PO_7^{'})$	$(OS_7, PO_1^{'})$	$(OS_7, PO_1^{'})$
		BestDev	**0.320**	**0.298**	0.127	0.364	0.313	**0.121**
	50							
		AD(Nopt)	1.179(0)	1.106(0)	0.641(0)	**1.124**(0)	**0.995**(0)	**0.543**(0)
		CPU(s)	0.877	0.906	0.916	**0.186**	**0.300**	**0.469**
		BestHV	(OS_7, PO_7)	(OS_7, PO_7)	(OS_7, PO_7)	$(OS_2, PO_5^{'})$	$(OS_7, PO_5^{'})$	$(OS_7, PO_4^{'})$
		BestDev	**1.049**	1.027	0.587	1.056	**0.931**	**0.502**
	100							
		AD(Nopt)	1.645(0)	2.258(0)	1.385(0)	**1.487**(0)	**1.870**(0)	**1.037**(0)
		CPU(s)	7.165	7.515	7.641	**1.052**	**1.956**	**2.934**
		BestHV	(OS_2, PO_4)	(OS_7, PO_7)	(OS_7, PO_3)	$(OS_2, PO_2^{'})$	$(OS_3, PO_5^{'})$	$(OS_7, PO_5^{'})$
		BestDev	1.530	2.170	1.306	**1.417**	**1.783**	**1.004**

(Continued)

Table 3.8.9 . *(Continued)*

200							
	AD(Nopt)	1.666(0)	3.896(0)	2.783(0)	**1.412**(0)	**2.900**(0)	**1.911**(0)
	CPU(s)	59.229	62.788	64.373	**6.528**	**12.979**	**19.626**
	BestHV	(OS_6, PO_8)	(OS_7, PO_7)	(OS_7, PO_3)	$(OS_3, PO_4^{'})$	$(OS_3, PO_4^{'})$	$(OS_7, PO_5^{'})$
	BestDev	1.585	3.782	2.683	**1.321**	**2.817**	**1.878**

3.8.2.3 Performance of the heuristic approaches on large instances

We study in this section the performance of the heuristic approaches H1mFSCn and H2mFSCn. In what follows, we compare between the two heuristic approaches, a comparison among all 64 heuristic variants of each heuristic approach is available in the technical report [153]. Since the optimal solutions of the instances are not known, the error calculation is based on the deviation of the solution H_{sol} of a heuristic variant from the best lower bound LB, which is given by $(H_{sol} - LB)/LB$. The performance criteria used are:

- AD: average deviation from the best lower bound of a given heuristic variant.
- Nopt: number of times a given heuristic variant coincides with the best lower bound i.e., with the optimal solution.
- BestHV: the heuristic variant that yields to the best AD.
- BestDev: average of the best deviation found by the 64 heuristic variants of a given heuristic approach for each instance of $S(n, m, p)$.
- CPU: average CPU times in seconds of a given heuristic variant.

For each set $S(n, m, p)$, we run all the 64 heuristic variants of a given heuristic approach and we select the heuristic variant that yields to the best AD (because for Nopt most of the results are null). Then, we report in Table 3.8.9 the AD, Nopt and CPU obtained by the selected heuristic variant, which is given in BestHV. Whereas, BestDev is computed over the best heuristic variant on each instance, i.e., for each instance of $S(n, m, p)$, we keep the best deviation over all the deviations given by the 64 heuristic variants, and we average these best deviations.

By looking at the results of the two heuristic approaches given in Table 3.8.9 and Figures 3.8.5, 3.8.6 and 3.8.7, H2mFSCn appears to be more efficient than H1mFSCn especially for large size instances and for high densities. However, H1mFSCn outperforms H2mFSCn for small instances and relatively low densities (according to all the criteria except the CPU time). Indeed, according to AD and for $p = 0.2$, H1mFSCn outperforms H2mFSCn in 60% of the sets and this is for ($m = 5$, all the values of n), ($m = 10, n \leq 20$) and ($m = 20, n \leq 20$). For $p = 0.5$, H2mFSCn outperforms H1mFSCn in 73.33% of the

sets and this is for $(m = 5, n \geq 50)$, $(m = 10, n \geq 20)$ and $(m = 20, n \geq 20)$. For $p = 0.8$, H2mFSCn outperforms H1mFSCn for all the values of n and m. These results are due to the fact that for small instances and for low densities, it is more important to compute as much partial sequences as possible, but when the number of conflicting jobs increases, especially for large instances, it is more effective to gather non conflicting jobs (independent sets) in the sequence as to minimize the number of idle times in the final schedule.

When considering the CPU time, H2mFSCn requires on average less CPU time than H1mFSCn for all the sets $S(n, m, p)$. This can be explained by the fact that the number of partial sequences computed by H1mFSCn is greater than the number of partial sequences generated by H2mFSCn. It should be noticed that the density of the conflict graph and the CPU time are directly proportional, but the CPU time required is relatively short since less than 20 seconds are needed to solve large instances with high densities.

The best results found by H1mFSCn and H2mFSCn can be summarized as follows: the average deviation from the best lower bound is about 0.469 for $p = 0.2$, 0.818 for $p = 0.5$ and 0.693 for $p = 0.8$. As it is observed, the deviation is quite large (more than 40%). To check if this is a lack of performance from the lower bounds or from the heuristic approaches, a third simple heuristic has been implemented. In this heuristic, we start from a priority list of the jobs, we schedule every operation in that order, each one as soon as possible subject to the conflict constraints. This has been repeated with many random priority lists, up to an equivalent CPU usage than the heuristic variant that produces the best deviation. We observed from the implementation that the proposed heuristic approaches outperform this heuristic for all the instance classes. The average deviation from the best lower bound of this heuristic is about 0.725 for $p = 0.2$, 1.310 for $p = 0.5$ and 1.378 for $p = 0.8$, whereas Nopt=0 for all the sets $S(n, m, p)$.

3.9 Conclusion

In summary, we addressed the flow shop scheduling problem with conflict graph under minimization of the makespan. We started with a theoretical analysis of the problem in which we discussed the effects of imposing a conflict graph on the flow shop problem. Since conflicts may arise from shared resources, we first studied the relation between our problem and the resource-constrained scheduling problem. Indeed, we proved that the FSC problem is polynomially equivalent to $F|res^t.11|C_{max}$. From this result, we established that the two-machine FSC problem is NP-hard in the strong sense even when restricted to a complement of a complete split graph. We also showed that, for complements of complete bipartite graphs, the problem remains NP-hard. This later result holds even for the preemptive version. Then, we shifted to the case of unit-time operations. We found that this case is also NP-hard since it is polynomially equivalent to the MPC problem, which is a celebrated NP-hard problem. Unfortunately, only some positive results were found. An $O(n^{2.5})$ algorithm for the problem $F2|ConfG = (V, E), p_j = 1|C_{max}$ was given and graph classes on which the problem $F2|ConfG = (V, E), p_{ij} = 1|C_{max}$ is solvable in polyno-

mial time were presented. On the other hand, we showed that the permutation schedules are dominant for this later problem. However, for the case of arbitrary processing times the permutation schedules are no longer dominant even for two machines. Furthermore, new complexity results of the resource-constrained scheduling problem were established. Tables 3.9.1 and 3.9.2 summarize the results we obtained on the complexity of the FSC and the resource-constrained scheduling problems respectively.

Table 3.9.1: Complexity results of the FSC problem.

FSC Problem	Complexity	Reference
$m = 2$, G a complement of a complete split graph	sNP-Hard	consequence of Proposition 3.1
$m = 2$, G a complement of a complete bipartite graph	NP-Hard	Theorem 3.2
$m = 2$, $p_{ij} = 1$, $G = (V, E)$	sNP-Hard	Theorem 3.6
$m = 2$, $p_j = 1$, $G = (V, E)$	$O(n^{2.5})$	Theorem 3.9
$m = 2$, *pmtn*, G a complement of a complete bipartite graph	NP-Hard	Theorem 3.3

Table 3.9.2: New complexity results of the flow shop problem under resource constraints.

Problem	Complexity	Reference
$F2\|pmtn, res^t.11\|C_{max}$	sNP-Hard	Corollary 3.5
$F2\|res^t.11, p_{ij} = 1\|C_{max}$	sNP-Hard	Corollary 3.7

In the second part of this chapter, a variety of solution methods were presented. For the case of unit-time operations, we proposed three MILP models (P_3), (P_4) and (P_5) and a branch and bound algorithm. We included in this algorithm new lower bounds, upper bounds and branching strategies. The computational study clearly showed that the branch and bound algorithm is better than the MILP models in most of the cases and can solve instances of size up to 20000 jobs in an acceptable amount of time. However, the best MILP model (P_4) can solve instances of size up to 450 jobs. An essential part of this algorithm is the use of an appropriate branching strategy that dramatically accelerate the process. For the case of arbitrary processing times, we discussed two MILP models (P_2) and (P_1) for the FSC and the permutation FSC problems respectively. Furthermore, 128 heuristic variants were designed. The solutions they produced were compared to the optimal solutions obtained by the MILP models for small instances and to the proposed lower bounds for larger instances.

A wise man will make more opportunities than he finds.

Francis Bacon

Conclusions and future directions

In the following paragraphs, we provide a summary of the results of this thesis and the conclusions we can draw from them, then we discuss future directions.

We studied in this thesis the problem of scheduling on dedicated machines subject to conflict constraints represented by a conflict graph G, such that jobs represented by adjacent vertices in G cannot be processed simultaneously on different machines. We considered two types of machine environments: open shop and flow shop.

We began our theoretical study of the two problems by analysing the relation between scheduling with conflict graph and the resource-constrained scheduling problem. This study was motivated by the conflicts that may arise from shared resources. In general, $O|res^t \ldots|C_{max}$ and $F|res^t \ldots|C_{max}$ can be modelled by a conflict hypergraph. However, in special cases a conflict graph suffices. In this thesis, we showed that the OSC (resp. FSC) problem is polynomially equivalent to $O|res^t.11|C_{max}$ (resp. $F|res^t.11|C_{max}$).

The open shop problem with conflict graph was the subject of Chapter 2. We started by considering the difficulty of the problem. For the two-machine case, we showed that the problem is strongly NP-hard even when the conflict graph is a complement of a bipartite graph and the operations of each job have the same processing time with $p_{ij} \in \{1, 2, 3\}$. However, when $p_{ij} \in \{0, 1, 2\}$ the problem can be solved in $O(n^{2.5})$-time for an arbitrary conflict graph. For the three-machine case, we found that when all the operations have one unit of processing time the problem is NP-hard in the strong sense, but when the conflict graph is a complement of a triangle free graph the problem is solvable in $O(n^{2.5})$-time. We also considered the preemptive version and showed that the two-machine case can be solved in $O(n^3)$-time for arbitrary processing times and conflict graph. From these complexity results and the polynomial equivalence between OSC and $O|res^t.11|C_{max}$, new complexity results of the later problem have been derived including NP-hardness results and well solvable cases (see Table 2.6.2).

Since the OSC problem is NP-hard in general, we proposed a two-phase heuristic approach for the problem $Om|ConfG = (V, E)|C_{max}$. In the first phase, the set of operations of all the jobs is partitioned into a minimum number of subsets, called schedule slices, such that the operations of each slice can be processed simultaneously. The goal of this phase is to construct schedule slices as balanced as possible.

This is done by computing min-max matchings in weighted bipartite graphs constructed from the conflict graph. In the second phase, the resulting schedule slices are inserted one by one to make a complete schedule by using a beam search technique. We then conducted computational experiments to measure the performance of the 24 heuristic variants. These variants are derived from the heuristic approach by varying the beam search technique and the selecting rule of the schedule slices. The experiments were carried out on 3600 instances, which are constructed by considering the benchmark instances of the open shop problem given by Taillard [144] and for each instance a set of conflict graphs are generated using the $G(n, p)$ Erdős Rényi method. Since the optimal solutions of these instances are unknown, we proposed three different lower bounds on the makespan to study the quality of the solutions obtained by the heuristic variants. The overall results were promising: the average deviation from the best lower bound is about 1.347% for instances with high conflicts, 3.93% for instances with medium conflicts and 4.706% for instances with low conflicts between the jobs.

Our treatment of the flow shop problem with conflict graph, which was the subject of Chapter 3, is similar to that of the open shop problem. The FSC problem seems to be much more difficult than the OSC problem because most of the FSC problems have been shown to be NP-hard even for two machines, and only some cases are polynomial. For the two-machine case, we showed that the FSC problem is NP-hard when restricted to split graphs and complements of complete bipartite graphs. We have also found that even if the preemption is allowed the problem remains NP-hard. Then we restricted ourselves to the case of unit-time operations in an effort to find polynomial cases. Unfortunately, the problem remains NP-hard for arbitrary conflict graphs. However, this problem is solvable in polynomial time when restricted to special graph classes including trees, block graphs, interval graphs, cographs, bipartite distance-hereditary graphs, bipartite permutation graphs, cocomparability graphs and distance-hereditary graphs. Also, when all the jobs have the same processing time and they comprise a unique operation, the two-machine FSC problem can be solved in $O(n^{2.5})$-time. Similarly to Chapter 2, new complexity results of the flow shop problem under resource constraints have been derived, these results are summarised in Table 3.9.2.

As mentioned before, most research on flow shop scheduling problems has focused on permutation schedules due to the relative combinatorial simplicity of schedules that can be specified simply by giving a permutation of the jobs. When the conflict constraints are taken into account, we found that the permutation schedules are no longer dominant even for two machines and an example in which the optimal schedule is not a permutation schedule was given in Figure 3.2.6. Fortunately, for the restricted case of unit-time operations, we proved that the permutation schedules are dominant for two machines. This result motivates the study of algorithms to solve the later restricted case. Indeed, we proposed three MILP models and a branch and bound algorithm. The first MILP model is based on an assignment of jobs to various possible positions (at most $2n - 1$ position) and the objective is to minimize the position of the last job in the sequence. The number of positions considered was obtained by the heuristics H_{11} and H_{12}. The second MILP model assigns jobs to various sequence positions and the objective is to minimize the total number of idle times generated. The third MILP model was inspired from the

Liao-You model ([99, 156]) which was proposed for the regular permutation flow shop. This model uses pairs of dichotomous constraints to control the assignment of jobs to sequence positions. The computational experiments we performed showed that the second MILP model is better than the other models and can solve instances of size up to 450 jobs in an acceptable amount of time. The proposed branch and bound algorithm is based on new lower and upper procedures and can solve instances of size up to 20000 jobs. An essential part of this algorithm is the use of an appropriate branching strategy that dramatically accelerate the process.

For the general case of m machines and arbitrary processing times, we proposed two MILP models for the FSC and the permutation FSC problems. These models are based on the Liao-You model ([99, 156]) and can solve only small size instances. Somewhat surprisingly, the makespan of the optimal permutation schedule coincides with the optimal makespan for all the tested instances. Furthermore, we described two different heuristic approaches for the permutation m-machine FSC problem that use the idea of the NEH heuristic. From these approaches, 128 heuristic variants have been derived and an extensive computational experiments were performed on a wide range of test problems to measure the performance of the proposed approaches. The quality of the solutions they produced was compared to the optimal solutions obtained by the MILP models for small instances and to five lower bounds (that we designed for this purpose) for larger instances.

Although several results have been obtained in this thesis, there exist numbers of unexploited areas and some unsolved problems. We provide in the following the ongoing work as well as the future openings that could to investigated.

- In Chapters 2 and 3, we studied the complexity of the OSC and FSC problems for some particular conflict graphs. The extension of these results to other graph classes that may include polynomial cases, e.g. interval, permutation and comparability graphs, is a possible future direction.
- Since preemptions result in an efficient algorithm for the OSC problem on two machines, it would be important to study the complexity of the preemptive version of the OSC problem on m machines with $m \geq 3$.
- Additional work is needed concerning the solution methods. For the FSC problem, we proposed a branch and bound algorithm when restricted to two machines and unit-time operations. An important future direction would be to propose a branch and bound algorithm for the two-machine case with arbitrary processing times in which the permutation schedules are no longer dominant. It is also of interest to propose exact methods for the OSC problem, including mathematical models and branch and bound algorithms and to design metaheuristics for both OSC and FSC problems.
- The MILP model we presented in Chapter 3 for the general m-machine FSC problem can solve only small size instances. It is in our ongoing work to enhance the current model by trying to add

cuts and inequalities on one hand and by the interoperation of (meta)heuristics and the mathematical model on the other hand. This later technique yields to the so-called matheuristics.

- Another important future direction is to study the online version of scheduling with conflict graph on dedicated machines.
- This thesis has focused on the open shop and flow shop problems, a promising path would be to study the problem of scheduling with conflict graph in a job shop environment.
- It could be interesting also to study the problem of scheduling with conflict graph for other objective functions.

References

[1] J. O. Achugbue and F. Y. Chin. Scheduling the open shop to minimize mean flow time. *SIAM Journal on Computing*, 11:709–720, 1982.

[2] N. Alon. A note on the decomposition of graphs into isomorphic matchings. *Acta Mathematica Hungarica*, 42:221–223, 1983.

[3] D. Applegate and W. Cook. A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3:149–156, 1991.

[4] K. Asdre and S. D. Nikolopoulos. The 1-fixed-endpoint path cover problem is polynomial on interval graphs. *Algorithmica*, 58:679–710, 2010.

[5] I. Averbakh, O. Berman, and I. Chernykh. The *m*-machine flowshop problem with unit-time operations and intree precedence constraints. *Operations Research Letters*, 33(3):263–266, 2005.

[6] B. S. Baker and E. G. Coffman. Mutual exclusion scheduling. *Theoretical Computer Science*, 162: 225–243, 1996.

[7] K. R. Baker. *An Introduction to Sequencing and Scheduling*. Wiley, New York, 1974.

[8] Bamboo. edited by experian-prologia sas, marseille, france. `http://www.experian-prologia.fr/solutions/bamboo/`.

[9] P. Baptiste and G. V. Timkovsky. Shortest path to nonpreemptive schedules of unit-time jobs on two identical parallel machines with minimum total completion time. *Mathematical Methods of Operations Research*, 60:145–153, 2004.

[10] M. Bendraouche. *Ordonnancement sur machines identiques avec graphe de concordance*. PhD thesis, University of Science and Technology Houari Boumediene, Algiers, Algeria, 2011.

[11] M. Bendraouche and M. Boudhar. Scheduling jobs on identical machines with agreement graph. *Computers and Operations Research*, 39:382–390, 2012.

[12] M. Bendraouche and M. Boudhar. Scheduling with agreements: new results. *International Journal of Production Research*, 54:3508–3522, 2016.

[13] M. Bendraouche, M. Boudhar, and A. Oulamara. Scheduling: Agreement graph vs resource constraints. *European Journal of Operational Research*, 240:355–360, 2015.

[14] P. Bjorstad, W.M. Coughran, and E. Grosse. Parallel domain decomposition applied to coupled transport equations. In D.E. Keys and J. Xu, editors, *Proceedings of the Domain Decomposition Methods in Scientific and Engineering Computing, AMS*, page 369–380, Providence, 1995.

[15] J. Błażewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan. Scheduling subject to resource constraints: classification and complexity. *Discrete Applied Mathematics*, 5:11–24, 1983.

[16] J. Błażewicz, W. Cellary, R. Slowinski, and J. Weglarz. Scheduling under resource constraints-deterministic models. *Annals of Operations Research*, 7, 1986.

[17] J. Błażewicz, W. Kubiak, and J. Szwarcfiter. Scheduling unit-time tasks on flow-shops under resource constraints. *Annals of Operations Research*, 16(1):255–266, 1988.

[18] J. Błazewicz, K.H. Ecker, E. Pesch, G. Schmidt, and J. Weglarz. *Handbook on Scheduling: From Theory to Applications*. Springer-Verlag Berlin Heidelberg, 2007.

[19] C. Blum. Beam-aco-hybridizing ant colony optimization with beam search: an application to open shop scheduling. *Computers & Operations Research*, 32:1565–1591, 2005.

[20] C. Blum and M. Sampels. An ant colony optimization algorithm for shop scheduling problems. *Journal of Mathematical Modelling and Algorithms*, 3:285–308, 2004.

[21] H. L. Bodlaender and F. V. Fomin. Equitable colorings of bounded treewidth graphs. In J. Fiala, V. Koubek, and J. Kratochvíl, editors, *Proceedings of the 29th International Symposium of Mathematical Foundations of Computer Science MFCS'2004*, pages 180–190, Prague, Czech Republic, August 22–27 2004. Springer Berlin Heidelberg.

[22] H. L. Bodlaender and K. Jansen. Restrictions of graph partition problems. part i. *Theoretical Computer Science*, 148:93–109, 1995.

[23] H. L. Bodlaender, K. Jansen, and G. J. Woeginger. Scheduling with incompatible jobs. *Discrete Applied Mathematics*, 55:219–232, 1994.

[24] F. T. Boesch and J. F. Gimpel. Covering points of a digraph with point-disjoint paths and its application to code optimization. *Journal of the ACM*, 24:192–198, 1977.

[25] F. T. Boesch, S. Chen, and J. A. M. McHugh. On covering the points of a graph with point disjoint paths. In *Graph Theory and Combinatorics*, Vol. 406 of Lecture Notes in Mathematics, pages 201–212. Springer Berlin Heidelberg, 1974.

[26] J. A. Bondy and U. S. R. Murty. *Graph Theory with Applications*. American Elsevier Publishing Co. Inc., New York, 1976.

[27] H. Bräsel, T. Tautenhahn, and F. M. Werner. Constructive heuristic algorithms for the open shop problem. *Computing*, 51:95–110, 1993.

[28] P. Brucker, B. Jurisch, and M. Jurisch. Open shop problems with unit time operations. *Zeitschrift für Operations Research*, 37:59–73, 1993.

[29] P. Brucker, J. Hurink, B. Jurisch, and B. Wöstmann. A branch and bound algorithm for the open-shop problem. *Discrete Applied Mathematics*, 76:43–59, 1997.

[30] F. Burns and J. Rooker. Technical note-johnson's three-machine flow shop conjecture. *Operations Research*, 24(3):578–580, 1976.

[31] F. Burns and J. Rooker. Technical note-three stage flow shops with recessive second stage. *Operations Research*, 26(1):207–208, 1978.

[32] H. G. Campbell, R. A. Dudek, and M. L. Smith. A heuristic algorithm for the n job, m machine sequencing problem. *Management Science*, 16:B630–B637, 1970.

[33] J. Carlier and E. Pinson. Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.

[34] J. Carlier and I. Rebaï. Two branch and bound algorithms for the permutation flow shop problem. *European Journal of Operational Research*, 90:238–251, 1996.

[35] Y. Cho and S. Sahni. Preemptive scheduling of independent jobs with release and due times on open, flow and job shops. *Operations Research*, 29:511–522, 1981.

[36] Y. Cho and S. Sahni. Preemptive scheduling of independent jobs with release and due times on open, flow and job shops. *Operations Research*, 29:511–522, 1981.

[37] E. G. Coffman and R. L. Graham. Optimal scheduling for two-processor systems. *Acta Informatica*, 1:200–213, 1972.

[38] E. Cohen and M. Tarsi. Np-completeness of graph decomposition problems. *Journal of Complexity*, 7:200–212, 1991.

[39] S. Colak and A. Agarwal. Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem. *Naval Research Logistics (NRL)*, 52:631–644, 2005.

[40] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J. M. Vincent, and F. Wagner. Random graph generation for scheduling simulations. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, number 60 in SIMUTools '10, pages 1–10, Torremolinos, Malaga, Spain, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[41] D. G. Corneil, B. Dalton, and M. Habib. Ldfs-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM Journal on Computing*, 42:792–807, 2013.

[42] D.G. Corneil. The complexity of generalized clique packing. *Discrete Applied Mathematics*, 12 (3):233–239, 1985.

[43] A. Custic, B. Klinz, and G. J. Woeginger. Geometric versions of the three-dimensional assignment problem under general norms. *Discrete Optimization*, 18:38–55, 2015.

[44] E. Dahlhaus and M. Karpinski. Matching and multidimensional matching in chordal and strongly chordal graphs. *Discrete Applied Mathematics*, 84(1):79–91, 1998.

[45] D. G. Dannenbring. An evaluation of flow shop sequencing heuristics. *Management Science*, 23: 1174–1182, 1977.

[46] D. de Werra. Restricted coloring models for timetabling. *Discrete Mathematics*, 165:161–170, 1997.

[47] D. de Werra and J. Blazewicz. Some preemptive open shop scheduling problems with a renewable or a nonrenewable resource. *Discrete Applied Mathematics*, 35:205–219, 1992.

[48] D. de Werra and J. Blazewicz. Addendum: some preemptive open shop scheduling problems with a renewable or a non renewable resource. *Discrete Applied Mathematics*, 43:103–104, 1993.

[49] D. de Werra, J. Blazewicz, and W. Kubiak. A preemptive open shop scheduling problem with one resource. *Operations Research Letters*, 10:9–15, 1991.

[50] U. Dorndorf, E. Pesch, and T. Phan-Huy. Solving the open shop scheduling problem. *Journal of Scheduling*, 4:157–174, 2001.

[51] H. Emmons and G. Vairaktarakis. *Flow Shop Scheduling: Theoretical Results, Algorithms, and Applications*. Springer US, Boston, MA, 2013.

[52] P. Erdős and A. Rényi. On random graphs **1**. *Publicationes Mathematicae*, 6:290–297, 1959.

[53] G. Even, M. M. Halldórsson, L. Kaplan, and D. Ron. Scheduling with conflicts: online and offline algorithms. *Journal of Scheduling*, 12:199–224, 2009.

[54] T. Fiala. An algorithm for the open-shop problem. *Mathematics of Operations Research*, 8:100–109, 1983.

[55] L. R. Ford and D. R. Fulkerson. Maximal flow through a network. *Canadian Journal of Mathematics*, 8:399–404, 1956.

[56] J. M. Framinan, J. N. D. Gupta, and R. Leisten. A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *The Journal of the Operational Research Society*, 55:1243–1255, 2004.

[57] D. S. Franzblau and A. Raychaudhuri. Optimal hamiltonian completions and path covers for trees, and a reduction to maximum flow. *ANZIAM Journal*, 44:193–204, 2002.

[58] F. Gardi. Mutual exclusion scheduling with interval graphs or related classes, part **1**. *Discrete Applied Mathematics*, 157:19–35, 2009.

[59] M. R. Garey and R. L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4, 1975.

[60] M. R. Garey and D. S. Johnson. *Computers and intractability: A Guide to the Theory of NP-Completeness*. New York: Freeman, 1979.

[61] M. R. Garey, D. S. Johnson, and R. Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1:117–129, 1976.

[62] M. R. Garey, D. S. Johnson, R. E. Tarjan, and M. Yannakakis. Scheduling opposing forests. *SIAM Journal on Algebraic Discrete Methods*, 4:72–93, 1983.

[63] P.C. Gilmore and R.E. Gomory. Sequencing a one-state variable machine: a solvable case of the traveling salesman problem. *Operations Research*, 12:655–679, 1964.

[64] M. C. Golumbic. *Algorithmic Graph Theory and Perfect Graphs*. Academic Press, NewYork, 1980.

[65] M. Gondran and M. Minoux. *Graphs and algorithms*. Wiley/ Interscience, New York, 1984.

[66] T. Gonzalez. A note on open shop preemptive schedules. *IEEE Transactions on Computers*, 28: 782–786, 1979.

[67] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the Association for Computing Machinery*, 23:665–679, 1976.

[68] T. Gonzalez and S. Sahni. Flowshop and jobshop schedules: Complexity and approximation. *Operations Research*, 26:36–52, 1978.

[69] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of discrete mathematics*, 5: 287–326, 1979.

[70] C. Guéret and C. Prins. Classical and new heuristics for the open-shop problem: A computational evaluation. *European Journal of Operational Research*, 107:306–314, 1998.

[71] C. Guéret and C. Prins. Forbidden intervals for open-shop problems. Research report 98/10/auto, Ecole de Mines de Nantes, Nantes, 1998.

[72] C. Guéret, N. Jussien, and C. Prins. Using intelligent backtracking to improve branch-and-bound methods: An application to open-shop problems. *European Journal of Operational Research*, 127: 344–354, 2000.

[73] J. N. D. Gupta. A functional heuristic algorithm for the flowshop scheduling problem. *Operational Research Quarterly*, 22:39–47, 1971.

[74] J. N.D. Gupta, C. Koulamas, and G. J. Kyparisis. Performance guarantees for flowshop heuristics to minimize makespan. *European Journal of Operational Research*, 169:865–872, 2006.

[75] M. M. Halldórsson, G. Kortsarz, A. Proskurowski, R. Salman, H. Shachnai, and J. A. Jan Arne Telle. Multicoloring trees. *Information and Computation*, 180:113–129, 2003.

[76] P. Hansen, A. Hertz, and J. Kuplinsky. Bounded vertex colorings of graphs. *Discrete Mathematics*, 111:305–312, 1993.

[77] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[78] T. C. Hu. Parallel sequencing and assembly line problems. *Operations Research*, 9:841–848, 1961.

[79] T. S. Hundal and J. Rajgopal. An extension of palmer's heuristic for the flow shop scheduling problem. *International Journal of Production Research*, 26:1119–1124, 1988.

[80] R. W. Hung and M. S. Chang. Solving the path cover problem on circular-arc graphs by using an approximation algorithm. *Discrete Applied Mathematics*, 154:76–105, 2006.

[81] S. Ignall and L. Schrage. Application of the branch and bound technique to some flow-shop scheduling problems. *Operations Research*, 13(3):400–412, 1965.

[82] S. Irani and V. Leung. Scheduling with conflicts, and applications to traffic signal control. In *Proceedings of the 7th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 85–94, Atlanta, GA, January 28–30 1996.

[83] S. Irani and V. Leung. Scheduling with conflicts on bipartite and interval graphs. *Journal of Scheduling*, 6(3):287–307, 2003.

[84] H. Ishibuchi, S. Misaki, and H. Tanaka. Modified simulated annealing algorithms for the flow shop sequencing problem. *European Journal of Operational Research*, 81:388–398, 1995.

[85] J. R. Jackson. An extension of johnson's results on job lot scheduling. *Naval Research Logistics quarterly*, 3:201–203, 1956.

[86] K. Jansen. The mutual exclusion scheduling problem for permutation and comparability graphs. *Information and Computation*, 180:71–81, 2003.

[87] M. Jarvis and B. Zhou. Bounded vertex coloring of trees. *Discrete Mathematics*, 232:145–151, 2001.

[88] S. M. Johnson. Optimal two and three stage production schedules with set up times included. *Naval Research Logistics quarterly*, 1:61–68, 1954.

[89] V. Jost. *Ordonnancement chromatique: Polyhédres, complexité et classification*. PhD thesis, Université Joseph Fourier-Grenoble I, Grenoble, France, 2006.

[90] B. Jurisch and W. Kubiak. Two-machine open shops with renewable resources. *Operations Research*, 45:544–552, 1997.

[91] O. Kariv and S. Even. An $o(n^{2.5})$ algorithm for maximum matching in general graphs. In *Proceedings of the 16th IEEE Annual Symposium on Foundations of Computer Science*, pages 100––112, 1975.

[92] R. M. Karp. Reducibilty among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Proceedings of a symposium on the Complexity of Computer Computations*, pages 85––103. Plenum Press, NewYork, NY, 1972.

[93] F. Kitagawa and H. Ikeda. An existential problem of a weight-controlled subset and its application to school timetable construction. *Discrete Mathematics*, 72:195–211, 1988.

[94] T. Ladhari and M. Haouari. A computational study of the permutation flow shop problem based on a tight lower bound. *Computers and Operations Research*, 32(7):1831–1847, 2005.

[95] B. J. Lageweg, J. K. Lenstra, and A. H. G. Rinnooy Kan. A general bounding scheme for the permutation flow-shop problem. *Operations Research*, 26:53–67, 1978.

[96] E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Minimizing maximum lateness in a two-machine open shop. *Mathematics of Operations Research*, 6:153–158, 1981.

[97] E. L. Lawler, J. K. Lenstra, A. H.G. Rinnooy Kan, and D. B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In *Logistics of Production and Inventory*, volume 4 of *Handbooks in Operations Research and Management Science*, pages 445–522. 1993.

[98] J. K. Lenstra, R. H. G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[99] C.L. Liao and C.T. You. An improved formulation for the job-shop scheduling problem. *Journal of the Operational Research Society*, 43:1047–1054, 1992.

[100] C. F. Liaw. An iterative improvement approach for the nonpreemptive open shop scheduling problem. *European Journal of Operational Research*, 111:509–517, 1998.

[101] C. F. Liaw. A tabu search algorithm for the open shop scheduling problem. *Computers & Operations Research*, 26:109–126, 1999.

[102] C. F. Liaw. Applying simulated annealing to the open shop scheduling problem. *IIE Transactions*, 31:457–465, 1999.

[103] C. F. Liaw. A hybrid genetic algorithm for the open shop scheduling problem. *European Journal of Operational Research*, 124:28–42, 2000.

[104] Z. A. Lomnicki. A branch-and-bound algorithm for the exact solution of the three-machine scheduling problem. *OR*, 16:89–100, 1965.

[105] Z. Lonc. On complexity of some chain and antichain partition problems. In *Proceedings of the 17th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG'91, pages 97–104, London, UK, 1992. Springer-Verlag.

[106] C. Magnant and D.M. Martin. A note on the path cover number of regular graphs. *The Australasian Journal of Combinatorics*, 43:211––217, 2009.

[107] A. S. Manne. On the job-shop scheduling problem. *Operations Research*, 8(2):219–223, 1960.

[108] P. Martin and D. B. Shmoys. A new approach to computing optimal schedules for the job-shop scheduling problem. In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, *Proceedings of the 5th International Integer Programming and Combinatorial Optimization Conference*, pages 389–403, Vancouver, British Columbia, Canada, June 3–5 1996. Springer Berlin Heidelberg.

[109] R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6:1–12, 1959.

[110] C. L. Monma. The two-machine maximum flow time problem with series-parallel precedence constraints: An algorithm and extensions. *Operations Research*, 27:792–798, 1979.

[111] C. L. Monma. Sequencing to minimize the maximum job cost. *Operations Research*, 28:942–951, 1980.

[112] C. L. Monma and J. B. Sidney. Sequencing with series-parallel precedence constraints. *Mathematics of Operations Research*, 4:215–224, 1979.

[113] S. Moran and Y. Wolfstahl. Optimal covering of cacti by vertex-disjoint paths. *Theoretical Computer Science*, 84:179–197, 1991.

[114] R. Motwani, S. Phillips, and E. Torng. Nonclairvoyant scheduling. *Theoretical Computer Science*, 130:17–47, 1994.

[115] B. Naderi, S. M. T. Fatemi Ghomi, M. Aminnayeri, and M. Zandieh. A contribution and new heuristics for open shop scheduling. *Computers & Operations Research*, 37:213–221, 2010.

[116] M. Nawaz, E. E. Enscore, and I. Ham. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *OMEGA, The Internalional Journal of Management Science*, 11:91–95, 1983.

[117] S. Noorvash. Covering the vertices of a graph by vertex-disjoint paths. *Pacific Journal of Mathematics*, 58:159–168, 1975.

[118] E. Nowicki and C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91:160–175, 1996.

[119] O. Ore. Arc coverings of graphs. *Annali di Matematica Pura ed Applicata*, 55:315–322, 1961.

[120] I. H. Osman and C. N. Potts. Simulated annealing for permutation flow-shop scheduling. *OMEGA, The Internalional Journal of Management Science*, 17:551–557, 1989.

[121] A. Oulamara, D. Rebaine, and M. Serairi. Scheduling the two-machine open shop problem under resource constraints for setting the jobs. *Annals of Operations Research*, 211:333–356, 2013.

[122] E. S. Page. An approach to the scheduling of jobs on machines. *Journal of the Royal Statistical Society. Series B (Methodological)*, 23:484–492, 1961.

[123] S. D. Palmer. Sequencing jobs through a multi-stage process in the minimum total time: A quick method of obtaining a near optimum. *Journal of the Operational Research Society*, 16:101–107, 1965.

[124] C. H. Pan. A study of integer programming formulations for scheduling problems. *International Journal of Systems Science*, 28:33–41, 1997.

[125] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1982.

[126] Y. B. Park, C. D. Pegden, and E. E. Enscore. A survey and evaluation of static flowshop scheduling heuristics. *International Journal of Production Research*, 22:127–141, 1984.

[127] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer-Verlag New York, 2012.

[128] S. S. Pinter and Y. Wolfstahl. On mapping processes to processors in distributed systems. *International Journal of Parallel Programming*, 16:1–15, 1987.

[129] C. N. Potts, D. B. Shmoys, and D. P. Williamson. Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, 10:281–284, 1991.

[130] C. Prins. Competitive genetic algorithms for the open-shop scheduling problem. *Mathematical Methods of Operations Research*, 52:389–411, 2000.

[131] A. Ramudhin and P. Marier. The generalized shifting bottleneck procedure. *European Journal of Operational Research*, 93:34–48, 1996.

[132] S. Reza Hejazi and S. Saghafian. Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, 43:2895–2929, 2005.

[133] H. Röck. Scheduling unit task shops with resource constraints and excess usage costs. Technical Report 83–21, Fachbereich Informatik, Technical University of Berlin, Berlin (FRG), 1983.

[134] H. Röck. Some new results in flow shop scheduling. *Zeitschrift für Operations Research*, 28:1–16, 1984.

[135] R. Ruiz and C. Maroto. A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165:479–494, 2005.

[136] S. Sakai, M. Togasaki, and K. Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126:313–322, 2003.

[137] D. Y. Sha and C. Y. Hsu. A new particle swarm optimization for the open shop scheduling problem. *Computers & Operations Research*, 35:3243–3261, 2008.

[138] J. B. Sidney. The two-machine maximum flow time problem with series parallel precedence relations. *Operations Research*, 27:782–791, 1979.

[139] W. E. Smith. Various optimizers for single-stage production. *Naval Research Logistics quarterly*, 3:59–66, 1956.

[140] V.A. Strusevich. Dominating machines in deterministic systems. In *Applications of Mathematical Methods and Computer Engineering for Solving Problems of National Economy*, pages 151–152, Minsk, 1986. Proceedings of Republican Meeting.

[141] H. Süral, S. Kondakci, and N. Erkip. Scheduling unit-time tasks in renewable resource constrained flowshops. *Zeitschrift für Operations Research*, 36:497–516, 1992.

[142] W. Szwarc. Optimal two-machine orderings in the 3 × n flow shop problem. *Operations Research*, 25(1):70–77, 1977.

[143] E. Taillard. Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47:65–74, 1990.

[144] É. Taillard. Benchmarks. `http://mistic.heig-vd.ch/taillard/`, 2005.

[145] V. Tanaev, Y. N. Sotskov, and V. A. Strusevich. *Scheduling Theory. Multi-Stage Systems*. Springer Netherlands, 1994.

[146] T. Tautenhahn and G. J. Woeginger. Unit-time scheduling problems with time dependent resources. *Computing*, 58:97–111, 1997.

[147] N. E. H. Tellache and M. Boudhar. Flow shop scheduling problem with conflict graphs. *revised version submitted to Annals of Operations Research*, .

[148] N. E. H. Tellache and M. Boudhar. Open shop scheduling problems with conflict graphs. *revised version submitted to Discrete Applied Mathematics*, .

[149] N. E. H. Tellache and M. Boudhar. Flow-shop scheduling problem with conflict graphs. In *Proceedings of the 6th Operational Research Practice in Africa Conference (ORPA'2015)*, page 33–34, Algiers, Algeria, April 20-22 2015.

[150] N. E. H. Tellache and M. Boudhar. Open-shop scheduling problems with conflict graphs. In *Proceedings of the 12ème Colloque sur l'Optimisation et les Systèmes d'Information (COSI'2015)*, page 190–201, Oran, Algeria, Jun 01-03 2015.

[151] N. E. H. Tellache and M. Boudhar. Scheduling the two-machine flow shop problem with unit-time operations and conflict graph. In *Proceedings of the Conference-School on Discrete Mathematics and Computer Science (DIMACOS'2015)*, page 81–84, Algeria, Sidi Bel Abbès, November 15–19 2015.

[152] N. E. H. Tellache and M. Boudhar. Two-machine flow shop problem with unit-time operations and conflict graph. *International Journal of Production Research*, 55:1664–1679, 2016.

[153] N. E. H. Tellache and M. Boudhar. Heuristics for flow shop scheduling problem with conflict graphs: computational results. *Les annales RECITS* `http://www.lrecits.usthb.dz/annales.htm`, 3:47–79, 2016.

[154] N. E. H. Tellache and M. Boudhar. The two-machine flow shop problem with conflict graphs. *IFAC-PapersOnLine*, 49:1026–1031, 2016. 8th IFAC Conference on Manufacturing Modelling, Management and Control MIM'2016 Troyes, France, 28-30 June 2016.

[155] N. E. H. Tellache and M. Boudhar. Branch and bound algorithm for the two-machine flow shop problem with unit-time operations and conflict graph. In *Proceedings of the 3ème Journées Scientifiques du Laboratoire RECITS (JSL'2016)*, page 3–3, Algeria, Algiers, April 23–24 2016.

[156] F.T. Tseng, E.F. Stafford, and J.N.D. Gupta. An empirical analysis of integer programming formulations for the permutation flowshop. *OMEGA, The international Journal of Management Science*, 32:285–293, 2004.

[157] S. Turner and D. Booth. Comparison of heuristics for flow shop sequencing. *OMEGA, The Internalional Journal of Management Science*, 15:75–78, 1987.

[158] J. M. M. van Rooij, M. E. van Kooten Niekerk, and H. L. Bodlaender. Partition into triangles on bounded degree graphs. *Theory of Computing Systems*, 52:687–718, 2013.

[159] H. M. Wagner. An integer linear-programming model for machine scheduling. *Naval Research Logistics Quarterly*, 6:131–140, 1959.

[160] F. Werner and A. Winkler. Insertion techniques for the heuristic solution of the job shop problem. *Discrete Applied Mathematics*, 58:191–211, 1995.

[161] J. M. Wilson. Alternative formulations of a flow-shop scheduling problem. *The Journal of the Operational Research Society*, 40:395–399, 1989.

[162] G.I. Zobolas, C.D. Tarantilis, and G. Ioannou. Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. *Computers and Operations Research*, 36:1249–1267, 2009.