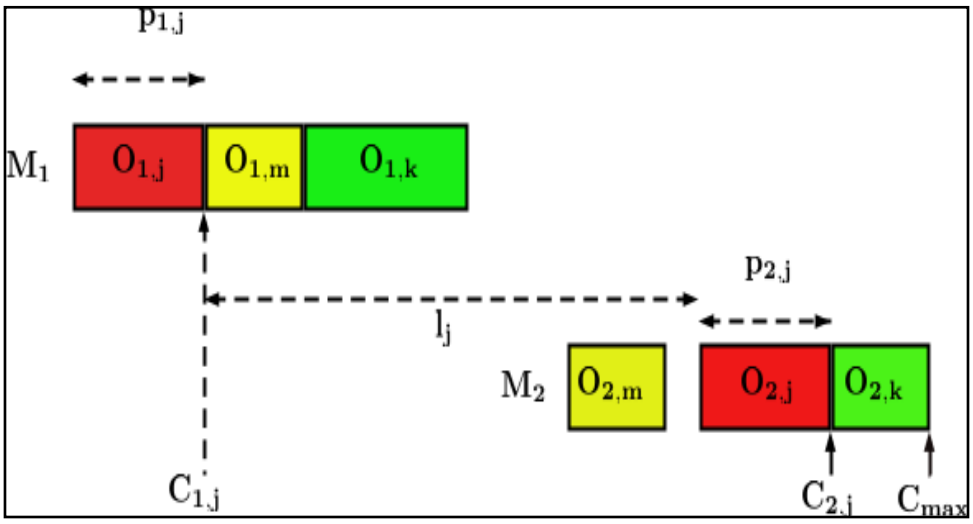
Flow-shop with time delays, linear modeling and exact solution approaches

Mohamed Amine Mkadem

Par **Mohamed Amine MKADEM**

Flow-shop with time delays, linear modeling and exact solution approaches

Thèse présentée pour l'obtention du grade de Docteur de l'UTC



Soutenue le 7 décembre 2017
Spécialité : Technologies de l'Information et des Systèmes :
Unité de recherche Heudyasic (UMR-7253)

D2390

Flow-shop with time delays, linear modeling and exact solution approaches

Par Mohamed Amine MKADEM

Spécialité : Technologies de l'Information et des Systèmes

Soutenue le 7 décembre 2017 devant le jury composé de :

M. E. SANLAVILLE (Président)
M. A. MOUKRIM (Directeur de thèse)
M. M. SERAIRI (Directeur de thèse)
M. A. OULAMARA (Rapporteur)
M. N. TCHERNEV (Rapporteur)
M. J. CARLIER
M^{me} A. YALAOUI

Flow-shop avec temps de transport, modélisation linéaire et approches de résolution exacte

Mohamed Amine MKADEM

07/12/2017

Membres du jury :

Jacques Carlier	Université de Technologie de Compiègne	Examinateur
Aziz Moukrim	Université de Technologie de Compiègne	Directeur de Thèse
Ammar Oulamara	Université de Lorraine	Rapporteur
Eric Sanlaville	Université du Havre	Examinateur
Mehdi Serairi	Université de Technologie de Compiègne	Directeur de Thèse
Nikolay Tchernev	Université Clermont Auvergne	Rapporteur
Alice Yalaoui	Université de Technologie de Troyes	Examinateur

Université de Technologie de Compiègne

Laboratoire Heudiasyc UTC UMR CNRS 7253

Technologies de l'Information et des Systèmes

Université de Technologie de Compiègne

Résumé

Titre : Flow-shop avec temps de transport, modélisation linéaire et approches de résolution exacte.

Dans le cadre de cette thèse, nous traitons le problème de *flow-shop* à deux machines avec temps de transport où l'objectif consiste à minimiser le temps de complétion maximal.

Dans un premier temps, nous nous sommes intéressés à la modélisation de ce problème. Nous avons proposé plusieurs programmes linéaires en nombres entiers. En particulier, nous avons introduit une formulation linéaire basée sur une généralisation non triviale du modèle d'affectation pour le cas où les durées des opérations sur une même machine sont identiques.

Dans un deuxième temps, nous avons élargi la portée de ces formulations mathématiques pour développer plusieurs bornes inférieures et un algorithme exact basé sur la méthode de coupe et branchement (*Branch-and-Cut*). En effet, un ensemble d'inégalités valides a été considéré afin d'améliorer la relaxation linéaire de ces programmes et d'accélérer leur convergence. Ces inégalités sont basées sur la proposition de nouvelles règles de dominance et l'identification de sous-instances faciles à résoudre. L'identification de ces sous-instances revient à déterminer les cliques maximales dans un graphe d'intervalles. En plus des inégalités valides, la méthode exacte proposée inclut la considération d'une méthode heuristique et d'une procédure visant à élaguer les noeuds.

Enfin, nous avons proposé un algorithme par séparation et évaluation (*Branch-and-Bound*) pour lequel, nous avons introduit des règles de dominance et une méthode heuristique basée sur la recherche locale.

Nos expérimentations montrent l'efficacité de nos approches qui dominent celles de la littérature. Ces expérimentations ont été conduites sur plusieurs classes d'instances qui incluent celles de la littérature, ainsi que des nouvelles classes d'instances où les algorithmes de la littérature se sont montrés peu efficaces.

Mots Clés : *recherche opérationnelle, flow-shop, temps de transport, programmation linéaire en nombres entiers, bornes inférieures, règles de dominance, heuristiques, inégalités valides, branch-and-cut, branch-and-bound.*

Directeurs de thèse : Aziz Moukrim et Mehdi Serairi

Université de Technologie de Compiègne

Abstract

Title: Flow-shop with time delays, linear modeling and exact solution approaches.

In this thesis, we study the two-machine flow-shop problem with time delays in order to minimize the makespan. First, we propose a set of Mixed Integer Programming (MIP) formulations for the problem. In particular, we introduce a new compact mathematical formulation for the case where operations are identical per machine.

The proposed mathematical formulations are then used to develop lower bounds and a branch-and-cut method. A set of valid inequalities is proposed in order to improve the linear relaxation of the MIPs. These inequalities are based on proposing new dominance rules and computing optimal solutions of polynomial-time-solvable sub-instances. These sub-instances are extracted by computing all maximal cliques on a particular *Interval graph*. In addition to the valid inequalities, the branch-and-cut method includes the consideration of a heuristic method and a node pruning procedure.

Finally, we propose a branch-and-bound method. For which, we introduce a local search-based heuristic and dominance rules.

Experiments were conducted on a variety of classes of instances including both literature and new proposed ones. These experiments show the efficiency of our approaches that outperform the leading methods published in the research literature.

Key Words: *Operations Research, Flow-Shop, Time Delays, Integer Programming, Lower Bounds, Dominance Rules, Heuristics, Valid Inequalities, Branch-and-Bound, Branch-and-Cut.*

Supervisors: Aziz Moukrim and Mehdi Serairi

To my lovely family ...

Acknowledgements

In the name of God, Most Gracious, Most Merciful.

I would like to express my special appreciation and thanks to my remarkable supervisors Prof. Aziz Moukrim and Dr. Mehdi Serairi, whose expertise, understanding, and patience, added considerably to this research. I appreciate their guidance, support and continuous enthusiasm and interest.

I am also very grateful and extend my sincere thanks to members of the jury Prof. Jacques Carlier, Prof. Ammar Oulamara, Prof. Eric Sanlaville, Prof. Nikolay Tchernev and Dr. Alice Yalaoui for their acceptance to be member of the jury and for their rewarding discussion and helpful comments.

To my family, I am particularly thankful. Words cannot express how grateful I am to my parents for all the sacrifices that they have made on my behalf. Last but not least, I would also like to thank all of my friends and labmates who supported me and assisted me to achieve my goal.

Contents

List of figures

List of tables

List of algorithms

Publications

Articles dans des journaux

- **M. A. Mkadem**, A. Moukrim et M. Serairi. An exact method for the two-machine flow-shop problems with time delays. Submitted to Annals of operations research.

Ouvrages scientifiques

- **M. A. Mkadem**, A. Moukrim and M. Serairi. Lower bounds for the two-machine flow shop problem with time delays. In A. Fink, A. Fügenschuh, and M. J. Geiger, editors, Operations Research Proceedings 2016. Springer International Publishing, 2017 ([Mkadem et al., 2018]).

Conférences internationales avec actes

- **M. A. Mkadem**, A. Moukrim and M. Serairi. A branch-and-bound algorithm for the two-machine flow-shop problem with time delays. In: Proc. of CoDIT'17, The International Conference on Control, Decision and Information Technologies, Barcelona, Spain, April, 2017 ([Mkadem et al., 2017b]).

- **M. A. Mkadem**, A. Moukrim and M. Serairi. An exact method for solving the two-machine flow-shop problem with time delays. In: Proc. of MAPSP 2017, Seeon-Abbey, Germany, June, 2017 ([Mkadem et al., 2017a]).

Conférences nationales et internationales

- **M. A. Mkadem**, A. Moukrim and M. Serairi. Lower bounds for the two-machine flow shop problem with time delays. OR-2016 Annual International Conference of the German Operations Research Society, Hamburg, Germany, Sept, 2016 ([Mkadem et al., 2016a]).

- **M. A. Mkadem**, A. Moukrim et M. Serairi. Le problème flow-shop à deux machines avec temps de transport. ROADEF 2016, Compiègne, France, Feb, 2016 ([Mkadem et al., 2016b]).

Revues en préparation

- **M. A. Mkadem**, A. Moukrim et M. Serairi. MIP formulations and lower bounds for the two-machine flow-shop problems with time delays.
- **M. A. Mkadem**, A. Moukrim et M. Serairi. A mixed integer programming approach for the two-machine flow shop problems with time delays.

Introduction

Nowadays, methods of management and production have a large economic impact on the competitiveness of the industries, and become essential for their survival. However, previous studies suggest that management methods such as planning, scheduling, staffing, directing and controlling are not done properly in many small and medium industries ([Asil and Naralan, 2016]). On this basis, the challenge of solving management problems is more and more attractive to researchers who try to enhance the existing methods and models. Scheduling methods, in particular, gain an increasing interest since they are widely used in a large number of practical operations management software. In fact, scheduling can be defined as a task that determines when each activity should start or end depending on many constraints. These constraints may include the duration, predecessor activities, resource availability and the objective of the project. Also, this task manages to allocate resources over time to realize these activities. Depending on the nature of the resources and the activities, scheduling problems cover a variety of application areas. Some examples arise in project scheduling, personnel scheduling, timetabling problems, etc..

Within all these categories of problems, the manufacturing environment stands as the most common and attractive application area. Indeed, the resolution of such problems in the manufacturing system can be included as frameworks in Decisions Support Systems. These frameworks represent interesting tools to companies that allow them to optimize their manufacturing processes. As a consequence, they yield significant benefit by achieving solution at minimum total cost and time.

Among the manufacturing environments, the Flexible Manufacturing System (FMS) has gained an increasing interest in the last decade. This system is a highly automated manufacturing one. It is composed of a set of processing workstations, interconnected by a material handling system that serves to optimize parts flow. It is controlled by a computer which manages material movements and machine flow. This system is called flexible for two reasons. First, the machine flexibility permits to process different product types simultaneously, and can change the execution order

of operations. Second, routing flexibility gives the ability to use multiple machines to do the same job. In addition, the system is able to engage large-scale changes.

We are interested in proposing new approaches to solve scheduling problems in the flexible manufacturing system. Precisely, we study two variants of the flow-shop problem. In such a problem, we dispose of a set $J = \{1, ..., n\}$ of n jobs to be executed in the same order on a set $M = \{M_1, ..., M_m\}$ of m machines. It is assumed that each job has m operations and it never revisits any machine. Thus, each job of J has to visit them in the numerical order. Precisely, each job has to be executed first on M_1 during a certain amount of time, then on M_2 and so forth until M_m. In the classic flow-shop problem, it is assumed that once the execution of a job is accomplished on a given machine, the job becomes instantaneously available for processing on the next one. However, in many real applications, each job needs an intermediate delay between each pair of machines. This delay can be due to transportation, cooling, heating or drying activities. Depending on the characteristics of the product, a variety of time delay values may be observed. The derived problem is called the flow-shop problem with time delays. Note that the flow-shop problem can be solved in $O(n \log n)$-time for $m = 2$, and is NP-hard in the strong sense for $m \geq 3$. However, the two-machine flow-shop problem with time delays is NP-hard in the strong sense even with unit-time operations. Thus, the investigation of exact and heuristic methods is well justified.

In this dissertation, we introduce several methods to solve the two-machine flow-shop problem with time delays. Precisely, we propose new lower bounds, heuristic approaches, mixed integer programming models and a branch-and-bound method to solve the no-restricted problem and a particular case. This thesis is organized as follows. Chapter 1 provides, in the first part, an introduction to the different optimization problems related to scheduling problems. The second part is concerned with giving the reader a background on the typology of scheduling problems. We also define the two-machine flow-shop problems and recall some known methods used to solve them. Finally, we introduce the notation used and recall basic properties.

After the overview of the field of study, we propose in Chapter 2 an efficient non trivial Integer Linear Programming (ILP) formulation for a specific case of the two-machine flow-shop problem with time delays in order to minimize the makespan. The particularity of this problem is that the processing times are identical per machine. Moreover, we introduce a pre-processing procedure that aims at reducing the number of decision variables of the mathematical formulation, and

thus improving its performance. The computational results show that the proposed approach outperforms the branch-and-bound method of [Moukrim et al., 2014] on the unit-time operations's variant of the problem.

In Chapter 3, we tackle the two-machine flow-shop problem with time delays in order to minimize the makespan. First, we present a comprehensive theoretical analysis of the different lower bounds in the literature and we elucidate dominance relationships between them. Moreover, we provide several mathematical formulations. We point out the good quality of the linear relaxation of a Mixed Integer Linear Programming (MIP) formulation that is based on linear ordering variables. A set of valid inequalities is proposed. In particular, we introduce clique-based valid inequalities. These inequalities are based on computing optimal solutions on easy-to-solve sub-instances. These sub-instances are identified after computing all maximal cliques on a particular *interval graph*. Furthermore, three linear programming-based lower bounds are introduced for the problem. The proposed approaches demonstrate an excellent performance from both the point of view of effectiveness and efficiency. Precisely, the derived lower bounds deliver the best lower bound value on all literature instances.

In the continuity of Chapter 3, we investigate in Chapter 4 two exact methods for the flow-shop scheduling problem with two machines and time delays with respect to the makespan. First, we introduce a branch-and-cut algorithm based on a linear ordering variable-based mathematical formulation. This method includes the implementation of dominance rules and valid inequalities. Moreover, a critical path-based branching scheme, a new heuristic method and a node pruning procedure are considered. Then, we propose an exact algorithm based on a branch-and-bound enumeration scheme, for which we introduce a heuristic method based on a local search technique and three dominance rules. Finally, we present the results of an extensive computational study that was carried out on a set of 480 instances including new hard ones. Our exact methods outperform the exact method of [Dell'Amico, 1996].

Finally, we provide a general conclusion that includes a synthesis of the contributions presented in this thesis and some perspectives and future works.

Field of Study

Contents

1.1 Introduction

In this chapter, we introduce the combinatorial problem basics in general and the scheduling ones in particular. First, we introduce the combinatorial problems and their related terminologies. We then present a list of general resolution methods for these problems. Next, an introduction to scheduling theory is given. We first describe the typology of scheduling problems. After that, we introduce the flow-shop problem. In particular, we study the two-machine flow-shop problem with time delays. Furthermore, we provide the notation used and basic properties. Finally, the chapter ends with a conclusion.

1.2 Combinatorial Optimization

This section is organized as follows. First, we define the combinatorial optimization problems. We then present notion, algorithms and complexity for this kind of problems. The main objective is to recall the main definitions and related terminologies.

1.2.1 Combinatorial Optimization Problems

The optimization problem is the problem of finding the best solution according to an objective function from a set of feasible solutions, which are with respect to a set of constraints. Two categories of optimization problems are identified depending on whether the variables are continuous or discrete. If we dispose of discrete variables, then the problem is known as a combinatorial optimization problem.

Definition 1.1. *A combinatorial optimization problem* Φ*=(*Ω*,f) can be defined as follows:*

- *A set of variables* $X=\{X_1, ..., X_n\}$.
- *Every variable* X_k *is associated to a domain* D_k.
- *A set of constraints that links the variables.*
- *An objective function either to minimize or to maximize:* $f: D_1 \times ... \times D_n \longrightarrow \mathbb{R}$.

The set Ω *stands for the search space. Given a feasible solution* s *to the problem* Φ*, then it holds that* $s \in \Omega$ *such that:*

$$s = \{v_1, ..., v_n | v_k \in D_k \text{ and all the constraints are verified}\}.$$

Solving the problem Φ *with a minimization objective function (resp. maximization) consists in finding a feasible solution* s^* *such that* $f(s^*) \leq f(s)$ *(resp.* $f(s^*) \geq f(s)$*) for all* s *in* Ω.

1.2.2 Complexity Theory

The formal definition of an algorithm is introduced by [Turing, 1936]. This definition is founded on the notion of a formal language using an abstract machine called the *Turing machine*. In fact, an algorithm A consists of a finite series of well-defined instructions that allow to solve a problem Φ. For example, in order to solve a combinatorial optimization problem, the algorithm reports the different steps needed

to obtain the optimal solution s^*. In case the algorithm is a heuristic approach, it provides a solution s that is not guaranteed to be optimal but sufficient for the immediate goals.

Many performance measures are used to analyze the quality of algorithms including the computational time, the required memory, the quality of the solution and the robustness. However, the time and space are the most well-known and used complexity measures. Since the CPU time is related to the characteristics of the machine that we use, it is commonly presented by the number of instructions needed by the algorithm. It is assumed that each instruction is equivalent to a simple elementary operation (Like: addition, multiplication,...).

Definition 1.2. *We define the complexity C_A of an algorithm A, which is introduced to solve a problem Φ of size n, using the number of instructions required to solve any instance of the problem Φ.*

Using Landrau's notation, an upper bound of C_A is expressed asymptotically to the size n using the notation O. An algorithm A is said to be of complexity $O(g(n))$ if there exist two parameters $M > 0$ and n_0 such that $\forall n \geq n_0$, $C_A \leq M \times g(n)$. Depending to g, the most known complexities are described as follows:

- $O(1)$: constant complexity that is independent of the size of Ω.
- $O(\log n)$: logarithmic in the size of Ω.
- $O(n)$: linear in the size of Ω.
- $O(n^k)$ (with $k \geq 2$): polynomial in the size of Ω.
- $O(a^n)$ (with $a > 1$): exponential in the size of Ω.

In some cases, the complexity of the algorithm can be related in the same time to the problem size n and to some parameters of the problem, e.g. $g(n) = n^p \times v^q$ where v is a value of the input and q is a constant. In such a case, the algorithm is called *Pseudo-polynomial*.

1.2.3 Problem complexity

Solving complex problems is very attractive to researchers especially with the recent advance in computer performance. For some optimization problems, no polynomial-time algorithm was found until now to solve them. For others, there exists a way to solve them in a polynomial time. A classification of the problems can be given then

according to the complexity of algorithms that solve them.

We identify two types of problems: optimization and decision problems. An optimization problem consists in finding the best solution from all feasible ones with the minimum or the maximum value of an objective function f. However, a decision problem verifies the existence of a feasible solution with a given objective value. Interestingly, for each optimization problem $\Phi = (\Omega, f)$, we can derive an equivalent decision problem. This decision problem is formulated as follows: given $k \in \mathbb{N}$, does there exist a solution s where $f(s) = k$?

There is a variety of classes of decision problems:

Definition 1.3. *A problem is in class P if a polynomial algorithm that can solve it exists.*

Definition 1.4. *A decision problem is in class NP if it can be proven in a polynomial time that a given solution is valid.*

Definition 1.5. *A decision problem is called NP-Complete if it is in class NP but not in class P. Moreover, every NP problem can be reduced into this problem in polynomial time.*

Definition 1.6. *An optimization problem is called NP-Hard if its equivalent decision problem is NP-Complete.*

Definition 1.7. *An optimization problem (or decision) is NP-Hard (resp. NP-Complete) in the ordinary sense if it is NP-Hard (resp. NP-Complete) and a pseudo-polynomial algorithm that can solve the problem exists.*

As a consequence, it holds that P $\subseteq$ NP. However, the question if P $\subset$ NP or P = NP is not answered yet. Assuming that P = NP implies the existence of a polynomial-time algorithm for each NP problem, and then the set NP-complete is empty. For every NP-complete problem, no polynomial-time algorithm has yet been found. Therefore, it is more likely that P $\neq$ NP.

1.2.4 Solution approaches

In this section, we provide a general overview of the different methods used to solve combinatorial optimization problems. There are two types of methods: exact methods and approximation methods. However, before starting the resolution of a combinatorial optimization problem, pre-processing methods can be applied in

order to limit the size of the problem or to discover useful dependencies to make the problem easier to solve. First, we present the pre-processing methods. We then describe some exact methods including tree search and mixed integer programming based ones. Finally, approximation methods are introduced.

1.2.4.1 Pre-processing

The pre-processing is a preliminary method performed on the problem in order to make the resolution process easier. After analyzing the data and its nature, this technique usually limits the search space of the problem. For example, after analyzing the constraints of a combinatorial optimization problem Φ, we can fix a set of variables X_i or we can reduce the size of D_i. However, we should guarantee that the optimal solution is not affected since the search space can be modified.

1.2.4.2 Exact algorithms

Exact algorithms are methods that compute optimal solutions and ensure their optimality for every instance of a combinatorial optimization problem. Among these methods, we find the branch-and-bound (B&B), the dynamic programming, the lagrangian relaxation based methods, and the linear and integer programming based methods such as branch-and-cut, branch-and-price and branch-price-and-cut. In this section, we define the theoretical bounds and we present two general exact approaches used to solve combinatorial optimization problems.

Theoretical Bounds: For combinatorial optimization problems, bounds define boundaries on the search space that includes the optimal solution. The search for methods of resolution is based on the possible improvements that can be applied to their bounds. In practice, upper and lower bounds are always defined.

Definition 1.8. *We define a lower bound LB (resp. an upper bound UB) on the objective function of a combinatorial optimization problem $\Phi = (\Omega, f)$ if for all s in Ω, $f(s) \geq LB$ (resp. $f(s) \leq UB$).*

For a minimization problem, the upper bounds (UB) are obtained using heuristic approaches and the lower bounds (LB) are determined after solving relaxed versions of the problem. If it holds that $LB = UB$ for an instance I, then I is solved to optimality.

The branch-and-bound scheme: The branch-and-bound scheme is a systematic method that enumerates solutions by means of state space search. Applied

on combinatorial optimization problem $\Phi = (\Omega, f)$, the algorithm enumerates sub-spaces $(S_1, ..., S_k)$ that are obtained after successive branching on Ω. These sub-spaces are then bounded by calculating lower and upper bounds of Ω on every sub-space. Depending on the bounds values, some sub-spaces can be discarded under some conditions. For example, given a minimization problem, a sub-space S_i can be fathomed if $LB(S_i) \geq UB_{best}$, where UB_{best} stands for the best known upper bound value. If we dispose of an efficient evaluation and pruning strategies, the branch-and-bound usually becomes faster than the case where a complete enumeration of all possible solutions is done.

The efficiency and the effectiveness of a branch-and-bound method in term of CPU time depends on the following factors:

- Bounds: lower and upper bounds.
- Heuristic: a method that computes a feasible solution on the problem without guarantying its optimality.
- Dominance rules: a set of constraints that are used to reduce the search space. These constraints guarantee the respect of the optimal solution.
- Branching rule: the strategy adopted to generate sub-spaces for possible exploration.
- Branching strategy: the strategy adopted to select a node on which we branch.

Linear Programming: A linear program is a continuous optimization problem with linear constraints and objective function. This method is a special case of mathematical optimization where the objective is to achieve the best outcome in a mathematical model. The exact resolution of a linear program is done in polynomial time using the ellipsoid method of [Khachiyan, 1980]. Many commercial softwares are available to solve this kind of problem like: CPLEX, GUROBI, XPress and SCIP. Most of these softwares use the simplex method despite that it is an exponential-time algorithm.

We can generally model every combinatorial optimization problem using Mixed Integer linear Programming. The fact of having integer variables makes the resolution process harder. We present in Figure 1.1 the difference between a MIP and an LP in term of solutions space. For the relaxed version of the model, we observe that the LP solution does not belong to the convex hull of the strong formulation. The bold dots stands for the ILP solutions that are determined after solving the MIP problem. In order to solve MIPs, we identify efficient techniques that are based on the

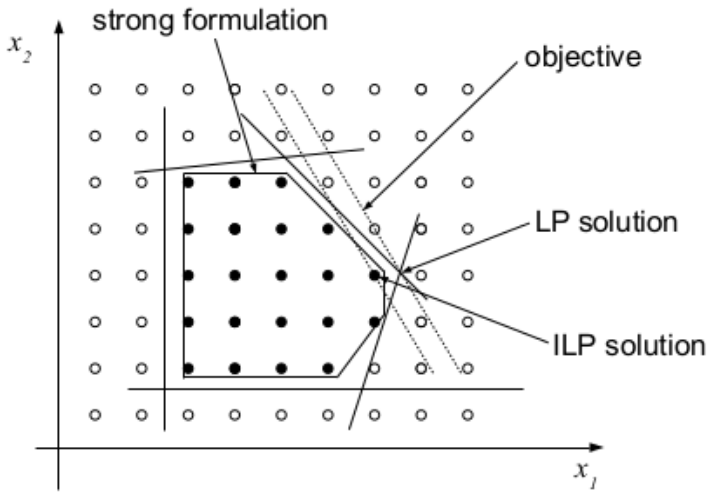


Figure 1.1 – Example of Mixed Integer Programming (MIP).

branch-and-bound scheme. In these techniques, we incorporate some mathematical tools as cutting planes. In case the cutting planes are included in the branch-and-bound algorithm, the algorithm is denoted branch-and-cut.

A combinatorial optimization problem can be modeled in several ways using an LP or a MIP. We say that a model is strong in case the integrity constraint is relaxed by the polyhedron. The fact of integrating cutting planes can strengthen the model. However, in some problems, we can dispose of an exponential number of variables and thus the search tree would be huge. In this case, we can efficiently solve the problem using column generation of [Desaulniers et al., 2006]. Precisely, we integrate the column generation in a branch-and-bound algorithm. The obtained algorithm is called a branch-and-price algorithm ([Desaulniers et al., 2006]). This latter disposes of a problem-dependent scheme.

The exact methods are often extremely time-consuming when solving real-world problems (i.e. problems with large dimensions, hardly constrained problems, multimodal and/or time-varying problems). In this case, the only possibility to handle larger instances of problems is to trade optimality for run-time. As a consequence, we use approximation algorithms.

1.2.4.3 Approximation algorithms

Heuristic and metaheuristic techniques are powerful and flexible search methodologies that have successfully tackled practical difficult problems. Heuristic and metaheuristic algorithms seek to produce good-quality solutions in reasonable

computation times and good enough for practical purposes. These exist two categories of these methods: PTAS and non-PTAS. The first category involves the algorithms with polynomial complexity that assure a certain distance from the optimal solution in the worst case. These algorithms are called PTAS (Polynomial-Time Approximation Scheme). For the second category, its algorithms generally produce very good solutions in a short time. However, no distance from the optimal solution is guaranteed. These algorithms are called non-PTAS (non Polynomial-Time Approximation Scheme).

1.3 Scheduling theory

Scheduling theory is a research area that was introduced in 1950s. It often arises when the availabilities of resources are critical for long time planning decision. Scheduling consists in determining the starting times and the end times of a set of tasks. It also manages to allocate resources over time to realize these tasks.

Scheduling theory has been the scope of a variety of investigations. Motivation for scheduling problems comes from industrial applications, examples arise in workforce scheduling, crew scheduling, tournament scheduling and timetabling.

In this section, we first describe the typology of scheduling problems. Then, we study the flow-shop problem. In particular, we focus on the two-machine flow-shop problem with time delays. Finally, the notation used and some basic properties are given.

1.3.1 The typology of scheduling problems

A typology can be defined as a classification of problems depending to their nature. For scheduling problems, it is based on the environment of machines and the particularities of jobs. In order to point out a problem, the notation in scheduling are usually founded on the existing typologies.

[Graham et al., 1979] introduced a short-hand notation that permits to specify problems. The classification system is composed from three fields that are separated by bars: $\alpha|\beta|\gamma$.

The first field α stands for the shop environment. This field can be represented by two sub-fields $\alpha = \alpha_1\alpha_2$, one for the type and the other for the problem size. For

example:

- $\alpha = 1$: Single machine scheduling problem. It is widely used in computing to schedule tasks on a single processor of a computer.
- $\alpha_1 \in \{P, Q, R\}$: Parallel machines scheduling problem. It can be considered as an extension of the single machine environment, in which we dispose of a set of m machines that are available to process all jobs. We identify three types of this problem including: Identical parallel machines ($\alpha_1 = P$: the processing time of every job is independent of the machine), Uniform parallel machines ($\alpha_1 = Q$: the processing time of every job is dependent to the machine speed) and Unrelated parallel machines ($\alpha_1 = R$: every machine has a different speed, but this speed is dependent of the job).
- $\alpha_1 = J$: Job-shop problems. We dispose of a set of machines that are organized in series. Every job has to be scheduled on every machine. The execution order of every job is dependent to the job.
- $\alpha_1 = F$: Flow-shop problems. It is similar to the job-shop problem. Except that the jobs have the same executing order.
- $\alpha_1 = O$: Open-shop problems. It is similar to the job-shop problem. However, no restriction on the job routing is given.

An additional information about the problem is given by the field α_2. For example, $\alpha = F2$ (i.e. $\alpha_1 = F$ and $\alpha_2 = 2$) means that the problem is a two-machine flow-shop.

The second field β is used to present the job characteristics. We provide the following examples:

- r_j: Release date. It is the date at which a job j arrives on a machine and becomes ready for processing.
- d_j: Due date. It is the date at which a job j should be finished.
- prec: Precedence relationships. There exist some precedence relationships between jobs.
- perm: Permutation. In a flow-shop environment, this means that only permutation schedules are considered: the schedules in which the same job order is maintained at all machines.

- pmtn: Preemption. The preemption of operations is allowed.

The third field γ stands for the optimality criterion. The most common criteria are:

- C_{max}: Makespan. Here we minimize the maximal completion time on all machines.

- $\sum C_j$ (resp. $\sum w_j C_j$): Total completion time (resp. Total weighted completion time) of all jobs.

- L_{max}: Maximum lateness (resp. T_{max}: Maximum tardiness). We minimize the worst violation of the due dates. The lateness (resp. the tardiness) of a job is positive if it completes late and negative (resp. zero) otherwise.

- $\sum T_j$ (resp. $\sum w_j T_j$): Total tardiness (resp. Total weighted tardiness) of all jobs.

In some cases, we can dispose of more than one objective to achieve. Generally, we cannot handle all objectives at the same time. However, it is possible to incorporate all goals in the same objective. This kind of objective function is called multicriteria objective. For the case of two objectives X and Y, we identify three ways to formulate multicriteria objectives:

- multicriteria objective $X|Y$: optimize X subject to a constraint on Y. For instance, we can minimize the total completion time of jobs with respect to an upper bound value on the makespan ($\sum C_j | C_{max} \leq D$).

- composite objective $\alpha X + \beta Y$: optimize a linear combination of the two objectives using pre-specified weights α and β.

- bicriteria objective (X, Y): determine the schedules that provide the best pairs of values. Then, it is to the decision maker to select the best option.

Interestingly, [Pinedo, 2008] established a complexity hierarchy between a set of well-known objective functions for scheduling problems. This complexity hierarchy is illustrated by Figure 1.2. An outgoing arrow go from easy to harder problems.

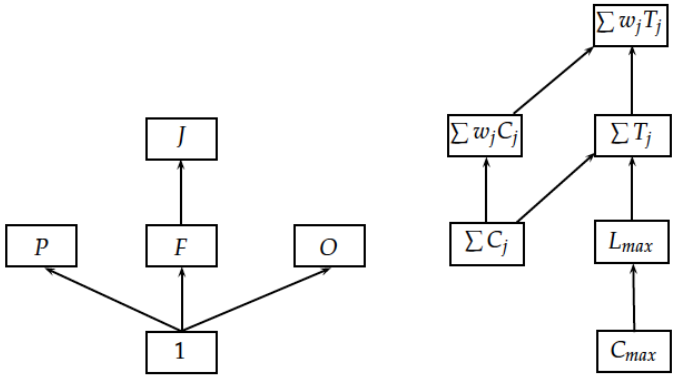


Figure 1.2 – Scheduling objectives hierarchy.

We observe that:

- C_{max} can be reduced to L_{max} by given the value zero to d_j for each job j.
- $\sum C_j$ can be reduced to $\sum T_j$ by given the value zero to d_j for each job j.
- $\sum C_j$ (resp. $\sum T_j$) can be reduced to $\sum w_j C_j$ (resp. $\sum w_j T_j$) by given the value 1 to w_j for each job j.
- $\sum w_j C_j$ can be reduced to $\sum w_j T_j$ by given the value zero to d_j for each job j.

1.3.2 Flow-shop problem

[Emmons and Vairaktarakis, 2013] define a flow-shop as a processing system in which the task sequence of each job is fully specified, and all jobs visit the work stations in the same order. Generally, each job must undergo several stages of work at a series of work stations or machine groups. In most works, every stage can be visited at most one time by a job. Each work station is identified by a unique number from the set $\{1, ..., m\}$, where m is the number of stations. As a consequence, every job is scheduled on the work stations in numerical order. In the case of a pure flow-shop, every job consists of m tasks and needs to visit all stages. Thus, we assume that a job cannot visit a stage i before visiting all stages $j < i$.

In this manuscript, we limit to a two-machine flow-shop environment. First, we recall the main contributions for the classic two-machine flow-shop problem. Then, we study the two-machine flow-shop problem with time delays.

A variety of two-machine flow-shop problems were studied in the literature. The well-known $F2||C_{max}$ problem was studied by [Johnson, 1954] where

a $O(n \log n)$-time exact method was proposed. [Lenstra et al., 1977] showed that $F2|(perm), r_j|C_{max}$ problem is ordinary NP-complete. Heuristics have been proposed for this problem by [Potts, 1985]. Moreover, [Hall, 1994] and [Kovalyov and Werner, 1997] proposed polynomial approximation schemes. Minimizing the total completion times in the two-machine flow-shop problem $F2|(perm)|\sum C_j$ was proved to be NP-complete by [Garey et al., 1976] even though an optimal permutation schedule exists ([Conway et al., 1967]). Some special cases of $F2|(perm)|\sum C_j$ have simple polynomial solutions. For others, branch-and-bound algorithms were developed and supported by lower bounding schemes (see [Emmons and Vairaktarakis, 2013]). Moreover, [Allahverdi and Al-Anzi, 2002] proposed a branch-and-bound method to solve the $F2|(perm), s_j|L_{max}$ that is equivalent to $F2|(perm), s_j|T_{max}$. The specificity of these two problems is that each job has a separable and sequence-independent setup time s_j on both machines, and a due date d_j, as well as processing times.

In the classic flow-shop problem, it is assumed that once the execution of a job is accomplished on a given machine, the job instantaneously becomes available for processing on the next one. However, in many real applications, each job needs a certain delay between each pair of machines. Depending on the characteristics of the product, a variety of time delay values may be observed. In some problems, minimal and maximal time delays are considered between each pair of successive operations. Minimal time delays are considered when waiting times are imposed between each two successive operations of a job. These delays are commonly encountered when an intermediate processing is needed, such as material handling ([Soukhal et al., 2005]), cooling activity ([Fondrevelle et al., 2006]), and chemical reactions ([Chu and Proth, 1996]). Moreover, maximal time delays arise when the waiting time between the two operations of the same job must not exceed a certain duration. An example arises in the wafer fabrication process ([Su, 2003]). After the operations in furnace tubes, the waiting time must not exceed a certain duration in order to prevent the absorption of the particulates in air. Motivation for maximal and minimal time delays comes from industrial applications. Examples arise in Agribusiness ([Hodson et al., 1985]), the production of printed circuits ([Kim et al., 1996]), and in chemistry and biotechnology ([Nawaz et al., 1983]). We identify four types of time delays:

- A start-start time delay: is the time delay that is required between the starting time of a job j on the k-th work station and its starting time on the $(k+1)$-th work station, k in $\{1, ..., m-1\}$.

- A finish-finish time delay: is the time delay that is required between the completion time of a job j on the k-th work station and its completion time on the $(k+1)$-th work station, k in $\{1, ..., m-1\}$.

- A start-finish time delay: is the time delay that is required between the starting time of a job j on the k-th work station and its completion time on the $(k+1)$-th work station, k in $\{1, ..., m-1\}$.

- A finish-start time delay: is the time delay that is required between the completion time of a job j on the k-th work station and its starting time on the $(k+1)$-th work station, k in $\{1, ..., m-1\}$.

Interestingly, it not needed to work on all these time delays since each type can be reduced to any other.

Many researches have been carried out on flow-shop problems with time delays in order to minimize the makespan. [Fondrevelle et al., 2006] introduced an exact method based an a branch-and-bound scheme for the m-machine permutation flow-shop problem with minimal and maximal time lags. Moreover, [Ye et al., 2017] proposed an iterated greedy heuristic for the non-permutation flow-shop scheduling problems with maximal and minimal time lags. Contributions, where only maximal time delays are considered, are identified. Several authors addressed numerous variants of the problem by providing exact algorithms based on a branch-and-bound scheme. A non-exhaustive list includes the two-machine permutation flow-shop with limited waiting times ([Yang and Chern, 1995]) and the two-machine flows-shop scheduling problem with limited waiting times and sequence-dependent setup times ([An et al., 2016]).

In this thesis, we study the two-machine flow-shop with minimal time delays in order to minimize the makespan denoted by $F2|l_j|C_{max}$. Motivation for $F2|l_j|C_{max}$ comes from industrial applications, an example arises in the case of consolidators ([Emmons and Vairaktarakis, 2013]). These latter provide front-end support logistic services to manufacturer, usually by supplying firms by raw materials or other orders, which considered to be as the first operation. The industrial treatments make up the time delay and the second operation consists in delivering goods to their destinations.

From complexity point of view, $Fm|l_j|C_{max}$ is NP-hard in the strong sense even with $m = 2$. At first, $F2|l_j|C_{max}$ was shown to be NP-complete by [Lawler et al., 1993]. Moreover, [Yu et al., 2004] strengthened this result after showing that $F2|p_{i,j} = 1, l_j|C_{max}$, which is a unit-time operations variant of

$F2|l_j|C_{max}$, is NP-hard in the strong sense. Other special cases were identified and proved to be NP-hard in the strong sense, e.g., $F2|p_{1,j} = p_{2,j}, l_j|C_{max}$ ([Dell'Amico and Vaessens, 1996]) and $F2|p_{1,j} = p_{2,j}, l_j \in \{0,1\}|C_{max}$ ([Yu, 1996]). Furthermore, [Dell'Amico, 1996] interested in the preemptive flow-shop problem $F2|pmtn, l_j|C_{max}$, which allows the interruption and the switching of operations being carried out in both machines. He showed that this problem is NP-hard in the strong sense after proving that every $F2|pmtn, l_j|C_{max}$ feasible schedule is equivalent to a $F2|l_j|C_{max}$ feasible one with the same makespan value. From the other hand, [Yu, 1996] proposed an algorithm for $F2|p_{i,j} = 1, l_j \in \{0,T\}|C_{max}$. This algorithm can determine an optimal non-permutation solution in $O(n \log n)$-time. Beside that, [Mitten, 1959] interested in the $F2\pi|l_j|C_{max}$ problem, in which the processing order of jobs is the same on both machines (permutation schedule). He showed that this problem is equivalent to $F2||C_{max}$, therefore it is computed in $O(n \log n)$-time using the algorithm of [Johnson, 1954].

It is noteworthy to indicate that permutation schedules are not dominant for $F2|l_j|C_{max}$. However, special cases exist where it is true. A permutation schedule that is optimal exists for an instance I if all time delays are equal to a constant T. This optimal solution is found by solving the classic $F2||C_{max}$ problem for I using the algorithm of [Johnson, 1954] and then shifting the obtained makespan value by T time units. Moreover, [Dell'Amico, 1996] mentioned that permutation schedules are dominant if one of the following conditions holds:

$$\max_{j=1}^{n}(l_j) \leq \min_{j=1}^{n}(p_{1,j} + l_j) \text{ or } \max_{j=1}^{n}(l_j) \leq \min_{j=1}^{n}(p_{2,j} + l_j). \tag{1.1}$$

Interestingly, [Yu, 1996] extended this result after showing that permutation solutions are dominant even when:

$$l_i \leq l_j + max(p_{1,j}, p_{2,j}),\ \forall i, j \in J. \tag{1.2}$$

$F2|l_j|C_{max}$ has been the scope of a variety of investigations. Lower bound methods were proposed by [Dell'Amico, 1996] and [Yu, 1996]. Heuristic approaches were also investigated by [Dell'Amico, 1996], [Karuno and Nagamochi, 2003] and [Zhang and van de Velde, 2010]. Precisely, [Dell'Amico, 1996] developed a set of 2-approximation constructive heuristics and a Tabu Search algorithm. [Karuno and Nagamochi, 2003] introduced a 11/6-approximation algorithm that is computed in $O(n \log n)-$time. Moreover, a Polynomial-Time Approximation scheme was introduced by [Zhang and van de Velde, 2010]. As far as we know, a unique

exact method was implemented. [Dell'Amico, 1996] introduced an exact method based on a branch-and-bound scheme to solve $F2|l_j|C_{max}$. He applied all of his lower bounds and approximation procedures at the root node of the branch-and-bound method.

1.3.3 Preliminaries and basic properties for $F2|l_j|C_{max}$

In this section, we introduce the notation used and we recall basic properties for the $F2|l_j|C_{max}$ problem. Let $I = (J, p_1, l, p_2)$ be an instance of $F2|l_j|C_{max}$, where $J = \{1, 2, ..., n\}$ is a set of n jobs, p_1 and p_2 are the vectors of processing times on the first and the second machines. Moreover, l is the vector of the time delays. Each job j has two operations, $O_{1,j}$ and $O_{2,j}$. The first operation $O_{1,j}$ (resp. the second operation $O_{2,j}$) must be executed without preemption during $p_{1,j}$ (resp. $p_{2,j}$) time units on M_1 (resp. M_2). For each job j of J, a time delay of at minimum l_j time units must separate the end of the first operation and the start of the second one. The objective is to find a feasible schedule that minimizes the completion time of the last scheduled job on M_2. A feasible schedule is such that at most one operation is processed at a time on a given machine. In addition, the operations are executed without preemption, where interruption and switching of operations are not allowed.

Considering the above definitions, we denote by:

- S: a valid schedule of an $F2|l_j|C_{max}$ instance.
- $C_{max}(S)$ (resp. $C^*_{max}(I)$): the makespan value (resp. the optimal makespan value) of schedule S (resp. instance I).
- $t_{m,j}(S)$: the starting time of operation $O_{m,j}$ on M_m in schedule S, for all j in J; m in $\{1, 2\}$.

Given a job sequence σ, we define the following notation:

- J_σ: the set of jobs that constitute job sequence σ.
- Ω_σ: the set of all schedules where job sequence σ is fixed first on M_1.
- $\sigma_{[k]}$: the job scheduled at the k-th position in σ.
- σ_j^{-1}: the position where the job j is scheduled in σ.

Moreover, let us consider the following three observations.

Observation 1.1. *Let us consider a fixed job sequence σ of all jobs on M_1. The schedules of Ω_σ where jobs are scheduled on M_2 in the non-decreasing order of their arrival times are dominant.*

Observation 1.2. *Consider a valid schedule $S = (\sigma, \pi)$ of an instance I of $F2|l_j|C_{max}$ where σ (resp. π) is the job sequence on M_1 (resp. M_2). A valid schedule $S^{'} = (\sigma^{'}, \pi^{'})$ of I exists such that $\sigma^{'} = \sigma$, $\pi^{'} = \pi$ and the jobs are continuously handled on each machine in a way that $C_{max}(S^{'}) = C_{max}(S)$.*

Let $I = (J, p_1, l, p_2)$ and $I' = (J', p'_1, l', p'_2)$ be two instances of $F2|l_j|C_{max}$, where $J' \subset J$ and for all $j \in J'$, $p'_{1,j} \leq p_{1,j}$, $p'_{2,j} \leq p_{2,j}$ and $l'_j \leq l_j$. Given an optimal schedule S^* of instance I, we can determine a feasible schedule $S^{'}$ of instance I' from S^* such that $C_{max}(S^{'}) \leq C^*_{max}(I)$. Thus, the following observation holds.

Observation 1.3. *Let us consider two instances $I = (J, p_1, l, p_2)$ and $I' = (J', p'_1, l', p'_2)$ of $F2|l_j|C_{max}$, where $J' \subset J$ and for all $j \in J'$, $p'_{1,j} \leq p_{1,j}$, $p'_{2,j} \leq p_{2,j}$ and $l'_j \leq l_j$. We observe that $C^*_{max}(I') \leq C^*_{max}(I)$. As a consequence, every lower bound on the makespan of instance I' is also a valid lower bound on the makespan of instance I.*

1.4 Conclusion

In this chapter, we shortly introduced the combinatorial problems and presented some of the resolution techniques used in the literature. Also, we described the scheduling theory in general and we especially detailed the flow-shop problems that are discussed in this thesis.

In the second chapter, we detail a compact mathematical formulation for the case where the processing times are identical per machine. The third chapter presents three lower bounds based on the linear programming for the general problem. Finally, the fourth chapter provides two exact approaches including a branch-and-bound method.

An effective compact formulation for $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$

Contents

2.1 Introduction

In this chapter, we investigate a particular case of the two-machine flow-shop problem with time delays in order to minimize the makespan. The particularity of this problem is that the processing time are machine-dependent, and thus all jobs have the same processing time per machine. Using the notation of [Graham et al., 1979], this problem is denoted by $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$.

Few researches have been conducted on the $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ problem. We recall that [Ageev and Baburin, 2016] presented a 1.628-approximation algorithm. However, various investigations have been carried out for particular cases

of $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$. Especially for the $F2|p_{i,j} = 1, l_j|C_{max}$ problem, [Yu, 1996] proved that the problem is NP-hard in the strong sense and he proposed a set of lower bounds. These latter were improved by [Moukrim et al., 2014] in the case where a sub-sequence of jobs is fixed on the first machine. Moreover, two approximation methods were investigated by [Rayward-Smith and Rebaine, 2008] with a performance ratio of $2 - 3/(n + 2)$ and $\frac{4}{3} - 2/(3n + 3)$, respectively. In addition, [Moukrim et al., 2014] proposed an exact algorithm based on a branch-and-bound scheme.

This chapter is organized as follows. First, we introduce a general assignment problem and we show that this problem and the $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ problem are equivalent. Then, we provide an Integer Linear Programming (ILP) formulation. A pre-processing procedure is also introduced in order to reduce the number of decision variables of the proposed formulation. Moreover, we perform a computer simulation on the instances of $F2|p_{i,j} = 1, l_j|C_{max}$, where we compare the performance of the ILP model to the literature exact method. Actually, the ILP solves 1519 instances out of 1560 possible ones while the branch-and-bound of [Moukrim et al., 2014] fails to solve 320 instances.

The remainder of this chapter is organized as follows. In Section 2.2, the general assignment problem is described, and a proof that this problem and $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ are equivalent is given. In Section 2.3, we introduce an ILP formulation to solve the general assignment problem. A pre-processing procedure is also provided. Computational experiments on a set of randomly generated instances are reported in Section 2.4. Finally, some concluding remarks are given in Section 2.5.

2.2 The general assignment problem

A general assignment problem GA_{ss} is a decision problem that can be defined as follows. We dispose of a set J of n jobs to be handled by two sets of n agents each, referred to as A_1 and A_2. Here, we assume that A_1=A_2=$\{1, ..., n\}$. The aim is to perform all jobs by assigning exactly a unique agent of A_1 and a unique agent of A_2 to each job in such a way that all jobs are assigned with respect to the capacities of the selected agents. Precisely, each pair of agents (k_1, k_2), k_1 in A_1 and k_2 in A_2 can be assigned to perform any job if k_1 and k_2 do not handle other jobs and the cost of the job is less than or equal to capacity of (k_1, k_2).

An instance $g = (J, d, w, C, a, b)$ of GA_{ss} can be defined as follows. J is a set

of n jobs, d is the vector of the costs of jobs, w is the matrix of the capacities of the agents and C, a and b are constant values used to compute the capacities. A set of n^2 pair of agents (k_1, k_2), k_1 in A_1 and k_2 in A_2 are available with capacities $w_{k_1,k_2} = C - a \times k_1 + b \times (k_2 - n - 1)$. A feasible solution of g is constituted of a set of n pairs of agents (k_1^j, k_2^j), j in J such that

$$k_m^i \neq k_m^j,\ \forall i, j \in J\ i \neq j; m \in \{1, 2\}.$$

and

$$w_{k_1^j,k_2^j} \geq d_j,\ \forall j \in J.$$

Let us first consider the following theorem.

Theorem 2.1. *$F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ and GA_{ss} are equivalent.*

Proof. First, let $I_{a,b} = (J, l)$ be an instance of $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ and C its possible makespan value, where J is a set of n jobs and l is the vector of time delays. We define an instance $g = (J, d, w, C, a, b)$ of GA_{ss} with n jobs, $d_j = l_j$, for all j in J and $w_{k_1,k_2} = C - a \times k_1 + b \times (k_2 - n - 1)$, k_1 and k_2 in $\{1, ..., n\}$. In this proof, we proceed as follows. We prove that the two instances g and $I_{a,b}$ are equivalent since the solution to one quickly leads to a solution to the other, and vice versa.

Let us denote by (k_1^j, k_2^j), j in J a set of n selected pairs of agents that presents a feasible solution to g. It is easy to observe that the solution obtained after scheduling each job $j \in J$ at the two positions k_1^j and k_2^j on M_1 and M_2, respectively is feasible for $I_{a,b}$ since $k_m^j \neq k_m^i$, for all i and j in J $i \neq j$; m in $\{1, 2\}$. Moreover, it holds that:

$$\begin{aligned} w_{k_1^j,k_2^j} = C - a \times k_1^j + b \times (k_2^j - n - 1) &\geq d_j,\ j \in J \\ &\geq l_j,\ j \in J. \end{aligned}$$

Thus, we obtain:

$$C + b \times (k_2^j - n - 1) \geq a \times k_1^j + l_j, \forall j \in J.$$

We observe that each job j of J is assigned to a unique position per machine and at least l_j time units separate the end of $O_{1,j}$ and the start of $O_{2,j}$. Therefore, we obtain a feasible schedule of $I_{a,b}$.

Let us consider now a feasible schedule $S = (\sigma, \tau)$ of $I_{a,b}$, where σ_j^{-1} (resp. τ_j^{-1}) stands for the position of job j on M_1 (resp. M_2) in S. It is obvious that the set of n pairs of agents $(\sigma_j^{-1}, \tau_j^{-1})$, j in J provides a solution with respect to the constraints of g since $\sigma_j^{-1} \neq \sigma_i^{-1}$ and $\tau_j^{-1} \neq \tau_i^{-1}$, for all i and j in J; $i \neq j$. Moreover, for each

valid schedule, a time delay must elapse between the completion time of a job on M_1 and its starting time on M_2. Thus, we obtain:

$$\begin{aligned} C - a \times \sigma_j^{-1} + b \times (\tau_j^{-1} - n - 1) &\geq l_j,\ j \in J \\ &\geq d_j,\ j \in J. \end{aligned}$$

We notice that each agent of A_1 (resp. A_2) is assigned to handle a unique job. We also observe that the cost of each job is less than or equal to the capacity of the selected pair of agents. Therefore, we obtain a feasible solution of g. □

2.3 Solution approach

In this section, we introduce a new compact mathematical formulation for GA_{ss}. Since GA_{ss} and $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ are equivalent, the formulation is described for the $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ problem. We also present a pre-processing procedure. Given an instance $I_{a,b} = (J, l)$ of $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$, we introduce the following notation:

- C: the possible makespan value of $I_{a,b}$.
- $d_{k_1,k_2} = C - a \times k_1 + b \times (k_2 - n - 1)$: the effective delay observed by a job if it is processed at position k_1 and k_2 on M_1 and M_2, respectively.
- $\bar{l}_1, \bar{l}_2, ..., \bar{l}_l, ..., \bar{l}_L$: the distinct values of time delays. Moreover, we suppose that $\bar{l}_1 > \bar{l}_2 > ... > \bar{l}_l > ... > \bar{l}_L$.
- $nb(\bar{l}_l)$: the occurrence number of $\bar{l}_l$ in the time delay vector.

2.3.1 ILP model

The ILP model is based on a non-trivial generalization of the assignment model. The decision variables are defined in order to determine a set of n couples of positions that are able to cover all time delay values without considering jobs. The decision variables are defined as follows:

$$z_{k_1,k_2} = \begin{cases} 1 \text{ if the couple of positions } (k_1, k_2) \text{ is selected, } \forall k_1, k_2 \in \{1, ..., n\}. \\ 0 \ \textit{Otherwise}. \end{cases}$$

The ILP model ILP_{ass} is given by:

$$\text{Find a set of } n \text{ couples of positions: } (k_1, k_2) \tag{2.1}$$

$$\text{s.t.} \quad \sum_{k_2=1}^{n} z_{k_1,k_2} = 1, \qquad \forall k_1 \in \{1, ..., n\} \tag{2.2}$$

$$\sum_{k_1=1}^{n} z_{k_1,k_2} = 1, \qquad \forall k_2 \in \{1, ..., n\} \tag{2.3}$$

$$\sum_{(k_1,k_2):d_{k_1,k_2} \geq \bar{l_l}} z_{k_1,k_2} \geq \sum_{\ell=1}^{l} nb(\bar{l_\ell}), \qquad \forall l \in \{1, ..., L\} \tag{2.4}$$

$$z_{k_1,k_2} \in \{0, 1\}, \qquad \forall k_1, k_2 \in \{1, ..., n\} \tag{2.5}$$

Constraints (2.2) (resp. (2.3)) ensure that each position on the first (resp. second) machine appears in a unique selected couple of positions. Constraints (2.4) guarantee that each time delay value is covered. Precisely, we verify that there exists at least a couple of positions for each time delay, such that its value is less than or equal to effective delay observed between the two positions. Finally, constraints (2.5) specify that the decision variables are boolean. The above model needs $O(n^2)$ variables and $O(n)$ constraints.

In order to reflect our optimization goal (minimizing the makespan), the resolution scheme that we apply for ILP_{ass} on an instance $I_{a,b}$ consists in invoking the ILP model on a set of feasibility test problems. Starting from a makespan value that we determine using the lower bounds of Section 3.2, we check out the existence of a feasible solution with the given makespan value. In case of feasibility, we stop the resolution of the instance. Otherwise, we repeat the same process after increasing the makespan value. This process ends if we reach the value $UB - 1$, where UB is an upper bound on the makespan value of instance $I_{a,b}$.

2.3.2 Pre-processing procedure

To improve the performance of the above ILP model, we introduce here a pre-processing procedure. The aim of this procedure is to reduce the number of decision variables and to discard partial solutions from additional expansions in order to limit the computational burden of the proposed method.

Definition 2.1. *Let $I_{a,b} = (J, l)$ be an instance of $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$. We define for each job j of J a new delay referred to as the critical delay $\tilde{l}_j$. Precisely, $\tilde{l}_j$ presents the maximum time delay that a job j could observe without affecting the existence of a feasible solution to $I_{a,b}$ with C as a makespan value.*

To compute $\tilde{l}_j$, we apply a lower bound of $F2|l_j|C_{max}$ on $I_{a,b}$ and we increment the value of l_j each time $LB \leq C$. The critical delay of job j is equal to $l_j - 1$ when $LB > C$.

The pre-processing procedure is given in the following proposition.

Proposition 2.1. *Let $I_{a,b} = (J, l)$ be an instance of $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$. Let (k_1, k_2) be a pair of positions such that $d_{k_1,k_2} > \tilde{l}_j$, k_1 and k_2 in $\{1, ..., n\}$. Then, job j cannot be scheduled at positions k_1 and k_2 on M_1 and M_2, respectively.*

Proof. Let (k_1, k_2) be a pair of positions, k_1 and k_2 in $\{1, ..., n\}$. If we consider a job j of J such that the critical delay of j is less than d_{k_1,k_2}, then job j cannot be scheduled at positions k_1 and k_2 on M_1 and M_2, respectively. Otherwise, there is no feasible solution for $I_{a,b}$ with C as a makespan value. □

Corollary 2.1. *Let j be a job and (k_1, k_2) a pair of positions, k_1 and k_2 in $\{1, ..., n\}$. If it holds that $d_{k_1,k_2} > \tilde{l}_j$, then each job i with $l_i \leq l_j$ cannot be scheduled at positions k_1 and k_2 on M_1 and M_2, respectively.*

Proof. It is sufficient to note that for each job i with $l_i \leq l_j$, it holds that $\tilde{l}_i \leq \tilde{l}_j$. □

Corollary 2.2. *Let j be a job such that for a pair of positions (k_1, k_2) it holds that $d_{k_1,k_2} > \tilde{l}_j$, k_1 and k_2 in $\{1, ..., n\}$. We define j^* as a job of J such that $j^* = arg\min_{l_{j'} > l_j}(l_{j'})$. If it holds that $d_{k_1,k_2} < l_{j'}$, then no job can be scheduled at positions k_1 and k_2 on M_1 and M_2, respectively.*

Proof. As a consequence of Proposition 2.1, it holds that job j and the jobs with a time delay less than or equal to l_j cannot be scheduled at positions k_1 and k_2 on M_1 and M_2, respectively. Moreover, since $d_{k_1,k_2} < l_{j'}$, the jobs with a time delay greater than or equal to $l_{j'}$ cannot be assigned to positions k_1 and k_2 on M_1 and M_2, respectively. Therefore, there is no job that can be scheduled at positions k_1 and k_2 on M_1 and M_2, respectively. □

Corollary 2.3. *Given an integer p in $\{1, ..., L\}$, we denote by $\hat{l}_p$ the critical delay that can be observed by a job with the time delay $\bar{l}_p$. Moreover, let (k_1, k_2) be a pair of positions such that $\hat{l}_p < d_{k_1,k_2} < \bar{l}_{p-1}$, k_1 and k_2 in $\{1, ..., n\}$. Then, the variable z_{k_1,k_2} should be eliminated from the model.*

Interestingly, we can reduce the computation burden of the pre-processing procedure as follows:

Remark 2.1. *If z_{k_1,k_2} should not be considered, then $z_{k_1+\ell,k_2+\ell}$ is also eliminated, $\forall \ell \in \{1, ..., n - \max(k_1, k_2)\}$.*

Remark 2.2. *If z_{k_1,k_2} should not be considered, and $\bar{l}_p < d_{k_1,k_2+1} < \bar{l}_{p-1}$, then z_{k_1,k_2+1} should be eliminated.*

Remark 2.3. *If z_{k_1,k_2} should not be considered, then we can save the result of the distance between the two positions (k_1, k_2) (i.e., $k_2 - k_1$) and the integer p, so that we can avoid the computation of the lower bound for the same values when other positions are considered.*

2.4 Computational results

In this section, we report the results of a computational study that aims at assessing the performance of the proposed ILP model. In fact, we only study the $F2|p_{i,j} = 1, l_j|C_{max}$ problem since a dedicated exact method exists for the problem. At first, we introduce two classic ILP models for comparison reasons. We then study the impact of the pre-processing procedure on the performance of the three ILP models. Moreover, we present the results of two comparisons in which we analyze the influence of the problem size and the time delay distribution on the proposed ILP model and the state of the art exact methods.

At first, we recall here two position variable-based models for $F2|p_{i,j} = 1, l_j|C_{max}$. These two ILP models are developed to solve a decision problem. Given a possible makespan value C, we check out the existence of a feasible solution with that makespan value. The first ILP model is based on determining separately the position where a job $j \in J$ is scheduled on the first machine and also its position on the second machine. Let us define the following decision variables:

$$x^k_{m,j} = \begin{cases} 1 \text{ if operation } O_{m,j} \text{ is processed at position } k \text{ on machine } M_m,\ \forall m \in \{1,2\}, \\ \quad j \in J; k \in \{1, ..., n\}. \\ 0\ Otherwise. \end{cases}$$

The ILP1 is given by:

$$\text{Find a feasible schedule} \tag{2.6}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} x^k_{m,j} = 1, \qquad \forall m \in \{1,2\},\ j \in J \tag{2.7}$$

$$\sum_{j=1}^{n} x^k_{m,j} = 1, \qquad \forall m \in \{1,2\},\ k \in \{1, ..., n\} \tag{2.8}$$

$$\sum_{k=1}^{n} k.x^k_{2,j} - \sum_{k'=1}^{n} k'.x^{k'}_{1,j} \geq l_j + 1 + n - C, \quad \forall j \in J \tag{2.9}$$

$$x^k_{m,j} \in \{0,1\}, \qquad \forall j \in J, m \in \{1,2\}; k \in \{1, ..., n\} \tag{2.10}$$

Constraints (2.7) ensure that each job $j \in J$ has to be processed at only one position

per machine. Constraints (2.8) specify that, at each position on M_1 and M_2, only one operation can be processed. Constraints (2.9) ensure that each job $j \in J$ has to wait at least its time delay between its completion time on the first machine and its starting time on the second. Finally, constraints (2.10) specify that the decision variables are boolean. This formulation needs $O(n^2)$ variables and $O(n)$ constraints.

The second ILP model is based on decision variables that determine jointly, on both machines, the positions of a job $j \in J$. For the seek of clarity, let us define the following two subsets:

- $E^1(j,k) = \{k' : C - n + k - k' - 1 \geq l_j\}$.
- $E^2(j,k) = \{k' : C - n + k' - k - 1 \geq l_j\}$.

$E^1(j,k)$ (resp. $E^2(j,k)$) is the set of the different allowed positions where job $j \in J$ can be processed on the first (resp. second) machine if it is scheduled at position k on the second (resp. first) machine.

Let us define the following decision variables:

$$y^j_{k,k'} = \begin{cases} 1 \text{ if operation } O_{1,j} \text{ and operation } O_{2,j} \text{ are processed at position } k \text{ and } k' \\ \quad \text{on } M_1 \text{ and } M_2, \text{ respectively, } \forall k, k' \in \{1,...,n\}, j \in J. \\ 0 \ Otherwise. \end{cases}$$

The ILP2 is given by:

$$\text{Find a feasible schedule} \tag{2.11}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} \sum_{k' \in E^2(j,k)} y^j_{k,k'} = 1, \quad \forall j \in J \tag{2.12}$$

$$\sum_{j=1}^{n} \sum_{k' \in E^2(j,k)} y^j_{k,k'} = 1, \quad \forall k \in \{1,...,n\} \tag{2.13}$$

$$\sum_{j=1}^{n} \sum_{k \in E^1(j,k')} y^j_{k,k'} = 1, \quad \forall k' \in \{1,...,n\} \tag{2.14}$$

$$y^j_{k,k'} \in \{0,1\}, \qquad \forall j \in J, \ k \in \{1,...,n\}; \ k' \in E^2(j,k) \tag{2.15}$$

Constraints (2.12) ensure that each job $j \in J$ is processed at a unique position per machine. Constraints (2.13) (resp. (2.14)) specify that at a given position on the first (resp. second) machine only one operation can be processed. Finally, constraints (2.15) specify that the decision variables are boolean. The above model needs $O(n^3)$ variables and $O(n)$ constraints.

The following computational study is carried out between four exact methods:

- ILP_{ass}: the proposed destructive exact method for $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$. We use the value of the lower bound of [Yu et al., 2004] as an intial solution.

- $ILP1$, $ILP2$: the exact resolution of the proposed ILP models using CPLEX 12.6. In order to reflect our optimization goal (minimizing the makespan), the resolution scheme that we apply for each ILP model on an instance I consists in invoking the ILP model on a set of feasibility test problems. Starting from a makespan value that we determine using the lower bound of [Yu et al., 2004], we check out the existence of a feasible solution with the given makespan value. In case of feasibility, we stop the resolution of the instance. Otherwise, we repeat the same process after increasing the makespan value. This process ends if we reach the value $UB - 1$, where UB is an upper bound.

- $B\&B_M$: the branch-and-bound method of [Moukrim et al., 2014].

To get a better picture of the performance of the exact methods, we provide the following measures:

- USI: the number of unsolved instances.

- Time: the average computational time.

All the discussed ILP models were coded in C++ and compiled under CentOS 6.6. Moreover, we used CPLEX 12.6 to implement them. The experiments were conducted on an Intel(R) Xeon(R) @ 2.67GHz processor where for each invoked method, we set a time limit of 600 seconds.

The data set that we used in the experiments is composed from 1560 instances that were introduced by [Moukrim et al., 2014]. Precisely, 13 classes of instances were considered where the time delays were randomly generated between $[0..\lceil \frac{n}{r} \rceil]$, with $r \in \{\frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{2}{3}, 1, \frac{3}{2}, 2, \frac{5}{2}, 3, \frac{7}{2}, 4, 8, 16\}$. For each class, the different tested sizes are $n = 20, 40, 60, 80, 100, 120, 150, 160, 180, 200, 250$ and 300. For each combination of class and number of jobs, 10 instances were randomly generated.

2.4.1 The impact of the pre-processing procedure

In order to get a detailed image of the impact of the pre-processing technique, we provide in Table 2.1 a comparison of performance between the three ILP models. For each model, two versions were considered: the exact resolution of the model

itself and the exact resolution of the model with the consideration of the processing procedure.

Remark 2.4. *The pre-processing procedure is applied on ILP1 and ILP2 in the following way. Let j be a job and (k_1,k_2) a couple of positions with respect to Proposition 2.1 conditions, k_1 and k_2 in $\{1, ..., n\}$. Then, it holds that:*

- $x_{1,j}^{k_1} + x_{2,j}^{k_2} \leq 1$ *is a valid inequality for ILP1.*
- *The decision variable y_{k_1,k_2}^{j} should be eliminated from ILP2.*

Table 2.1 – Impact of the pre-processing procedure.

$ILP1$		$ILP1_{pre}$		$ILP2$		$ILP2_{pre}$		ILP_{ass}		$ILP_{ass,pre}$	
USI	**Time**	**USI**	**Time**	**USI**	**Time**	**USI**	**Time**	**USI**	**Time**	**USI**	**Time**
1034	414.15	1017	412.77	820	354.72	710	311.35	72	52.38	41	41.88

From Table 2.1, we notice that:

- An improvement of performance is observed for all ILP models when we incorporate the pre-processing method. Especially for ILP_{ass} and ILP2, the total number of unsolved instances decreases from 72 (resp. 820) to 41 (resp. 710).
- ILP_{ass} provides a much better performance using the pre-processing procedure on big size instances where the number of unsolved instances passes from 44 (resp. 24) to 28 (resp. 12) for $n = 300$ (resp. $n = 250$).
- For ILP1, we observe an improvement of performance since 17 new instances were solved. Precisely, the evolution of the number of unsolved instances for ILP1 significantly depends on the problem size. On one hand, ILP1 performs better after solving 26 new instances including 17, three and one when $n = 60$, 80 and 150, respectively. However, nine instances are not solved anymore for $n = 40$, 100, 160 and 300.

2.4.2 The impact of the problem size

We study here the impact of the problem size on the performance of the early discussed methods. We provide in Figure 2.1 (resp. Figure 2.2) the evolution of the number of unsolved instances (resp. the evolution of the average computational time) in terms of n.

From Figure 2.1 and Figure 2.2, we make the following observations:

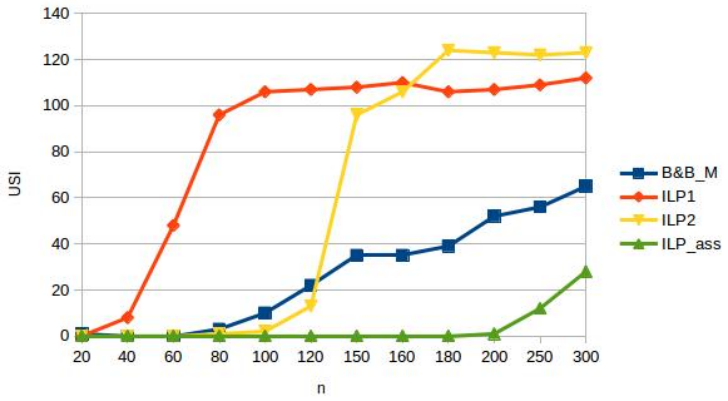


Figure 2.1 – Total unsolved instances as a function of n.

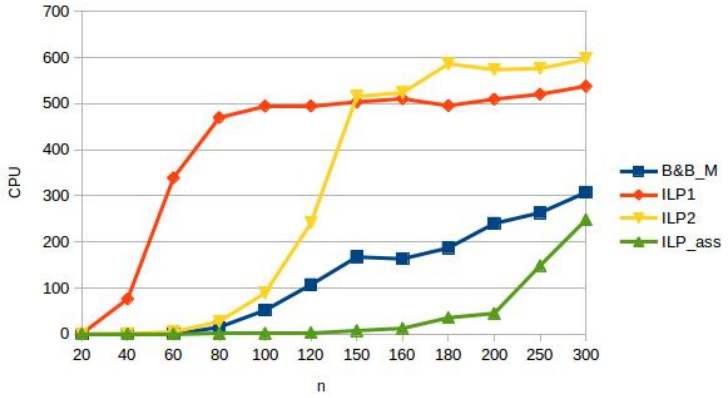


Figure 2.2 – The average CPU time as a function of n.

- ILP_{ass} provides the best results while requiring a short time. Precisely, ILP_{ass} solves all instances for $n \leq 180$ and only one, 12 and 28 instances for $n = 200, 250$ and 300, respectively.

- $B\&B_M$ gets competitive results in a good time. In fact, it got similar results to ILP_{ass} when $n \leq 80$ where it fails to solve three instances comparing to zero unsolved instances for ILP_{ass}. Then, for $n \geq 100$, its performance decreases. Actually, the total number of unsolved instances for $n \geq 100$ is equal to 314 instances out of 1040 possible ones.

- $ILP1$ and $ILP2$ yield the worst performance. In fact, their performances pass with three phases. In the first phase, they yield a good performance where the number of unsolved instances is equal to eight (resp. 16) when $n \leq 40$ (resp. $n \leq 120$) for $ILP1$ (resp. $ILP2$). In the second one, the number of solved instances decreases rapidly for $ILP1$ (resp. $ILP2$) when $40 \leq n \leq 100$

(resp.$120 \leq n \leq 180$). Finally, a partial stabilization of performance is seen when $n \geq 100$ (resp. $n \geq 180$) where $106 \leq USI \leq 112$ (resp. $122 \leq USI \leq 124$) for $ILP1$ (resp. $ILP2$).

A detailed version of the results is given in Table 2.2.

Table 2.2 – Impact of the problem size.

n	$B\&B_M$			$ILP1$			$ILP2$			ILP_{ass}		
	USI	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**
20	0	18.73	0.00	0	106.69	0.43	0	0.00	0.07	0	0.00	0.02
40	0	131.84	0.00	8	15536.54	76.11	0	0.00	0.88	0	0.00	0.09
60	0	30404.93	0.06	48	32323.03	339.10	0	0.50	4.55	0	0.00	0.29
80	3	7535758.50	14.91	96	12381.48	469.75	1	1.85	27.30	0	0.00	0.77
100	10	20608064.00	51.71	106	3130.37	494.72	2	2.76	89.08	0	0.00	1.69
120	22	35646528.00	107.07	107	1921.88	495.01	13	1.11	241.32	0	0.03	3.18
150	35	50516908.00	167.28	108	1159.34	503.80	96	2.34	515.73	0	0.25	7.43
160	35	43327028.00	163.20	110	934.18	510.55	106	0.01	523.59	0	2.13	12.39
180	39	44737948.00	186.88	106	320.15	495.62	124	0.62	586.40	0	1.88	35.38
200	52	54876388.00	240.04	107	153.77	509.79	123	7.71	574.00	1	3.12	44.82
250	56	49176792.00	263.26	109	48.12	520.61	122	0.07	576.32	12	0.56	148.47
300	65	45889228.00	307.56	112	20.01	537.75	123	0.00	596.99	28	0.01	248.03

2.4.3 The influence of the time delay distribution

We provide here a comparison of performance between the three ILP models and the literature exact method in terms of the parameter r. Recall that the time delays were randomly generated between $[0..\lceil \frac{n}{r} \rceil]$. Therefore, the size of the time delay interval depends on the parameter r directly. If $r \leq 1$, then the interval is large and the possibilities to have different time delay values are high. Otherwise when $r > 1$, the time delay interval becomes tight and the number of redundant jobs is high. Therefore, when we increase the value of parameter r, the number of jobs with the same time delay will increase to.

We present in Figure 2.3 (resp. Figure 2.4) the evolution of the number of unsolved instances (resp. the evolution of the average computational time) in terms of r.

The findings from the computations in Figure 2.3 and Figure 2.4 are summarized below:

- We note that the performance of all methods significantly depends on the value of r.
- Globally, ILP_{ass} exhibits the best performance in a short time on all classes, except for $r = 2/3$ where $B\&B_M$ slightly outperforms it by solving five more instances.

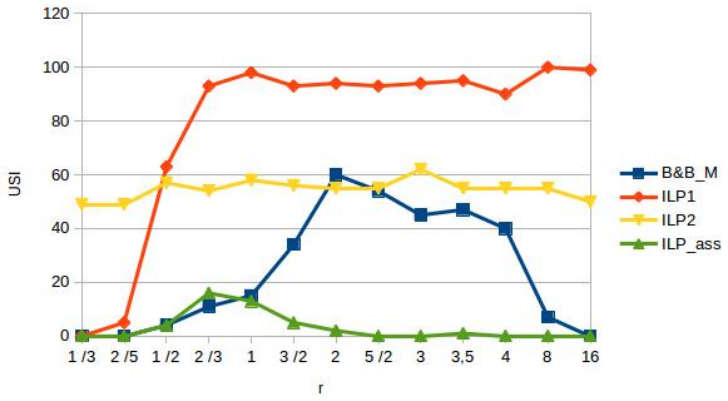


Figure 2.3 – Total unsolved instances as a function of r.

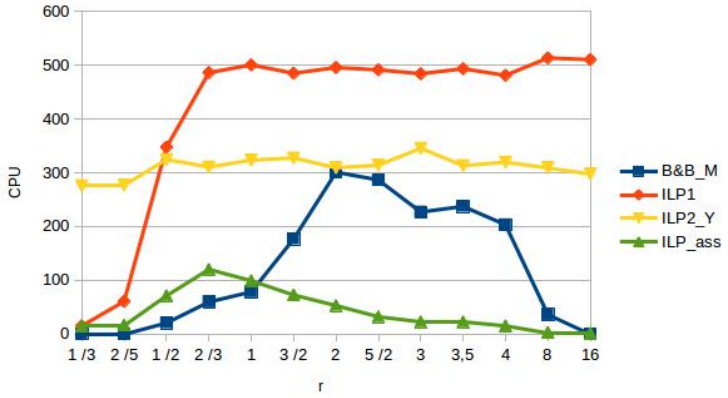


Figure 2.4 – The average CPU time as a function of r.

- Interestingly, we remark that the instances become harder to solve for all methods when $r \geq 2/5$. However, the difference between these methods resides in the rapidity of recovery. The number of unsolved instances increases rapidly when $2/5 \leq r \leq 2/3$ (resp. $2/5 \leq r \leq 2$) for ILP_{ass} (resp. $B\&B_M$). After that, ILP_{ass} gains its effectiveness and manages to improve the results, where it only fails to solve one instance when $r \geq 5/2$. The same behaviour is observed by $B\&B_M$ when $r \geq 8$. For $ILP2$, it yields a constant performance where the number of unsolved instances varies between 49 and 62 for all values of r. $ILP1$ yields the worst performance where the number of unsolved instances and the CPU time increase rapidly when $2/5 \leq r \leq 1$. For $r \geq 1$, we observe a partial stabilization of performance where $93 \leq \text{USI} \leq 100$.

A detailed version of the results is given in Table 2.3.

2.5 Conclusion

In this chapter, we addressed the two-machine flow-shop problem with time delays and identical processing times per machine. We presented a new mathematical approach to solve the problem. For which, we integrated a pre-processing procedure to improve its performance. The conducted experiments show that the exact resolution of the proposed ILP model with the consideration of the pre-processing procedure outperform the literature exact method on the $F2|p_{i,j} = 1, l_j|C_{max}$ instances.

Table 2.3 – Performance by class.

		$B\&B_M$			$ILP1$			$ILP2$			ILP_{ass}	
Class	**n**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**
$r = 1/3$	20	0	9.40	0.00	0	0.00	0.03	0	0.00	0.05	0	0.00	0.01
	40	0	62.40	0.00	0	39.90	0.47	0	0.00	0.58	0	0.00	0.07
	60	0	113.50	0.00	0	0.00	0.55	0	0.00	5.49	0	0.00	0.25
	80	0	235.40	0.00	0	35.00	1.60	0	0.00	23.48	0	0.00	0.62
	100	0	357.80	0.00	0	94.10	4.23	1	0.00	159.87	0	0.00	1.54
	120	0	412.90	0.00	0	0.00	2.96	4	0.00	391.87	0	0.00	2.46
	150	0	885.70	0.00	0	112.20	11.78	7	0.00	424.69	0	0.00	5.93
	160	0	1063.30	0.00	0	6.80	9.06	7	0.00	424.99	0	0.00	7.41
	180	0	987.70	0.01	0	3.00	11.07	10	0.00	603.67	0	0.00	10.77
	200	0	1286.00	0.01	0	3.00	17.34	8	0.00	487.78	0	0.00	14.69
	250	0	2233.30	0.02	0	21.90	38.47	7	0.00	440.39	0	0.00	48.35
	300	0	3521.40	0.04	0	15.50	84.94	5	0.00	347.50	0	0.00	89.65
$r = 2/5$	20	0	17.50	0.00	0	9.00	0.04	0	0.00	0.05	0	0.00	0.01
	40	0	60.90	0.00	0	1.00	0.32	0	0.00	0.70	0	0.00	0.07
	60	0	136.40	0.00	0	0.00	0.69	0	0.00	3.73	0	0.00	0.25
	80	0	467.80	0.00	0	48.00	2.89	0	0.00	22.63	0	0.00	0.83
	100	0	643.40	0.00	0	2061.20	63.91	0	0.00	181.33	0	0.00	1.64
	120	0	1147.20	0.00	1	269.60	67.73	4	0.00	333.77	0	0.00	3.21
	150	0	1452.80	0.01	1	160.40	71.11	9	0.00	542.91	0	3.00	6.69
	160	0	1718.40	0.01	1	515.60	81.06	5	0.00	305.35	0	0.00	8.26
	180	0	2117.80	0.01	0	77.80	28.47	9	0.00	543.24	0	0.00	12.32
	200	0	2956.90	0.02	0	209.50	86.15	8	0.00	486.04	0	0.00	20.80
	250	0	3885.40	0.03	0	150.30	90.93	5	0.00	321.51	0	0.00	44.95
	300	0	6314.40	0.06	2	235.10	236.54	9	0.00	578.42	0	0.00	99.67
$r = 1/2$	20	0	41.30	0.00	0	4.60	0.17	0	0.00	0.08	0	0.00	0.01
	40	0	172.60	0.00	0	315.20	2.19	0	0.00	1.74	0	0.00	0.11
	60	0	455.40	0.00	1	7465.60	117.23	0	6.50	6.41	0	0.00	0.45
	80	1	38014544.00	60.00	2	2896.80	188.87	0	0.00	20.92	0	0.00	1.22
	100	1	38281052.00	60.01	6	4742.20	369.91	0	0.00	70.14	0	0.00	3.27
	120	0	2234.20	0.01	6	1881.70	368.99	1	13.10	274.34	0	0.40	5.70
	150	0	51761.20	0.10	7	1472.50	467.68	6	0.00	469.69	0	0.20	15.82
	160	0	2070934.38	5.26	9	2080.00	548.03	10	0.00	600.92	0	0.00	12.36
	180	0	5572.70	0.02	6	897.70	397.01	10	0.00	602.32	0	0.00	37.66
	200	0	6535.00	0.03	7	692.20	511.05	10	52.20	604.48	0	0.00	83.48
	250	0	144080.20	0.65	9	447.20	599.48	10	0.00	618.62	1	1.50	294.29
	300	2	32318742.00	120.22	10	9.50	602.50	10	0.00	626.89	3	0.00	400.90
$r = 2/3$	20	0	21.70	0.00	0	33.90	0.27	0	0.00	0.07	0	0.00	0.02
	40	0	227.80	0.00	0	8152.80	64.05	0	0.00	0.52	0	0.00	0.17
	60	0	60108.60	0.09	3	38465.10	358.57	0	0.00	4.91	0	0.00	0.61
	80	0	54577.20	0.09	10	9219.30	599.43	0	8.00	17.22	0	0.00	1.83
	100	1	36692584.00	60.01	10	3841.70	599.20	0	19.90	107.14	0	0.00	3.45
	120	0	34514.80	0.08	10	2512.40	599.33	0	1.00	153.47	0	0.00	7.97
	150	1	25859770.00	60.04	10	1624.50	600.31	6	22.00	484.88	0	0.00	16.97
	160	0	6102.30	0.03	10	1851.00	600.10	9	0.00	581.97	0	0.00	31.84

Table 2.3 - Continued

			$B\&B_M$			$ILP1$			$ILP2$			ILP_{ass}	
Class	**n**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**
	180	1	14685514.00	61.44	10	616.60	600.59	10	0.00	599.86	0	2.00	177.01
	200	4	65784672.00	240.15	10	73.80	601.19	9	22.00	544.86	0	1.30	128.99
	250	2	31849734.00	120.28	10	0.00	605.22	10	0.00	609.07	6	0.00	472.82
	300	2	34235160.00	173.96	10	0.00	608.39	10	0.00	624.03	10	0.00	601.50
$r = 1$	20	0	9.70	0.00	0	47.40	0.37	0	0.00	0.07	0	0.00	0.03
	40	0	269.90	0.00	0	16298.90	84.96	0	0.00	0.77	0	0.00	0.13
	60	0	250.00	0.00	8	68079.50	512.02	0	0.00	4.40	0	0.00	0.50
	80	0	392.90	0.00	10	19545.70	599.37	0	7.00	40.75	0	0.00	1.36
	100	0	1282.20	0.01	10	3276.90	599.57	0	13.00	65.00	0	0.00	3.12
	120	1	31017316.00	60.01	10	2617.90	599.76	1	0.20	201.97	0	0.00	6.22
	150	3	63845940.00	195.00	10	1372.10	600.17	9	8.40	590.16	0	0.00	15.07
	160	2	26668164.00	120.01	10	1460.20	599.80	8	0.00	541.56	0	0.00	48.72
	180	1	15450182.00	60.03	10	655.00	599.87	10	0.00	601.09	0	2.40	104.12
	200	2	26135882.00	120.08	10	455.00	600.39	10	0.00	602.27	1	0.80	111.80
	250	4	55124960.00	241.28	10	0.00	604.35	10	0.00	614.90	4	0.00	376.30
	300	2	20362310.00	144.61	10	0.00	605.93	10	0.00	620.57	8	0.00	523.06
$r = 3/2$	20	0	10.50	0.00	0	181.90	0.70	0	0.00	0.07	0	0.00	0.02
	40	0	43.80	0.00	1	11143.10	79.66	0	0.00	1.12	0	0.00	0.11
	60	0	2280.50	0.00	3	38954.70	360.69	0	0.00	5.10	0	0.00	0.38
	80	0	245.40	0.00	9	17445.30	572.42	0	0.00	20.42	0	0.00	0.97
	100	0	275803.69	0.60	10	3413.40	599.42	0	0.00	70.61	0	0.00	2.02
	120	3	73684520.00	180.00	10	2694.90	599.56	1	0.10	363.89	0	0.00	4.04
	150	7	154560112.00	420.00	10	1041.90	599.93	7	0.00	500.07	0	0.00	9.51
	160	4	78485168.00	240.22	10	288.70	599.77	8	0.00	523.19	0	0.00	14.25
	180	4	71497072.00	243.29	10	154.10	601.36	10	8.00	602.79	0	20.00	67.11
	200	7	117519720.00	420.01	10	112.50	601.73	10	0.00	602.93	0	0.00	107.58
	250	4	66774644.00	299.76	10	0.00	602.15	10	0.90	610.54	1	0.60	206.20
	300	5	53465660.00	318.98	10	0.00	604.39	10	0.00	629.36	4	0.00	457.97
$r = 2$	20	0	20.00	0.00	0	143.60	0.57	0	0.00	0.07	0	0.00	0.02
	40	0	47.30	0.00	0	11941.20	61.90	0	0.00	0.70	0	0.00	0.09
	60	0	4580.70	0.01	5	24038.30	475.02	0	0.00	3.22	0	0.00	0.28
	80	0	3647780.00	8.68	9	21978.70	595.10	0	0.00	25.36	0	0.00	0.74
	100	3	60747032.00	184.56	10	3049.30	599.43	0	0.00	68.25	0	0.00	1.60
	120	7	136552368.00	420.00	10	2180.80	599.65	0	0.00	199.54	0	0.00	3.09
	150	7	128338560.00	420.20	10	870.80	600.31	6	0.00	410.43	0	0.00	7.49
	160	6	112755312.00	360.06	10	1281.20	599.76	10	0.00	601.37	0	0.00	8.64
	180	7	116000624.00	420.66	10	237.70	599.91	10	0.00	601.14	0	0.00	13.95
	200	10	152842640.00	600.00	10	69.20	601.81	9	0.00	562.28	0	0.00	39.52
	250	10	125323984.00	600.00	10	3.00	607.25	10	0.00	610.25	0	5.00	204.00
	300	10	94691544.00	600.00	10	0.00	607.92	10	0.00	630.60	2	0.10	355.96

Table 2.3 - Continued

		$B\&B_M$			$ILP1$			$ILP2$			ILP_{ass}	
Class	**n**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**
$r = 5/2$	20	0	9.90	0.00	0	226.90	0.77	0	0.00	0.08	0	0.00	0.02
	40	0	111.20	0.00	0	15487.00	72.74	0	0.00	1.00	0	0.00	0.08
	60	0	324996.69	0.66	4	59882.50	431.98	0	0.00	5.44	0	0.00	0.24
	80	1	26107180.00	60.86	9	15014.40	582.29	0	0.00	21.83	0	0.00	0.60
	100	2	65846484.00	185.43	10	2591.10	599.51	0	0.00	44.18	0	0.00	1.31
	120	4	77985160.00	266.73	10	2134.60	599.74	0	0.00	195.48	0	0.00	2.43
	150	6	103495216.00	419.26	10	1493.20	599.91	7	0.00	548.68	0	0.00	5.45
	160	7	104424592.00	420.04	10	1072.70	600.06	9	0.10	555.49	0	27.70	12.80
	180	7	102583232.00	466.75	10	246.60	600.61	10	0.00	603.72	0	0.00	11.24
	200	7	90252784.00	420.08	10	50.50	601.90	9	26.00	552.44	0	38.40	35.61
	250	10	102675896.00	600.00	10	2.90	604.27	10	0.00	610.07	0	0.00	127.10
	300	10	76597360.00	600.00	10	0.00	603.92	10	0.00	630.16	0	0.00	188.54
$r = 3$	20	0	19.50	0.00	0	208.00	0.72	0	0.00	0.09	0	0.00	0.02
	40	0	84.90	0.00	0	3226.30	28.07	0	0.00	0.92	0	0.00	0.08
	60	0	1491.70	0.00	4	50367.30	374.74	0	0.00	4.93	0	0.00	0.24
	80	0	8317.10	0.02	10	22905.10	599.01	1	5.40	76.87	0	0.00	0.53
	100	1	20997100.00	61.55	10	3980.40	599.32	1	3.00	124.84	0	0.00	1.12
	120	0	5070682.00	15.89	10	2607.60	599.44	2	0.00	332.05	0	0.00	2.08
	150	4	64778904.00	240.01	10	1292.20	599.40	8	0.00	556.42	0	0.00	4.32
	160	6	85876400.00	365.62	10	1082.00	599.39	10	0.00	602.12	0	0.00	5.39
	180	6	80780360.00	360.02	10	173.90	600.98	10	0.00	602.59	0	0.00	9.48
	200	8	91941744.00	480.02	10	63.80	602.96	10	0.00	603.11	0	0.00	16.34
	250	10	91057376.00	600.00	10	0.00	603.10	10	0.00	609.77	0	0.00	65.04
	300	10	87932392.00	600.00	10	0.00	604.08	10	0.00	633.68	0	0.00	166.03
$r = 3.5$	20	0	13.40	0.00	0	97.80	0.39	0	0.00	0.07	0	0.00	0.01
	40	0	278.80	0.00	0	6694.60	40.90	0	0.00	0.88	0	0.00	0.07
	60	0	166.80	0.00	5	45241.70	476.59	0	0.00	3.89	0	0.00	0.21
	80	1	28328522.00	60.00	10	14100.00	599.45	0	0.00	17.23	0	0.00	0.45
	100	2	45024280.00	120.00	10	3389.50	599.37	0	0.00	71.42	0	0.00	1.08
	120	5	102619968.00	329.15	10	1839.10	599.35	0	0.00	148.80	0	0.00	1.61
	150	4	67424000.00	240.01	10	1424.30	599.54	8	0.00	571.24	0	0.00	3.61
	160	6	84042928.00	360.01	10	400.20	600.18	8	0.00	508.87	0	0.00	4.55
	180	7	89209688.00	420.01	10	158.70	601.75	9	0.00	590.34	0	0.00	6.60
	200	4	48943368.00	240.04	10	28.00	600.72	10	0.00	604.30	0	0.00	9.40
	250	8	78488368.00	480.01	10	0.20	601.02	10	0.00	614.01	0	0.20	56.38
	300	10	80311744.00	600.00	10	0.00	602.98	10	0.00	628.15	1	0.00	187.04
$r = 4$	20	0	18.90	0.00	0	75.50	0.41	0	0.00	0.08	0	0.00	0.01
	40	0	88.70	0.00	0	4289.60	48.43	0	0.00	0.94	0	0.00	0.06
	60	0	180.70	0.00	3	27487.70	341.55	0	0.00	4.93	0	0.00	0.19
	80	0	1801581.25	4.13	7	14014.80	567.52	0	3.60	34.78	0	0.00	0.43
	100	0	36699.00	0.10	10	3188.10	598.91	0	0.00	65.02	0	0.00	0.86
	120	2	36434596.00	120.01	10	2371.40	599.49	0	0.00	244.58	0	0.00	1.43
	150	3	48357520.00	180.01	10	1505.40	599.63	10	0.00	600.55	0	0.00	3.25
	160	4	66386732.00	240.01	10	840.60	599.78	8	0.00	547.79	0	0.00	4.11

Table 2.3 - Continued

			$B\&B_M$			$ILP1$			$ILP2$			ILP_{ass}	
Class	**n**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**	**USI**	**Node**	**Time**
	180	6	90663208.00	393.81	10	302.20	600.04	8	0.00	542.88	0	0.00	5.39
	200	9	107839360.00	540.00	10	20.90	601.13	10	0.00	603.32	0	0.00	8.26
	250	6	66231212.00	360.03	10	0.00	603.89	10	0.00	614.14	0	0.00	23.43
	300	10	82813968.00	600.00	10	0.00	612.72	9	0.00	577.59	0	0.00	134.52
$r = 8$	20	0	33.70	0.00	0	291.90	0.73	0	0.00	0.08	0	0.00	0.01
	40	0	134.70	0.00	4	50411.70	267.68	0	0.00	0.83	0	0.00	0.04
	60	0	247.70	0.00	6	25767.60	483.00	0	0.00	3.06	0	0.00	0.09
	80	0	459.90	0.00	10	14890.40	599.36	0	0.00	16.73	0	0.00	0.25
	100	0	680.00	0.01	10	3852.00	599.17	0	0.00	82.48	0	0.00	0.66
	120	0	712.70	0.01	10	2037.60	599.69	0	0.00	162.56	0	0.00	0.68
	150	0	2334.80	0.02	10	1541.30	599.82	8	0.00	536.93	0	0.00	1.58
	160	0	2530750.75	10.27	10	554.20	600.17	7	0.00	480.21	0	0.00	1.67
	180	0	694541.12	3.25	10	479.00	600.18	10	0.00	601.19	0	0.00	2.88
	200	1	12119745.00	60.04	10	166.00	600.84	10	0.00	602.76	0	0.00	4.03
	250	2	21618662.00	120.20	10	0.00	604.49	10	0.00	606.89	0	0.00	7.73
	300	4	33785748.00	240.10	10	0.00	607.98	10	0.00	613.35	0	0.00	13.69
$r = 16$	20	0	18.00	0.00	0	66.50	0.50	0	0.00	0.07	0	0.00	0.01
	40	0	130.90	0.00	3	73973.70	238.09	0	0.00	0.68	0	0.00	0.03
	60	0	255.40	0.00	6	34449.40	475.73	0	0.00	3.56	0	0.00	0.06
	80	0	553.80	0.00	10	8865.80	599.38	0	0.00	16.75	0	0.00	0.14
	100	0	754.90	0.00	10	3214.90	599.41	0	0.00	47.71	0	0.00	0.28
	120	0	1040.30	0.01	10	1836.90	599.41	0	0.00	134.81	0	0.00	0.47
	150	0	3265.50	0.02	10	1160.60	599.77	5	0.00	467.90	0	0.00	0.94
	160	0	1741.70	0.02	10	711.20	599.99	7	0.00	532.80	0	0.00	1.08
	180	0	20246.60	0.12	10	159.60	601.27	8	0.00	528.35	0	0.00	1.38
	200	0	2268.90	0.04	10	54.60	600.08	10	0.00	605.45	0	0.00	2.12
	250	0	3447.30	0.08	10	0.00	603.28	10	0.00	611.97	0	0.00	3.55
	300	0	35653.00	0.36	10	0.00	608.43	10	0.00	620.59	0	0.00	5.91

Lower bounds for $F2|l_j|C_{max}$

Contents

3.1 Introduction

In this chapter, we are interested in the two-machine flow-shop problem with time delays with respect to the makespan. Compared the problem studied in Chapter 2,

the processing times are here job-dependent and thus multiple processing values can be observed per machine.

The main contributions of this chapter consist in:

- In term of theoretical approaches, we provide theoretical and experimental analyses of the existing lower bounds. Precisely, we propose a new classification scheme of the different lower bounds and we enhance a time delay-based one. Then, we elucidate dominance relationships between them.

- In term of resolution approaches, we discuss some key challenges and limitations of considering MIP-based procedures in order to solve the two-machine flow-shop problem with time delays. In particular, we point out the performance of a linear ordering variable-based model. Moreover, we derive three relaxation schemes for $F2|l_j|C_{max}$. In the first scheme, we solve the linear relaxation of the linear ordering variable-based model. This lower bound includes the consideration of clique-based valid inequalities. These inequalities consist in computing optimal solutions of polynomial-time-solvable sub-instances, which are extracted from the original instance by computing all maximal cliques on a particular *Interval graph*. The second and third scheme are based on relaxing the $F2|l_j|C_{max}$ to the $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ problem and to a particular assignment problem, respectively.

- In terms of computational results, extensive experiments carried out on the different classes of instances, demonstrate the efficiency of the proposed lower bounds. The lower bounds outperform the state of the art ones. In particular, the linear relaxation of the linear ordering variable-based model together with the consideration of the clique-based inequalities yields the best lower bound value on all instances compared to the literature lower bounds.

The remainder of this chapter is organized as follows. In Section 3.2, we present a new classification scheme of the state of the art lower bounds and we enhance a time delay-based one. We also introduce dominance relationships between them. In Section 3.3, we recall a set of MIP formulations for $F2|l_j|C_{max}$ problem. We then introduce linear programming-based lower bounds for $F2|l_j|C_{max}$ in Section 3.4. Computational experiments on a set of randomly generated instances are reported in Section 3.5. Finally, some concluding remarks are given in Section 3.6.

3.2 Combinatorial lower bounds for $F2|l_j|C_{max}$

Many works investigated lower bounds for $F2|l_j|C_{max}$. A survey on the $F2|l_j|C_{max}$ problem was done by [Emmons and Vairaktarakis, 2013], in which lower bound methods were recalled. However, the major contribution of [Yu, 1996] was omitted. The aim of this section is to classify the state of the art lower bounds according to their relaxation scheme. Then, dominance relationships are elucidated between them.

3.2.1 Lower bounds

To the best of our knowledge, the only contributions for $F2|l_j|C_{max}$ lower bounds are those of [Yu, 1996] and [Dell'Amico, 1996]. The proposed lower bounds are based on three relaxation schemes, namely the resource constraint-based relaxation scheme, the time delay-based relaxation scheme and the job splitting-based relaxation scheme.

Basic bounds. To begin with, we recall two $O(n)$ basic lower bounds. The first lower bound $LB_{bas,0}$ was proposed by [Dell'Amico, 1996] and is given by:

$$LB_{bas,0} = \max_{1\leq j\leq n}(p_{1,j} + l_j + p_{2,j}). \tag{3.1}$$

The second lower bound $LB_{bas,1}$ was introduced by [Yu, 1996] and is given by expression (3.2). Note that [Dell'Amico, 1996] proposed a similar lower bound that is dominated by $LB_{bas,1}$.

$$LB_{bas,1} = \max(\sum_{j=1}^{n} p_{1,j} + \min_{1\leq j\leq n}(l_j + p_{2,j}), \sum_{j=1}^{n} p_{2,j} + \min_{1\leq j\leq n}(l_j + p_{1,j})). \tag{3.2}$$

Interestingly, [Yu, 1996] proposed an enhancement of $LB_{bas,1}$ hereafter denoted by $LB_{bas,2}$. Let $\mu = (\mu(1), \ldots, \mu(n))$ (resp. $\eta = (\eta(1), \ldots, \eta(n))$) be a job sequence such that the jobs are sorted in a non-decreasing order of $l_j + p_{1,j}$ (resp. $l_j + p_{2,j}$), for all j in J. The $O(n \log n)$ lower bound $LB_{bas,2}$ is given by expression (3.3).

$$LB_{bas,2} = \max_{1\leq k\leq n}(\max(\sum_{k'=k}^{n} p_{1,\eta(k')} + l_{\eta(k)} + p_{2,\eta(k)}, \sum_{k'=k}^{n} p_{2,\mu(k')} + l_{\mu(k)} + p_{1,\mu(k)})). \tag{3.3}$$

Example 3.1. *Let us consider an $F2|l_j|C_{max}$ instance with four jobs. We report in Table 3.1 the characteristics of the instance.*

Jobs	$p_{1,j}$	l_j	$p_{2,j}$
1	2	2	3
2	4	3	1
3	5	1	2
4	2	1	2

Table 3.1 – $F2|l_j|C_{max}$ instance.

The above lower bounds are computed as follows:

$$LB_{bas,0} = \max(2+2+3, 4+3+1, 5+1+2, 2+1+2) = 8.$$

$$LB_{bas,1} = \max(13+\min(2+3, 3+1, 1+2, 1+2), 8+\min(2+2, 4+3, 5+1, 2+1)) = 16.$$

For $LB_{bas,2}$, we determine first the two job sequences $\mu = (\mu(1), \ldots, \mu(4))$ and $\eta = (\eta(1), \ldots, \eta(4))$, where $\mu(1) = 4$, $\mu(2) = 1$, $\mu(3) = 3$ and $\mu(4) = 2$ (resp. $\eta(1) = 4$, $\eta(2) = 3$, $\eta(3) = 2$ and $\eta(4) = 1$). Then, we compute the lower bound as it is given in expression 3.3:

$$LB_{bas,2} = \max(\max(13+1+2, 8+1+2), \max(11+1+2, 6+2+2), \max(6+3+1, 3+5+1), \max(2+2+3, 1+4+3)) = 16.$$

Resource constraint-based lower bound. Let us consider $I = (J, p_1, l, p_2)$ an instance of $F2|l_j|C_{max}$ and assume that the resource constraint of the first machine is relaxed. This means that M_1 can now process any number of jobs at the same time. Therefore, we obtain an instance I_r of the $1|r_j|C_{max}$ problem with release dates $r_j = p_{1,j} + l_j$ and processing times $p_j = p_{2,j}$, for all j in J. Thus, $L_1 = C^*_{max}(I_r)$ is a valid lower bound that can be obtained by scheduling jobs in a non-decreasing order of r_j. Similarly, by interchanging the roles of M_1 and M_2, we can derive a symmetric lower bound L_2. This yields an $O(n \log n)$ lower bound given by expression (3.4). This lower bound was proposed by [Dell'Amico, 1996].

$$LB_{res,1} = \max(L_1, L_2). \tag{3.4}$$

[Yu, 1996] proposed a lower bound that can be considered as the relaxation of

the resource constraint on both machines. Let S^* be an optimal schedule and $p_{m,[\ell]}$ the processing time of the job scheduled at position ℓ on machine M_m, m in $\{1,2\}$. Moreover, let j^m be the position of job j on M_m. Then, for each job j of J we have:

$$C_{max}(S^*) \geq \sum_{\ell=1}^{j^1} p_{1,[\ell]} + l_j + \sum_{\ell=j^2}^{n} p_{2,[\ell]}, \qquad \forall j \in \{1,\ldots,n\} \tag{3.5}$$

Let $\rho_{m,\ell}$ be the sum of the ℓ smallest values in $\{p_{m,1}, p_{m,2}, \ldots, p_{m,n}\}$, m in $\{1,2\}$. We observe that: $\sum_{\ell=1}^{j^1} p_{1,[\ell]} \geq \rho_{1,j^1}$ and $\sum_{\ell=j^2}^{n} p_{2,[\ell]} \geq \rho_{2,n+1-j^2}$. We thus obtain:

$$C_{max}(S^*) \;\; \geq \;\; \rho_{1,j^1} + l_j + \rho_{2,n+1-j^2}, \qquad \forall j \in \{1,\ldots,n\}. \tag{3.6}$$

Furthermore, by adding the above n equations together, it holds that:

$$n \times C_{max}(S^*) \;\; \geq \;\; \sum_{\ell=1}^{n} \rho_{1,\ell} + \sum_{j=1}^{n} l_j + \sum_{\ell=1}^{n} \rho_{2,n+1-\ell}. \tag{3.7}$$

By observing that the makespan is integral and that $\sum_{\ell=1}^{n} \rho_{2,n+1-\ell} = \sum_{\ell=1}^{n} \rho_{2,\ell}$, we obtain the following lower bound that can be computed in $O(n \log n)$-time:

$$C_{max}(S^*) \geq LB_{res,2} = \left\lceil (\sum_{j=1}^{n} l_j + \sum_{\ell=1}^{n} \rho_{1,\ell} + \sum_{\ell=1}^{n} \rho_{2,\ell})/n \right\rceil. \tag{3.8}$$

Example 3.2. *Let us consider the same instance of Example 3.1. The computation of $LB_{res,1}$ and $LB_{res,2}$ are given as follows.*

For $LB_{res,1}$, we start by sorting the jobs in a non-descending order of their release dates $r_4 = 3,\ r_1 = 4,\ r_3 = 6,\ r_2 = 7$ (resp. in a non-increasing order of their delivery dates $q_1 = 5,\ q_2 = 4,\ q_3 = 3,\ q_4 = 3$), and then we schedule them on M_2 (resp. M_1) with respect to that order. Therefore, the job 4 (resp. 1) is the first to be assigned to M_2 (resp. M_1) and so on until the job 2 (resp. 4), the lower bound value is given by:

$$LB_{res,1} = \max(C^*_{max}(R) = 11, C^*_{max}(Q) = 16) = 16$$

Moreover, in order to compute $LB_{res,2}$, we duplicate in Table 3.2 the needed values. $LB_{res,2}$ is computed as follows:

$$LB_{res,2} = \lceil ((2+3+1+1) + (2+4+8+13) + (1+3+5+8))/4 \rceil = 13.$$

Time delay-based relaxation scheme. The time delay-based relaxation scheme assumes that either a part or all time delays are reduced or eliminated in order to

Jobs	$p_{1,j}$	l_j	$p_{2,j}$	$\rho_{1,j}$	$\rho_{2,j}$
1	2	2	3	2	1
2	4	3	1	4	3
3	5	1	2	8	5
4	2	1	2	13	8

Table 3.2 – Computing $LB_{res,2}$.

relax the problem. For $F2|l_j|C_{max}$, it is assumed that the time delays are reduced in such a way that the permutation schedules become dominant. Let us consider $I = (J, p_1, l, p_2)$ as an instance of $F2|l_j|C_{max}$. [Dell'Amico, 1996] proves that permutation schedules are dominant if:

$$l_j \leq \min_{1 \leq i \leq n} (l_i + p_{1,i}), \forall j \in J. \tag{3.9}$$

From instance I, a new instance $\bar{I} = (J, p_1, \bar{l}, p_2)$ of $F2|l_j|C_{max}$ is derived, where $\bar{l}_j = \min(l_j, \min_{1 \leq i \leq n}(l_i + p_{1,i}))$, j in J. The optimal makespan value of instance $\bar{I}$ can be computed in $O(n \log n)$-time by using Mitten's algorithm ([Mitten, 1959]). It yields the lower bound:

$$LB_{tra,1} = C^*_{max}(\bar{I}). \tag{3.10}$$

Moreover, we propose an enhancement to the above lower bound. The main idea is to improve $LB_{tra,1}$ by using the result of [Yu, 1996]. Yu proves that permutation schedules are dominant if the following condition holds:

$$l_j \leq l_i + \max(p_{1,i}, p_{2,i}), \ \forall j, i \in J. \tag{3.11}$$

Let us consider an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$. From this instance, we derive a new instance $\tilde{I}(J, p_1, \tilde{l}, p_2)$, where $\tilde{l}_j = \min(l_j, \min_{1 \leq i \leq n}(l_i + \max(p_{1,i}, p_{2,i})))$, j in J. We observe that condition (3.11) holds for $\tilde{I}$. As a consequence, we obtain an $O(n \log n)$ lower bound:

$$LB_{tra,2} = C^*_{max}(\tilde{I}). \tag{3.12}$$

Interestingly, this lower bound can be improved by considering subsets of jobs. Let us consider $\xi = (\xi(1), \ldots, \xi(n))$ a job sequence such that jobs are sorted in a non-decreasing order of $l_j + \max(p_{1,j}, p_{2,j})$, j in J. Moreover, we define n instances $\tilde{I}^k = (\tilde{J}^k, \tilde{p}^k_1, \tilde{l}^k, \tilde{p}^k_2)$, k in $\{1, \ldots, n\}$, where $\tilde{J}^k = \{j \in J : l_j + \max(p_{1,j}, p_{2,j}) \geq l_{\xi(k)} + \max(p_{1,\xi(k)}, p_{2,\xi(k)})\}$ and for all j in $\tilde{J}^k$, $\tilde{p}^k_{1,j} = p_{1,j}$, $\tilde{l}^k_j = \min(l_j, l_{\xi(k)} + \max(p_{1,\xi(k)}, p_{2,\xi(k)}))$ and $\tilde{p}^k_{2,j} = p_{2,j}$. It is noteworthy to indicate that instance $\tilde{I}^k$ verifies (3.11), k in $\{1, ..., n\}$. As a consequence, an optimal schedule can be

achieved using the algorithm of [Mitten, 1959]. From Observation 1.3, we obtain an $O(n^2 \log n)$ lower bound:

$$LB_{tra,3} = \max_{1 \leq k \leq n} (C^*_{max}(\tilde{I}^k)). \tag{3.13}$$

Example 3.3. *Applied on the same instance of Example 3.1, the above lower bounds are computed as follows:*

For $LB_{tra,1}$ (resp. $LB_{tra,2}$), we start by computing the value $min_{1 \leq i \leq 4}(l_i + p_{1,i}) = 3$ (resp. $min_{1 \leq i \leq 4}(l_i + max(p_{1,i}, p_{2,i})) = 3$). A new instance $\bar{I} = (J, p_1, \bar{l}, p_2)$ (resp. $\tilde{I} = (\tilde{p}_{1,j}, \tilde{l}_j, \tilde{p}_{2,j})$) is derived. On which, the algorithm of [Mitten, 1959] is applied. We obtain that $LB_{tra,1} = C^*_{max}(\bar{I}) = 16$ (resp. $LB_{tra,2} = C^*_{max}(\tilde{I}) = 16$). The details of these computational steps are duplicated in Table 3.3.

Jobs	$p_{1,j}$	l_j	$p_{2,j}$	$l_j + p_{1,j}$	$l_j + max(p_{1,j}, p_{2,j})$	$\bar{l}_j$	$\tilde{l}_j$
1	2	2	3	4	5	2	2
2	4	3	1	7	7	3	3
3	5	1	2	6	6	1	1
4	2	1	2	3	3	1	1

Table 3.3 – Computing time delay-based lower bounds.

Moreover, for $LB_{tra,3}$, we determine first the job sequence $\xi = (\xi(1), ..., \xi(4))$, where $\xi(1) = 4$, $\xi(2) = 1$, $\xi(3) = 3$ and $\xi(4) = 2$. Then, 4 instances $\tilde{I}^k = (\tilde{J}^k, \tilde{p}^k_1, \tilde{l}^k, \tilde{p}^k_2)$, k in $\{1, \ldots, 4\}$, are defined such that $\tilde{J}^1 = \{1, 2, 3, 4\}$, $\tilde{J}^2 = \{1, 2, 3\}$, $\tilde{J}^3 = \{2, 3\}$ and $\tilde{J}^4 = \{2\}$. Using the algorithm of [Mitten, 1959], $LB_{tra,3}$ is given by $\max(C^*_{max}(\tilde{I}^1) = 16, C^*_{max}(\tilde{I}^2) = 14, C^*_{max}(\tilde{I}^3) = 12, C^*_{max}(\tilde{I}^4) = 8) = 16$.

Job splitting-based relaxation scheme. Let us first recall [Yu et al., 2004]'s lower bound for the $F2|p_{i,j} = 1, l_j|C_{max}$ problem. If we suppose that the time delays are sorted in the non-decreasing order of l_j, then a valid lower bound on the makespan is given by:

$$LB_{uni} = \max_{1 \leq k \leq n} \left(\left\lceil \sum_{j=1}^{k} l_j / k \right\rceil + k + 1 \right) \tag{3.14}$$

A lower bound for $F2|l_j|C_{max}$ can be derived by relaxing the constraint that operations have to be processed without preemption on a given machine. [Yu, 1996] considered the relaxed problem where operations can be split into unitary sub-operations. Given an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$, we denote by $a_j =$

$min(p_{1,j}, p_{2,j})$ and $b_j = max(p_{1,j}, p_{2,j})$, j in J. Every job j of J is divided into a_j unitary sub-jobs j_k $(k = 1, ..., a_j)$, the eventually remaining processing time of operation $O_{1,j}$ or $O_{2,j}$ is transformed to a time delay. We identify two cases:

Case 1: $p_{1,j} \geq p_{2,j}$**.**

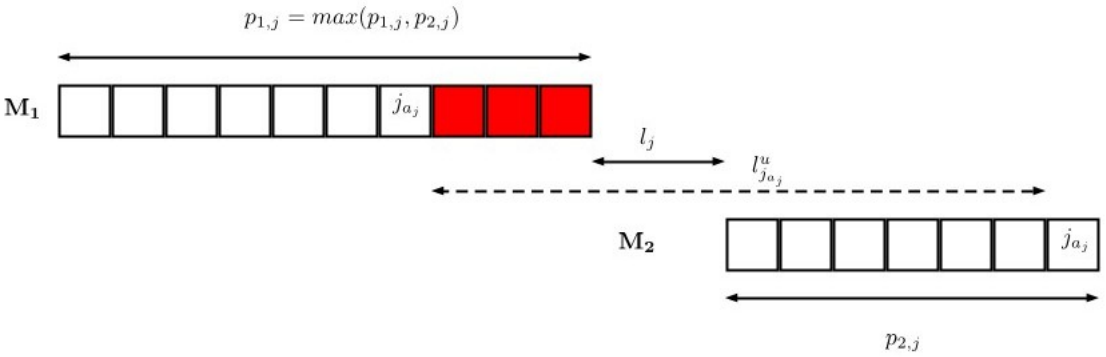


Figure 3.1 – Case 1: $p_{1,j} \geq p_{2,j}$.

In this case, $a_j = p_{2,j}$ and $b_j = p_{1,j}$. For each sub-job j_k $(k = 1, ..., a_j)$, its first operation on M_1 takes the k-th time slot of job j, and its second operation takes the k-th time slot of job j. These new sub-jobs have the same time delay:

$$l^u_{j_k} = (b_j - k) + l_j + (k - 1) = l_j + b_j - 1 \quad (k = 1, ..., a_j)$$

We provide in Figure 3.1 a graphical description of the splitting process of a job with respect to Case 1.

Case 2: $p_{1,j} < p_{2,j}$.

In this case, $a_j = p_{1,j}$ and $b_j = p_{2,j}$. For each sub-job j_k $(k = 1, ..., a_j)$, its first operation on M_1 takes the k-th time slot of job j, and its second operation takes the $(b_j - a_j + k)$-th time slot of job j. These new sub-jobs have the same time delay:

$$l^u_{j_k} = (a_j - k) + l_j + (b_j - a_j + k - 1) = l_j + b_j - 1 \quad (k = 1, ..., a_j)$$

We provide in Figure 3.2 a graphical description of the splitting process of a job with respect to Case 2.

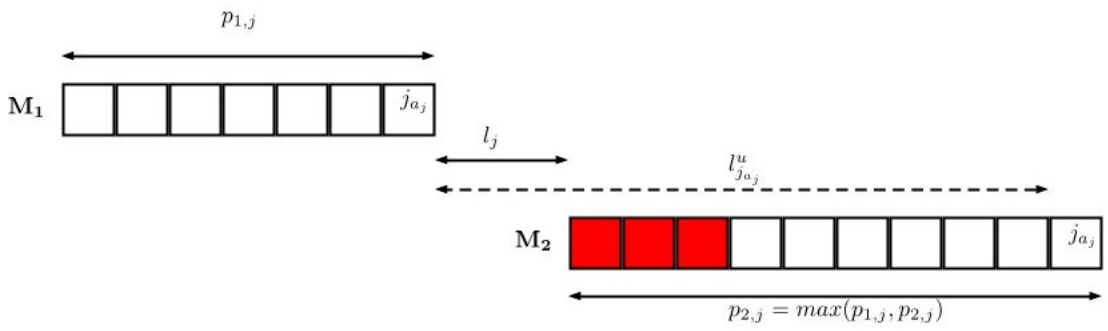


Figure 3.2 – Case 2: $p_{1,j} < p_{2,j}$.

By applying LB_{uni} on the new derived instance, we yields the $O(n)$ lower bound:

$$LB_{split} = \left\lceil (\sum_{j=1}^{n} a_j.l_j^u)/\sum_{j=1}^{n} a_j \right\rceil + 1 + \sum_{j=1}^{n} a_j \tag{3.15}$$

Example 3.4. *Applied on the same instance of Example 3.1, Table 3.4 gives all needed values for LB_{split} computation.*

$$LB_{split} = \lceil (2\times 4+1\times 6+2\times 5+2\times 2)/(2+1+2+2)) \rceil + 1 + (2+1+2+2) = 12.$$

Jobs	$p_{1,j}$	l_j	$p_{2,j}$	a_j	b_j	l_j^u
1	2	2	3	2	3	4
2	4	3	1	1	4	6
3	5	1	2	2	5	5
4	2	1	2	2	2	2

Table 3.4 – Computing job splitting-based lower bound.

3.2.2 Dominance results

In this section, we elucidate dominance relationships between the different lower bounds. We say that LB_x dominates LB_y if for every instance I of the $F2|l_j|C_{max}$ problem, $LB_x(I) \geq LB_y(I)$. For the sake of clarity, we recall the following notation: $\mu = (\mu(1), \ldots, \mu(n))$, $\eta = (\eta(1), \ldots, \eta(n))$ and $\xi = (\xi(1), \ldots, \xi(n))$ are the job sequences such that jobs are sorted in a non-decreasing order of $l_j + p_{1,j}$, $l_j + p_{2,j}$ and $l_j + \max(p_{1,j}, p_{2,j})$, respectively, j in J.

It is noteworthy to mention that [Yu, 1996] proves that $LB_{bas,2}$ dominates $LB_{bas,1}$ and $LB_{bas,0}$. Moreover, we provide the dominance relationships determined in the following propositions.

Proposition 3.1. *$LB_{res,1}$ and $LB_{bas,2}$ are equivalent.*

Proof. Recall that, $LB_{res,1} = \max(L_1, L_2)$, where $L_1 = C^*_{max}(I_r)$ (resp. $L_2 = C^*_{max}(I_q)$) and I_r (resp. I_q) is an instance of the $1|r_j|C_{max}$ problem that is obtained by relaxing the resource constraint of the first machine (resp. the second machine). Since the optimal solution of instance I_r can be obtained by scheduling the jobs in a non-decreasing order of their release dates (i.e., $r_j = p_{1,j} + l_j$), we obtain:

$$L_1 = \max_{1\leq k\leq n}(l_{\mu(k)} + p_{1,\mu(k)} + \sum_{\ell=k}^{n} p_{2,\mu(\ell)}).$$

In the same way, we prove that:

$$L_2 = \max_{1\leq k\leq n}(l_{\eta(k)} + p_{2,\eta(k)} + \sum_{\ell=k}^{n} p_{1,\eta(\ell)}).$$

Therefore,

$$\begin{aligned} LB_{res,1} &= \max(L_1, L_2) \\ &= \max_{1\leq k\leq n}(\max(l_{\eta(k)} + p_{2,\eta(k)} + \sum_{\ell=k}^{n} p_{1,\eta(\ell)}, l_{\mu(k)} + p_{1,\mu(k)} + \sum_{\ell=k}^{n} p_{2,\mu(\ell)})) \\ &= LB_{bas,2}. \end{aligned}$$

□

Proposition 3.2. *$LB_{tra,2}$ dominates $LB_{bas,1}$.*

Proof. Let us consider the instance $\tilde{I} = (J, p_1, \tilde{l}, p_2)$ defined earlier in the implementation of $LB_{tra,2}$, where $\tilde{l_j} = \min(l_j, \min_{1\leq i\leq n}(l_i + \max(p_{1,i}, p_{2,i})))$, j in J. If we denote the last scheduled job on M_1 in an optimal schedule of $\tilde{I}$ by j_{last}, the following inequality holds:

$$\begin{aligned} C^*_{max}(\tilde{I}) &\geq \sum_{j=1}^{n} p_{1,j} + \tilde{l}_{j_{last}} + p_{2,j_{last}} \\ &\geq \sum_{j=1}^{n} p_{1,j} + \min(l_{j_{last}}, \min_{1\leq i\leq n}(l_i + \max(p_{1,i}, p_{2,i}))) + p_{2,j_{last}}. \end{aligned}$$

We identify two cases:

- $l_{j_{last}} \leq \min_{1\leq i\leq n}(l_i+\max(p_{1,i},p_{2,i}))$, then $C^*_{max}(\tilde{I}) \geq \sum_{j=1}^{n} p_{1,j}+l_{j_{last}}+p_{2,j_{last}} \geq \sum_{j=1}^{n} p_{1,j} + \min_{1\leq i\leq n}(l_i + p_{2,i})$.
- Otherwise, it holds that $C^*_{max}(\tilde{I}) \geq \sum_{j=1}^{n} p_{1,j} + \min_{1\leq i\leq n}(l_i + \max(p_{1,i},p_{2,i})) + p_{2,j_{last}} \geq \sum_{j=1}^{n} p_{1,j} + \min_{1\leq i\leq n}(l_i + p_{2,i})$.

By using the same arguments, it is true that $C^*_{max}(\tilde{I}) \geq \sum_{j=1}^{n} p_{2,j}+\min_{1\leq i\leq n}(l_i+p_{1,i})$.

Thus,

$$\begin{aligned} LB_{tra,2} = C^*_{max}(\tilde{I}) &\geq \max(\sum_{j=1}^{n} p_{1,j} + \min_{1\leq i\leq n}(l_i + p_{2,i}), \sum_{j=1}^{n} p_{2,j} + \min_{1\leq i\leq n}(l_i + p_{1,i})) \\ &\geq LB_{bas,1}. \end{aligned}$$

□

Proposition 3.3. *$LB_{tra,3}$ dominates $LB_{tra,1}$ and $LB_{tra,2}$.*

Proof. Recall that $LB_{tra,1} = C^*_{max}(\bar{I})$, $LB_{tra,2} = C^*_{max}(\tilde{I})$ and $LB_{tra,3} = \max_{1\leq k\leq n}(C^*_{max}(\tilde{I}^k))$, where $\bar{I} = (J, p_1, \bar{l}, p_2)$ and $\tilde{I} = \tilde{I}^1 = (J, p_1, \tilde{l}, p_2)$ with $\bar{l}_j = \min(l_j, min_{1\leq i\leq n}(l_i+p_{1,i}))$ and $\tilde{l}_j = \min(l_j, min_{1\leq i\leq n}(l_i + \max(p_{1,i},p_{2,i})))$, j in J.

Note that since $\tilde{l}_j \geq \bar{l}_j$ for all j in J, $LB_{tra,3} \geq C^*_{max}(\tilde{I}^1) \geq C^*_{max}(\tilde{I}) \geq C^*_{max}(\bar{I})$. Thus,

$$LB_{tra,3} \geq LB_{tra,2} \geq LB_{tra,1}.$$

□

Proposition 3.4. *$LB_{tra,3}$ dominates $LB_{bas,2}$.*

Proof. Recall that $LB_{tra,3} = \max_{1\leq k\leq n}(C^*_{max}(\tilde{I}^k))$, where $\tilde{I}^k = (\tilde{J}^k, \tilde{p}^k_1, \tilde{l}^k, \tilde{p}^k_2)$ is an instance of $F2|l_j|C_{max}$ and $\tilde{J}^k = \{j \in J : l_j + \max(p_{1,j},p_{2,j}) \geq l_{\xi(k)} + \max(p_{1,\xi(k)},p_{2,\xi(k)})\}$, k in $\{1,...,n\}$.

With respect to equation (3.3), let k_0 be an integer in $\{1,...,n\}$ such that:

$$LB_{bas,2} = \max(\sum_{k'=k_0}^{n} p_{1,\eta(k')} + l_{\eta(k_0)} + p_{2,\eta(k_0)}, \sum_{k'=k_0}^{n} p_{2,\mu(k')} + l_{\mu(k_0)} + p_{1,\mu(k_0)}).$$

In this proof, we show that instances of $\tilde{I}^k$, k in $\{1,...,n\}$ exist such that their makespan values dominate $\sum_{k'=k_0}^{n} p_{1,\eta(k')}+l_{\eta(k_0)}+p_{2,\eta(k_0)}$ and $\sum_{k'=k_0}^{n} p_{2,\mu(k')}+l_{\mu(k_0)}+p_{1,\mu(k_0)})$.

Let k_{min} be an integer such that $k_{min} = \arg\min_{k\geq k_0}(l_{\mu(k)} + \max(p_{1,\mu(k)},p_{2,\mu(k)}))$ and p the position of $\mu(k_{min})$ in the job sequence $(\xi(1),\ldots,\xi(n))$. Obviously, it holds that:

- $l_{\xi(p)} + \max(p_{1,\xi(p)}, p_{2,\xi(p)}) \geq l_{\xi(p)} + p_{1,\xi(p)} \geq l_{\mu(k_0)} + p_{1,\mu(k_0)}$ since $\xi(p) = \mu(k_{min})$.
- $\forall k \geq k_0,\ l_{\mu(k)} + \max(p_{1,\mu(k)}, p_{2,\mu(k)}) \geq l_{\xi(p)} + \max(p_{1,\xi(p)}, p_{2,\xi(p)})$. Therefore, for all $k \geq k_0$, $\mu(k)$ belongs to $\tilde{J}^p$.

Moreover, we observe that:

$$\forall j \in \tilde{J}^p,\ \tilde{l_j}^p = \min(l_j, l_{\xi(p)} + \max(p_{1,\xi(p)}, p_{2,\xi(p)})).$$

Thus,

$$\begin{array}{rcl} \forall \mu(k) \in \tilde{J}^p \text{ and } k \geq k_0,\ \tilde{l}^p_{\mu(k)} & \geq & \min(l_{\mu(k)}, l_{\mu(k_0)} + p_{1,\mu(k_0)}) \\ \forall \mu(k) \in \tilde{J}^p \text{ and } k \geq k_0,\ \tilde{l}^p_{\mu(k)} + p_{1,\mu(k)} & \geq & \min(l_{\mu(k)} + p_{1,\mu(k)}, l_{\mu(k_0)} + p_{1,\mu(k_0)} + p_{1,\mu(k)}) \\ & \geq & l_{\mu(k_0)} + p_{1,\mu(k_0)}. \end{array}$$

It follows that $\sum_{k'=k_0}^{n} p_{2,\mu(k')} + l_{\mu(k_0)} + p_{1,\mu(k_0)}$ is a valid lower bound of $\tilde{I}^p$.

By using the same arguments, we prove that an integer $p^{'}$ in $\{1, ..., n\}$ exists such that $\sum_{k'=k_0}^{n} p_{1,\eta(k')} + l_{\eta(k_0)} + p_{2,\eta(k_0)}$ is a valid lower bound of instance $\tilde{I}^{p^{'}}$. Finally, we obtain:

$$\begin{array}{rcl} LB_{tra,3} & \geq & \max(C^*_{max}(\tilde{I}^p), C^*_{max}(\tilde{I}^{p^{'}})) \\ & \geq & \max(\sum_{k'=k_0}^{n} p_{1,\eta(k')} + l_{\eta(k_0)} + p_{2,\eta(k_0)}, \sum_{k'=k_0}^{n} p_{2,\mu(k')} + l_{\mu(k_0)} + p_{1,\mu(k_0)}) \\ & \geq & LB_{bas,2}. \end{array}$$

□

In order to obtain a complete picture of the dominance relationships between the different lower bounds, we consider the $F2|l_j|C_{max}$ instances I_1, I_2, I_3 and I_4. The details of these instances are reported in Table 3.5. The results of the different lower bounds on these instances are summarized in Table 3.6.

- On the basis of Instances I_2, I_3 and I_4, no dominance relationship exists between $LB_{res,2}$ and the set of lower bounds: $LB_{bas,0}$, $LB_{bas,1}$, $LB_{bas,2}$, $LB_{res,1}$, LB_{split}, $LB_{tra,1}$, $LB_{tra,2}$ and $LB_{tra,3}$.
- On the basis of Instances I_3 and I_4, no dominance relationship exists between LB_{split} and the set of lower bounds: $LB_{bas,0}$, $LB_{bas,1}$, $LB_{bas,2}$, $LB_{res,1}$, $LB_{tra,1}$, $LB_{tra,2}$ and $LB_{tra,3}$.
- On the basis of Instances I_1 and I_2, no dominance relationship exists between $LB_{bas,0}$ and the set of lower bounds: $LB_{bas,1}$, $LB_{tra,1}$ and $LB_{tra,2}$.

- On the basis of Instances I_1 and I_3, no dominance relationship exists between $LB_{bas,2}$, which is equivalent to $LB_{res,1}$, and the set of lower bounds: $LB_{tra,1}$ and $LB_{tra,2}$.
- On the basis of Instances I_2 and I_3, no dominance relationship exists between $LB_{tra,1}$ and $LB_{bas,1}$.

Table 3.5 – Details of instances I_1, I_2, I_3 and I_4.

Instance	Job	$p_{1,j}$	l_j	$p_{2,j}$	Instance	Job	$p_{1,j}$	l_j	$p_{2,j}$
I_1	1	1	10	3	I_3	1	3	0	3
	2	1	5	6		2	1	8	1
	3	2	20	2		3	5	6	5
I_2	1	27	100	71	I_4	1	8	2	1
	2	71	150	27		2	7	3	2
	3	27	200	71		3	8	4	1
	4	71	250	27					

Table 3.6 – Lower bound results on Table 3.5 counter examples.

Instance	$LB_{bas,0}$	$LB_{bas,1}$	$LB_{bas,2}$	$LB_{res,1}$	$LB_{res,2}$	LB_{split}	$LB_{tra,1}$	$LB_{tra,2}$	$LB_{tra,3}$
I_1	24	18	24	24	21	21	18	18	24
I_2	348	367	369	369	376	354	350	373	375
I_3	16	12	16	16	14	18	17	17	17
I_4	23	26	26	26	21	15	26	26	26

Before closing this section, the dominance relationships between the presented lower bounds are illustrated in Figure 3.3, where each node represents a lower bound. If a lower bound LB_x dominates LB_y, then an outgoing arc from the corresponding node of LB_y to that of LB_x exists.

3.3 Mixed Integer Linear Programming models for $F2|l_j|C_{max}$

Several mathematical formulations were proposed for scheduling problems [Keha et al., 2009]. These formulations are based on different decision variables, mainly, the time indexed-based variables, the completion time-based variables, the assignment-based variables and the linear ordering-based variables.

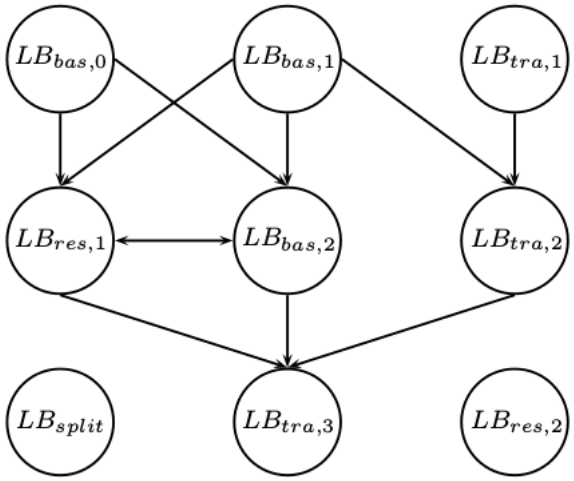


Figure 3.3 – Dominance relationships between the different lower bounds.

In this section, we study only three of the above mathematical formulations for the $F2|l_j|C_{max}$ problem. The time indexed variable-based formulation is not considered since it has a pseudo-polynomial number of variables.

As a consequence of observation 1.2, we suppose in the proposed formulations that the jobs are continuously processed on M_1 and M_2.

3.3.1 Position-based formulation

This formulation is based on binary decision variables $X^m_{j,k}$ that determine the positions of jobs on the two machines, j in J, m in $\{1,2\}$ and k in $\{1,...,n\}$. Precisely, $X^m_{j,k} = 1$ if job j is scheduled at the k-th position on M_m and 0 otherwise. Moreover, $C^m_{[k]}$ corresponds to the completion time of the job scheduled at position k on M_m. The model constraints and objective function are described as follows:

$$\textbf{min} \quad C^2_{[n]} \tag{3.16}$$

$$\text{s.t.} \quad \sum_{k=1}^{n} X^m_{j,k} = 1, \qquad \forall j \in J, m \in \{1,2\} \tag{3.17}$$

$$\sum_{j\in J} X^m_{j,k} = 1, \qquad \forall k \in \{1,...,n\}, m \in \{1,2\} \tag{3.18}$$

$$C^1_{[k]} = \sum_{\nu=1}^{k} \sum_{j\in J} X^1_{j,\nu} p_{1,j}, \qquad \forall k \in \{1,...,n\} \tag{3.19}$$

$$C^2_{[k]} = C^2_{[n]} - \sum_{\nu=k+1}^{n} \sum_{j\in J} X^2_{j,\nu} p_{2,j}, \qquad \forall k \in \{1,...,n-1\} \tag{3.20}$$

$$C^2_{[k]} \geq C^1_{[\nu]} + l_j + p_{2,j} - M(2 - X^1_{j,\nu} - X^2_{j,k}), \quad \forall j \in J; k,\ \nu \in \{1,...,n\} \tag{3.21}$$

$$C^m_{[k]} \geq 0, \qquad \forall k \in \{1,...,n\}, m \in \{1,2\} \tag{3.22}$$

$$X^m_{j,k} \in \{0,1\}, \qquad \forall j \in J, k \in \{1,...,n\}; m \in \{1,2\} \tag{3.23}$$

The objective function (3.16) aims to minimize $C^2_{[n]}$, which represents the completion time of the last scheduled job on M_2. Constraints (3.17) ensure that a job can only be assigned to a unique position on each machine. Constraints (3.18) indicate that at each position on the two machines only one job can be scheduled. Constraints (3.19) and (3.20) give the completion time of the job scheduled at position k on M_1 and M_2, respectively. Constraints (3.21) state that each job after being executed on the first machine has to wait at least its time delay to be assigned to the second one. Constraints (3.22) and (3.23) impose integrity and non-negative conditions on the problem variables. In this formulation, the value of M is taken to be equal to $\sum_{j\in J} p_{1,j} + \max_{j\in J}(p_{2,j+l_j})$.

3.3.2 Job precedence-based formulation

We focus in this formulation on the local precedence relationships between all jobs. The formulation is based on the following variables. $U^m_{i,j}$ is a binary decision variable that takes the value 1 if job i locally precedes job j on M_m and 0 otherwise, m in $\{1, 2\}$. Another decision variable μ^m_j is defined in order to indicate the position of job j on M_m. In addition, the completion time of job j on M_m is denoted by $C_{m,j}$ and the makespan is represented by Γ. The problem is modeled as follows:

$$\textbf{min} \quad \Gamma \tag{3.24}$$

$$\text{s.t.} \quad \sum_{i\in J\cup\{0\}} U^m_{i,j} = 1, \qquad \forall j \in J; m \in \{1,2\} \tag{3.25}$$

$$\sum_{j\in J\cup\{n+1\}} U^m_{i,j} = 1, \qquad \forall i \in J; m \in \{1,2\} \tag{3.26}$$

$$\mu^m_0 - \mu^m_j + (n+1)U^m_{0,j} \leq n, \qquad \forall j \in J; m \in \{1,2\} \tag{3.27}$$

$$\mu^m_j - \mu^m_{n+1} + (n+1)U^m_{j,n+1} \leq n, \qquad \forall j \in J; m \in \{1,2\} \tag{3.28}$$

$$\mu^m_i - \mu^m_j + (n+1)U^m_{i,j} \leq n, \qquad \forall i,j \in J; m \in \{1,2\} \tag{3.29}$$

$$C_{m,i} \geq C_{m,j} + p_{m,i} - M'(1 - U^m_{j,i}), \quad \forall i,j \in J; m \in \{1,2\} \tag{3.30}$$

$$C_{2,j} \geq C_{1,j} + l_j + p_{2,j}, \qquad \forall j \in J \tag{3.31}$$

$$\Gamma \geq C_{2,j}, \qquad \forall j \in J \tag{3.32}$$

$$C_{m,j}, \Gamma, \mu^m_j \in \mathbb{N}^+ \text{ and } U^m_{i,j} \in \{0,1\}, \quad \forall i,\ j \in J, m \in \{1,2\} \tag{3.33}$$

For the objective function (3.24), it aims to minimize the makespan value Γ. Constraints (3.25) and (3.26) state that a job has one successor and one predecessor after adding two fictive jobs (job 0 and job $n+1$). Constraints (3.27), (3.28) and (3.29) are the Miller-Tucker-Zemlin (MTZ) sub-tour elimination constraints (see

[Miller et al., 1960]). They invoke the absence of cyclic relationships between jobs. Constraints (3.30) allow us to compute the completion times of all jobs on the two machines. Constraints (3.31) ensure that for each job its time delay must elapse between the end of its first operation and the start of its second operation. Constraints (3.32) specify that the value of variable Γ is grater than or equal to the maximum completion time value on M_2. Type and non-negative conditions on the problem variables are given in constraints (3.33). The value of $M^{'}$ is calculated as follows $M^{'} = n \times \max_{j \in J}(p_{1,j}, p_{2,j})$.

3.3.3 Linear ordering-based formulation

The considered formulation is based on linear ordering-based variables. The decision variables are defined for each pair of jobs (i, j) of J^2, where $Y_{i,j}^m$ takes the value 1 if i precedes j on M_m and 0 otherwise, m in $\{1, 2\}$. Furthermore, $C_{m,j}$ represents the completion time of job j on M_m and the makespan is denoted by $\Gamma^{'}$. Using these definitions, the model is given by:

$$\textbf{min} \qquad \Gamma^{'} \tag{3.34}$$

$$\text{s.t.} \qquad Y_{i,j}^m + Y_{j,i}^m = 1, \qquad \forall i,\ j \in J\ i \neq j; m \in \{1,2\} \tag{3.35}$$

$$Y_{i,j}^m \geq Y_{i,v}^m + Y_{v,j}^m - 1, \qquad \forall i, j, v \in J; m \in \{1,2\} \tag{3.36}$$

$$C_{1,j} = \sum_{i=1}^{n} Y_{i,j}^1 p_{1,i} + p_{1,j}, \qquad \forall j \in J \tag{3.37}$$

$$C_{2,j} \geq C_{1,j} + l_j + p_{2,j}, \qquad \forall j \in J \tag{3.38}$$

$$C_{2,j} = \Gamma^{'} - \sum_{i=1}^{n} Y_{j,i}^2 p_{2,i}, \qquad \forall j \in J \tag{3.39}$$

$$\Gamma^{'} \geq 0,\ C_{m,j} \geq 0,\ Y_{i,j}^m \in \{0,1\}, \quad \forall i,\ j \in J,\ m \in \{1,2\} \tag{3.40}$$

The objective function (3.34) minimizes the makespan. Constraints (3.35) ensure that for each pair of jobs, one of them has to precede the other on each machine. Constraints (3.36) stand for the transitivity constraints that force a linear order between each three jobs. Constraints (3.37) and (3.39) take into account the job precedence and enforce them to be continuously processed without idle on M_1 and M_2, respectively. In addition, Constraints (3.38) ensure that a job after being processed on M_1 has to wait at least its time delay to be executed on M_2. The nature of decision variables $\Gamma^{'}$, $C_{m,j}$ and $Y_{i,j}^m$ is displayed by Constraints (3.40).

Remark 3.1. *From a complexity point of view, the three formulations have similar number of variables and constraints. Precisely, all three admit $O(n^2)$ variables and $O(n^3)$ constraints except for the job precedence-based formulation that has only $O(n^2)$ constraints.*

In our case, preliminary computational results conducted on the above formulations show that the linear ordering-based formulation is the most promising one. This observation can be explained by the fact that the first and the second models can be considered as a big-M model that is known to be the hardest to solve [Codato and Fischetti, 2006].

3.4 Linear programming-based lower bounds for $F2|l_j|C_{max}$

In this section, we introduce three linear programming-based lower bounds for $F2|l_j|C_{max}$. The first scheme is based on solving the linear relaxation of the linear ordering variable-based model. In the second scheme, we first relax the $F2|l_j|C_{max}$ problem to the $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ problem. We then apply the compact ILP model proposed in Chapter 2 (see Section 2.3) on it. The third scheme consists in introducing an assignment-based lower bound.

3.4.1 Linear ordering variable-based lower bound

We present here a lower bound for $F2|l_j|C_{max}$ that is obtained after solving the LP relaxation of the mathematical model (3.34)-(3.40). In order to strengthen the LP relaxation of the model, we propose two valid inequalities.

The first valid inequality is based on the additional waiting time that a job has to fulfill after being available for processing on M_2. We recall that given a sequence of jobs on M_1, solutions in which the jobs are scheduled on M_2 according to their arrival times are dominant. Therefore, if a job j is preceded by a job i on M_1, then a lower bound on the minimum additional waiting time observed by job j or job i is $\omega_{i,j}$, where $\omega_{i,j} = \begin{cases} \max(0, l_i + p_{2,i} - p_{1,j} - l_j), & \text{if } l_i \leq p_{1,j} + l_j \\ \max(0, p_{1,j} + l_j + p_{2,j} - l_i), & \textit{otherwise.} \end{cases}$

A lower bound on the total additional waiting time Δ can be obtained by solving the assignment problem where the assignment costs are $\delta_{i,j},\ i, j \in \{1, \ldots, n\}$. In the following, we describe how the assignment costs are computed. Note that the first

scheduled job (resp. the last scheduled job) on M_1 is assumed to be preceded (resp. succeeded) by a dummy job (job 0, resp. job $n+1$). Obviously, since job 0 cannot precede job $n+1$, and a job cannot precede itself, then we set $\delta_{0,n+1} = \infty$ and $\delta_{j,j} = \infty$, for all j in $\{1, \ldots, n\}$.

Remark 3.2. *Let I be an instance of $F2|l_j|C_{max}$ and LB (resp. UB) a lower bound (resp. an upper bound) on the the makespan of I. If a schedule of makespan LB exists, then the jobs can be continuously processed without any idle time, from time 0 on M_1 and from time $(LB - \sum_{j=1}^{n} p_{2,j})$ on M_2. Then, we obtain the following assignment costs:*

- *for all i in $\{1, \ldots, n\}$, $\delta_{0,i} = \max(0, LB - \sum_{j=1}^{n} p_{2,j} - l_i - p_{1,i})$*
- *for all i in $\{1, \ldots, n\}$, if $\sum_{j=1}^{n} p_{1,j} + l_i + p_{2,i} > UB$, i cannot be processed at the last position on M_1, then $\delta_{i,n+1} = \infty$. Otherwise $\delta_{i,n+1} = 0$*

In order to set $\delta_{i,j}$, for all i, j in $\{1, \ldots, n\}$, $i \neq j$, we introduce in the following lemma a new dominance rule.

Lemma 3.1. *Let $I = (J, p_1, l, p_2)$ be an instance of $F2|l_j|C_{max}$ and i, j two jobs of J such that $p_{1,j} + l_j \leq p_{1,i} + l_i$, $l_i \leq p_{2,j} + l_j$ and $p_{1,j} \leq p_{2,j}$. For any schedule S of I, if j and i are adjacent on M_1 then j should precede i on M_1.*

Proof. Let σ_1 be an arbitrary sub-sequence on M_1 where $J_{\sigma_1} \subset J$. In this proof, we show that we can construct a schedule S' of $\Omega_{(\sigma_1 \oplus j \oplus i)}$ from an arbitrary schedule S of $\Omega_{(\sigma_1 \oplus i \oplus j)}$ in a way that $C_{max}(S') = C_{max}(S)$. For S', the job sequence on M_1 is obtained by simply interchanging i and j and keeping the same executing order for the rest of the jobs, like in S. Therefore, it holds that $t_{1,j}(S') = t_{1,i}(S)$, $t_{1,i}(S') + p_{1,i} = t_{1,j}(S) + p_{1,j}$ and $t_{1,v}(S') = t_{1,v}(S)$ for all v in $J \backslash \{i, j\}$. Moreover, we identify two cases:

Case 1: j precedes i on M_2.
The job sequence on M_2 of S' is similar to that of S, where $t_{2,v}(S') = t_{2,v}(S)$, for all v in J. Thus, each job of $J \backslash \{i, j\}$ is available on M_2 before its starting time. In addition, it holds that:

$$\begin{aligned} t_{1,j}(S') + p_{1,j} + l_j &\leq t_{1,j}(S) + p_{1,j} + l_j \\ &\leq t_{2,j}(S) \\ &\leq t_{2,j}(S') \end{aligned}$$

and

$$t_{1,i}(S^{'}) + p_{1,i} + l_i \;\; = \;\; t_{1,j}(S) + p_{1,j} + l_i.$$

Since $l_i \leq l_j + p_{2,j}$, we obtain:

$$\begin{aligned} t_{1,i}(S^{'}) + p_{1,i} + l_i &\leq t_{1,j}(S) + p_{1,j} + l_j + p_{2,j} \\ &\leq t_{2,j}(S) + p_{2,j} \\ &\leq t_{2,i}(S^{'}). \end{aligned}$$

On the basis of the above remarks, we conclude that all jobs are available for processing on M_2 before their starting times in $S^{'}$.

Case 2: i precedes j on M_2.
The job sequence on M_2 of $S^{'}$ is derived from that of S by shifting left j on M_2 to locally precede i in a way that $t_{2,j}(S^{'}) = t_{2,i}(S)$. Since $t_{1,v}(S^{'}) = t_{1,v}(S)$ and $t_{2,v}(S^{'}) \geq t_{2,v}(S)$ for all v in $J\backslash\{i,j\}$, each job of $J\backslash\{i,j\}$ is available on M_2 before its starting time. We recall that:

$$t_{1,j}(S^{'}) + p_{1,j} + l_j = t_{1,i}(S) + p_{1,j} + l_j.$$

Since $p_{1,j} + l_j \leq p_{1,i} + l_i$, it holds that:

$$\begin{aligned} t_{1,j}(S^{'}) + p_{1,j} + l_j &\leq t_{1,i}(S) + p_{1,i} + l_i \\ &\leq t_{2,i}(S) \\ &\leq t_{2,j}(S^{'}). \end{aligned}$$

Moreover, we have:

$$\begin{aligned} t_{1,i}(S^{'}) + p_{1,i} + l_i &= t_{1,j}(S^{'}) + p_{1,j} + p_{1,i} + l_i \\ &= t_{1,i}(S) + p_{1,j} + p_{1,i} + l_i \\ &\leq t_{2,i}(S) + p_{1,j}. \end{aligned}$$

Since $p_{1,j} \leq p_{2,j}$, we obtain:

$$\begin{aligned} t_{1,i}(S^{'}) + p_{1,i} + l_i &\leq t_{2,i}(S) + p_{2,j} \\ &\leq t_{2,j}(S^{'}) + p_{2,j} \\ &\leq t_{2,i}(S^{'}). \end{aligned}$$

On the basis of the above remarks, we conclude that all jobs are available for processing on M_2 before their starting times in $S^{'}$. □

Corollary 3.1. *Let $I = (J, p_1, l, p_2)$ be an instance of $F2|l_j|C_{max}$ and $i,\ j$ two jobs of J. If $p_{1,j} + l_j \leq p_{1,i} + l_i$, $l_i \leq p_{2,j} + l_j$ and $p_{1,j} \leq p_{2,j}$, then $\delta_{i,j} = \infty$. Otherwise $\delta_{i,j} = \omega_{i,j}$.*

Similarly, by interchanging the role of M_1 and M_2, we obtain $\Delta^{'}$ another lower bound on the total additional waiting time. Therefore, the following valid inequality holds:

$$\sum_{j=1}^{n} C_{2,j} \geq \sum_{j=1}^{n} C_{1,j} + \sum_{j=1}^{n}(p_{2,j} + l_j) + \max(\Delta, \Delta^{'}). \tag{3.41}$$

In the second valid inequality, we take advantage of the property of [Yu, 1996]. Recall that [Yu, 1996] proved that a permutation schedule that is optimal exists for an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$ if the processing times and the time delays are with respect to

$$l_j \leq l_i + \max(p_{1,i}, p_{2,i}),\ \forall i, j \in J.$$

Thus, an optimal permutation solution for I can be computed in a polynomial time using the algorithm of [Mitten, 1959].

The second valid inequality consists in computing optimal solutions of polynomial-time-solvable sub-instances. These latter are extracted from the original instance by computing all maximal cliques on a particular *Interval graph*.

Definition 3.1. *Let us consider two instances $I = (J, p_1, l, p_2)$ and $I_s = (J_s, p_1, l, p_2)$ of $F2|l_j|C_{max}$ where $J_s \subset J$. I_s is called a Permutation Sub-Instance (PSI) of I if and only if it holds that:*

$$l_j \leq l_i + \max(p_{1,i}, p_{2,i}) \text{ and } l_i \leq l_j + \max(p_{1,j}, p_{2,j}),\ \forall i, j \in J_s; i \neq j.$$

Observation 3.1. *Let us consider two instances $I = (J, p_1, l, p_2)$ and $I_s = (J_s, p_1, l, p_2)$ of $F2|l_j|C_{max}$. If I_s is a PSI of I, then I_s can be solved to optimality in polynomial time.*

Proof. It holds that $l_j \leq l_i + \max(p_{1,i}, p_{2,i}),\ \forall i, j \in J_s$. Thus, an optimal permutation solution for I_s can be determined in a polynomial time using the algorithm of [Mitten, 1959]. □

Definition 3.2. *Let us consider two instances $I = (J, p_1, l, p_2)$ and $I_s = (J_s, p_1, l, p_2)$ of $F2|l_j|C_{max}$ where $J_s \subset J$. I_s is called a Maximal Permutation Sub-Instance*

(MPSI) if and only if I_s is PSI and there is no job j of $J \setminus J_s$ such that:

$$l_j \leq l_i + \max(p_{1,i}, p_{2,i}) \text{ and } l_i \leq l_j + \max(p_{1,j}, p_{2,j}),\ \forall i \in J_s.$$

In order to determine all MPSIs from an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$, we define an undirected graph $G = (V, E)$ for I, where V (resp. E) stands for the set of vertex (resp. edges). The set V contains n vertex such that each job j of J is presented using the vertex u_j. Moreover, an edge of E exists between the two vertex u_i and u_j if and only if

$$l_j \leq l_i + \max(p_{1,i}, p_{2,i}) \text{ and } l_i \leq l_j + \max(p_{1,j}, p_{2,j}),\ i, j \in J.$$

Interestingly, determining all MPSIs of instance I and computing maximal cliques in G are equivalent.

Proposition 3.5. *Let us consider an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$. At most, n MPSIs of I exist.*

Proof. Let us consider an undirected graph $G = (V, E)$ defined for I. Each vertex u_j of V is associated to an interval $h_j = [l_j, l_j + \max(p_{1,j}, p_{2,j})]$. Every edge of E that exists between the two vertex u_i and u_j of V can be given by a non-empty intersection between the two intervals h_i and h_j. Precisely, if an edge exists between u_i and u_j, then it holds that:

$$l_j \leq l_i + \max(p_{1,i}, p_{2,i}) \text{ and } l_i \leq l_j + \max(p_{1,j}, p_{2,j}).$$

Thus, we obtain:

$$h_i \cap h_j \neq \emptyset.$$

As a consequence, G is an interval graph. Thus, we can determine at most n MPSIs of an instance I by simply determining all maximal cliques in G. □

Corollary 3.2. *Let $I = (J, p_1, l, p_2)$ be an instance of $F2|l_j|C_{max}$ and $G = (V, E)$ an interval graph of I. We can determine all MPSIs of I in $O(n+ \mid E \mid)$-time.*

Corollary 3.3. *Let $I = (J, p_1, l, p_2)$ be an instance of $F2|l_j|C_{max}$ and $I_s = (J_s, p_1, l, p_2)$ a MPSI of I. It holds that:*

$$\max_{j \in J_s}(C_{2,j}) - \min_{j \in J_s}(C_{1,j} - p_{1,j}) \geq C^*_{max}(I_s). \tag{3.42}$$

We can use the same reasoning to inject the value of the best combinatorial lower

bound within our MIP model. Precisely, if we consider all jobs of J, it holds that:

$$\max_{j\in J}(C_{2,j}) - \min_{j\in J}(C_{1,j} - p_{1,j}) \geq LB(I).$$

As a result of Corollary 3.3, we provide in the following a linearization of the second valid inequality. Let $I = (J, p_1, l, p_2)$ be an instance of $F2|l_j|C_{max}$ and $I_s = (J_s, p_1, l, p_2)$ a MPSI of I. At first, we define two fictive jobs called job i_s and job j_s in a way that i_s (resp. j_s) must locally precede (resp. locally follow) the first scheduled job (resp. the last scheduled job) of J_s on M_1 (resp. M_2). We also define a set of new variables. $V^1_{s,j}$ (resp. $V^2_{s,j}$) is a binary decision variable that takes the value 1 if job j is the first job (resp. last job) of J_s to be executed on M_1 (resp. M_2). Using these definitions, the valid inequality is given by:

$$\sum_{j\in J_s} V^m_{s,j} = 1, \quad \forall m \in \{1,2\} \tag{3.43}$$

$$Y^1_{v,i_s} \geq \sum_{j\in J_s} Y^1_{v,j} - \mid J_s \mid +1, \quad \forall v \in J\backslash\{J_s\} \tag{3.44}$$

$$Y^2_{j_s,v} \geq \sum_{j\in J_s} Y^2_{j,v} - \mid J_s \mid +1, \quad \forall v \in J\backslash\{J_s\} \tag{3.45}$$

$$C_{1,i_s} \geq \sum_{v\in J\backslash\{J_s\}} Y^1_{v,i_s}.p_{1,v}, \tag{3.46}$$

$$C_{2,j_s} \leq \Gamma' - \sum_{v\in J\backslash\{J_s\}} Y^2_{j_s,v}.p_{2,v}, \tag{3.47}$$

$$\sum_{j\in J_s; j\neq i} Y^1_{i,j} \geq (\mid J_s \mid -1).V^1_{s,i}, \qquad \forall i \in J_s \tag{3.48}$$

$$\sum_{j\in J_s; j\neq i} Y^2_{j,i} \geq (\mid J_s \mid -1).V^2_{s,i}, \qquad \forall i \in J_s \tag{3.49}$$

$$C_{2,j_s} - C_{1,i_s} \geq C^*_{max}(I_s). \tag{3.50}$$

Constraints (3.43) ensure that a unique job of J_s precedes (resp. follows) the rest of jobs of J_s on M_1 (resp. M_2). Constraints (3.44) and (3.45) allow us to determine the set of jobs of $J\backslash\{J_s\}$ that precedes (resp. follows) job i_s (resp. job j_s) on M_1 (resp. M_2). The completion time of job i_s on M_1 (resp. job j_s on M_2) is determined using Constraint (3.46) (resp. Constraint (3.47)). Constraints (3.48) and (3.49) ensure the respect of precedence relationships between the first scheduled job of J_s on M_1 (resp. the last scheduled job of J_s on M_2) and the other jobs of J_s. Finally, Constraint (3.50) permits to inject the value of the optimal makespan value of instance I_s.

Hereafter, we define two versions of the lower bound: a version with all constraints called LB_1^N and a version of LB_1^N without Constraints (3.36), which are the transitivity constraints, called LB_2^N. Obviously, LB_1^N dominates LB_2^N. However, LB_1^N admits $O(n^3)$ constraints of (3.36) that could affect the effectiveness and the efficiency of the lower bound.

3.4.2 $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$-based lower bound

We propose here a new lower bound for the $F2|l_j|C_{max}$ problem. This lower bound is obtained as follows. We first relax the $F2|l_j|C_{max}$ problem to the $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ problem. We then apply ILP_{ass} (see Section 2.3) on the relaxed instance of $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$. The lower bound is described in the following proposition.

Proposition 3.6. *Let us consider an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$ and two integer $a,\ b \in \mathbb{N}^+$. From I, we derive an instance $I_{a,b} = (J', l')$ of $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ after splitting each job j of J into $\rho = \left\lfloor \min(\frac{p_{1,j}}{a}, \frac{p_{2,j}}{b}) \right\rfloor$ sub-jobs that are included in J'. A new time delay is defined for each sub-job j^k derived from job j of J, k in $\{1, ..., \rho\}$:*

$$l'_{j^k} = p_{1,j} - k \times a + l_j + p_{2,j} - (\rho + 1 - k) \times b.$$

Interestingly, applying ILP_{ass} on $I_{a,b}$, constitutes a valid lower bound on the makespan of I. Hereafter, we denote the obtained lower bound by $LB_3^N(a, b) = ILP_{ass}(I_{a,b})$.

Proof. Depending on the values of a and b, all jobs that verify that $p_{1,j} < a$ or $p_{2,j} < b$, j in J will be eliminated. Let J_r be the remaining set of jobs with respect to the values of a and b. Taking the view point of job splitting, we consider that each job j of J_r has ρ sub-jobs. We identify two cases:

Case 1: $\frac{p_{1,j}}{a} \leq \frac{p_{2,j}}{b}$.

In this case, $\rho = \left\lfloor \frac{p_{1,j}}{a} \right\rfloor$. For k in $\{1, ..., \rho\}$, the first operation of j^k is processed at the $(k-1) \times a + 1$-th time slot of job j on M_1. Moreover, its second operation is processed at the $p_{2,j} - b \times (\rho - k + 1) + 1$-th time slot of job j on M_2. These sub-jobs

have a common delay:

$$l'_{j^k} = p_{1,j} - k \times a + l_j + p_{2,j} - (\rho - k + 1) \times b.$$

Case 2: $\frac{p_{1,j}}{a} > \frac{p_{2,j}}{b}$.

In this case, $\rho = \left\lfloor \frac{p_{2,j}}{b} \right\rfloor$. For k in $\{1, ..., \rho\}$, the first operation of j^k is processed at the $(k-1) \times a + 1$-th time slot of job j on M_1. Moreover, its second operation is processed at the $p_{2,j} - b \times (\rho - k + 1) + 1$-th time slot of job j on M_2. These sub-jobs have a common delay:

$$l'_{j^k} = p_{1,j} - k \times a + l_j + p_{2,j} - (\rho - k + 1) \times b.$$

We conclude that the sub-jobs are executed within the delays of the original job. Thus, it holds that:

$$C^*_{max}(I) \geq C^*_{max}(I_{a,b}).$$

□

Corollary 3.4. *Let us consider an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$ and the parameters $a_{max} = \max_{j \in J}(p_{1,j})$ and $b_{max} = \max_{j \in J}(p_{2,j})$. It holds that:*

$$C^*_{max}(I) \geq LB_3^N = \max_{1 \leq a \leq a_{max}, 1 \leq b \leq b_{max}} (LB_3^N(a, b)).$$

Unfortunately, it is very difficult to compute LB_3^N in practice. In fact, there exits a pseudo-polynomial number of sub-instances that can be derived, i.e., $\max_{J \in J}(p_{1,j}) \times \max_{J \in J}(p_{2,j})$. Moreover, for each combination of a and b, a pseudo-polynomial number of sub-jobs can be introduced. These reasons make the lower bound LB_3^N to be sufficiency hard to solve. Therefore, we present a new lower bound procedure that inherits the advantage of ILP_{ass} without having a pseudo-polynomial number of jobs or instances to solve.

3.4.3 Assignment-based lower bound

In the following, we provide a lower bound by destructive improvement for $F2|l_j|C_{max}$. First, we propose a relaxation to the $F2|l_j|C_{max}$ problem. We then introduce an ILP model that permits to solve the derived instance.

The destructive improvement technique used in this lower bound restricts a problem by setting a possible objective function value C and try to contradict the feasibility of this reduced problem. In case of feasibility, we stop the processing.

Otherwise, we repeat the same process after increasing the value of C. As a start, C is equal to the best lower bound value of Section 3.2.

At first, we provide some notation. Given an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$, let $\varrho(1), ..., \varrho(n)$ (resp. $\upsilon(1), ..., \upsilon(n)$) be a job sequence such that jobs are sorted in a non-decreasing order (resp. non-increasing order) of $p_{1,j}$ (resp. $p_{2,j}$), for all j in J. For each pair of jobs (i, j), the notation $j \longrightarrow_m i$ means that j precedes i on M_m, m in $\{1, 2\}$. Moreover, we define the maximum effective delay observed between the two positions k_1 and k_2 on M_1 and M_2 by $\psi(k_1, k_2) = C - \sum_{k=k_2}^{n} p_{2,\upsilon(k)} - \sum_{k=1}^{k_1} p_{1,\varrho(k)}$, k_1 and k_2 in $\{1, ..., n\}$. We denote the distinct values of time delays by $\bar{l}_1, \bar{l}_2, ..., \bar{l}_l, ..., \bar{l}_L$ and we suppose that $\bar{l}_1 > \bar{l}_2 > ... > \bar{l}_l > ... > \bar{l}_L$. Finally, the occurrence number of $\bar{l}_l$ in the time delay vector is given by $nb(\bar{l}_l)$, l in $\{1, ..., L\}$.

Proposition 3.7. *Let $I = (J, p_1, l, p_2)$ be an instance of $F2|l_j|C_{max}$ and C a possible makespan value of I. For each pair of positions k_1 and k_2 in $\{1, ..., n\}$, $\psi(k_1, k_2)$ represents an upper bound on the effective delay observed by a job if it is processed at position k_1 and k_2 on M_1 and M_2, respectively.*

Proof. Let $S = (\sigma, \tau)$ be a feasible schedule of I, where σ_j^{-1} (resp. τ_j^{-1}) represents the position of job j on M_1 (resp. M_2) in the feasible schedule S. The effective delay that is observed by job j in S is equal to $C - \sum_{i \in J: j \longrightarrow_2 i} p_{2,i} - \sum_{i \in J: i \longrightarrow_1 j} p_{1,i} - p_{1,j} - p_{2,j}$. We observe that:

$$\sum_{i \in J: j \longrightarrow_2 i} p_{2,i} + p_{2,j} \geq \sum_{k=\tau_j^{-1}}^{n} p_{2,\upsilon(k)} \text{ and } \sum_{i \in J: i \longrightarrow_1 j} p_{1,i} + p_{1,j} \geq \sum_{k=1}^{\sigma_j^{-1}} p_{1,\varrho(k)}.$$

Thus, we obtain:

$$\begin{aligned} C - \sum_{i \in J: j \longrightarrow_2 i} p_{2,i} - \sum_{i \in J: i \longrightarrow_1 j} p_{1,i} - p_{1,j} - p_{2,j} &\leq C - \sum_{k=\tau_j^{-1}}^{n} p_{2,\upsilon(k)} - \sum_{k=1}^{\sigma_j^{-1}} p_{1,\varrho(k)} \\ &\leq \psi(\sigma_j^{-1}, \tau_j^{-1}). \end{aligned}$$

□

The new lower bound is based on the following reasoning. At each iteration, we suppose that the processing times are relaxed. The aim of this lower bound is to attribute each time delay value to a couple of positions without considering the

processing time constraints of jobs. A time delay can be attributed to a couple of positions (k_1, k_2) if and only if the effective delay that is observed between (k_1, k_2) is greater than or equal to the time delay value, k_1 and k_2 in $\{1, ..., n\}$. To make the resolution process easier, we suppose that the effective delay that is observed between each couple of positions (k_1, k_2) is equal to $\psi(k_1, k_2)$.

Let us define here an integer linear programming model called ILP_{lb} to solve the relaxed instance. ILP_{lb} is based on a non-trivial generalization of the assignment model. The decision variables are defined in order to determine a set of n couples of positions that are able to cover all time delays without considering the processing times of jobs. Precisely, $\xi_{k_1,k_2} = 1$ if the couple of positions (k_1, k_2) is selected and 0 otherwise. The model constraints and objective function are described as follows:

$$\text{Find a set of } n \text{ couples of positions: } (k_1, k_2) \tag{3.51}$$

$$\text{s.t.} \quad \sum_{k_2=1}^{n} \xi_{k_1,k_2} = 1, \qquad \forall k_1 \in \{1, ..., n\} \tag{3.52}$$

$$\sum_{k_1=1}^{n} \xi_{k_1,k_2} = 1, \qquad \forall k_2 \in \{1, ..., n\} \tag{3.53}$$

$$\sum_{(k_1,k_2):\psi(k_1,k_2)\geq \bar{l}_l} \xi_{k_1,k_2} \geq \sum_{\ell=1}^{l} nb(\bar{l}_\ell), \qquad \forall l \in \{1, ..., L\} \tag{3.54}$$

$$\xi_{k_1,k_2} \in \{0, 1\}, \qquad \forall k_1, k_2 \in \{1, ..., n\} \tag{3.55}$$

Constraints (3.52) (resp. (3.53)) ensure that each position on the first (resp. second) machine is only belong to one selected couple of positions. Constraints (3.54) guarantee that each time delay value is covered. Precisely, we verify that there exists at least a unique couple of positions for each time delay, such that its value is less than or equal to effective delay observed between the two positions. Finally, constraints (3.55) specify that the decision variables are boolean. The above model needs $O(n^2)$ variables and $O(n)$ constraints.

The new lower bound for $F2|l_j|C_{max}$ is given in the following proposition.

Proposition 3.8. *Let $I = (J, p_1, l, p_2)$ be an instance of $F2|l_j|C_{max}$ and I^r its relaxed instance as it is described above. It holds that:*

$$C^*_{max}(I) \geq ILP_{lb}(I^r).$$

Therefore, $LB_4^N = ILP_{lb}(I^r)$ is a valid lower bound on I.

3.5 Computational results

In this section, we report the results of a computational study that aims at assessing the performance of the proposed lower bounds. First, we study the performance of the combinatorial lower bounds. We then analyze the impact of the processing time intervals on the performance of LB_4^N. Finally, we compare the best combinatorial lower bound to the new linear-programming based lower bounds.

Note that all the discussed MIP and ILP models were coded in C++ and compiled under CentOS 6.6. Moreover, we used CPLEX 12.6 to implement them. The experiments were conducted on an Intel(R) Xeon(R) @ 2.67GHz processor.

We test all the discussed lower bounds on a set of six classes A-F that was proposed by [Dell'Amico, 1996]. The details of these classes are given as follows:

- Class A: $p_{1,j},\ p_{2,j}$ and l_j are randomly chosen in $[1, \ldots, 100]$.
- Class B: $p_{1,j},\ p_{2,j}$ are randomly chosen in $[1, \ldots, 100]$ and l_j in $[1, \ldots, 200]$.
- Class C: $p_{1,j},\ p_{2,j}$ are randomly chosen in $[1, \ldots, 100]$ and l_j in $[1, \ldots, 500]$.
- Class D: $p_{1,j},\ p_{2,j}$ are randomly chosen in $[1, \ldots, 200]$ and l_j in $[1, \ldots, 100]$.
- Class E: $p_{1,j},\ l_j$ are randomly chosen in $[1, \ldots, 100]$ and $p_{2,j}$ in $[1, \ldots, 200]$.
- Class F: $p_{2,j},\ l_j$ are randomly chosen in $[1, \ldots, 100]$ and $p_{1,j}$ in $[1, \ldots, 200]$.

Furthermore, preliminary computational results conducted on the literature classes show that previous lower bounds give poor performance when time delays are very large compared to processing times. To that aim, we introduced two new classes, 1 and 2 where the processing times on M_1 and M_2 and the time delays are randomly generated between $[1, ..., p_{1,max}]$, $[1, ..., p_{2,max}]$ and $[1, ..., l_{max}]$, respectively, with $p_{1,max} = p_{2,max} = 20$ and $l_{max} = \frac{n}{2}10$ (resp. $p_{1,max} = p_{2,max} = 100$ and $l_{max} = \frac{n}{2}100$) for Class 1 (resp. Class 2). For each class, the number of jobs is $n = 10,\ 30,\ 50,\ 100,\ 150$ and 200. For each combination of class and number of jobs, 10 instances were randomly generated.

3.5.1 Combinatorial lower bounds performance

In order to present a detailed image of the performance of the lower bounds described in Section 3.2, a pairwise comparison between them is given in Table 3.7 and Table 3.8. In these two tables, we illustrate for each pair of lower bounds LB_{row} and LB_{col},

which are displayed in some given row and column, respectively, the percentage of times where $LB_{col} > LB_{row}$. We observe on Classes A-F that $LB_{tra,3}$ outperforms $LB_{bas,2}$ in 10.83% of instances and $LB_{tra,1}$ in 26.39% of instances. In the same way, on the new Classes 1-2, we notice that $LB_{tra,3}$ provides a much better performance than the rest, since it outperforms all of them on at least 52.5% of instances.

To get a better picture of the lower bounds performance, we provide in Table 3.10 the average percentage deviation (over the instances of each class) with respect to the maximal lower bound value, that is delivered by the considered lower bounds. Note that the average CPU time of all lower bounds is less than 10^{-2} seconds. From Table 3.10, we observe that the average gap significantly depend on the classes. On one hand, $LB_{tra,3}$ exhibits a maximum average gap of 0.15% on all Classes A-F and Class 1. However, for the instances of Class 2, its average gap jumps to 5.9%. On the other hand, LB_{split} and $LB_{res,2}$ present a much better performance on the new classes. Indeed, the average gap of LB_{split} (resp. $LB_{res,2}$) is equal to 8.94% and 2.36% (resp. 9.62% and 2.24%) on Class 1 and Class 2, respectively.

Hereafter, we denote the best combinatorial lower bound value of Section 3.2 by $LB_{best} = \max(LB_{split}, LB_{res,2}, LB_{tra,3})$.

Table 3.7 – Pairwise comparison between lower bounds on Classes A-F.

	$LB_{bas,1}$	$LB_{bas,2}$	LB_{split}	$LB_{res,1}$	$LB_{res,2}$	$LB_{tra,1}$	$LB_{tra,2}$	$LB_{tra,3}$
$LB_{bas,1}$	-	8.33	1.94	8.33	1.67	8.61	12.78	15.28
$LB_{bas,2}$	0.00	-	0.56	0.00	0.28	3.33	6.11	10.83
LB_{split}	98.06	99.44	-	99.44	63.33	98.06	98.06	99.44
$LB_{res,1}$	0.00	0.00	0.56	-	0.28	3.33	6.11	10.83
$LB_{res,2}$	98.33	99.72	36.67	99.72	-	98.33	98.33	99.72
$LB_{tra,1}$	17.22	23.33	1.94	23.33	1.67	-	19.17	26.39
$LB_{tra,2}$	0.28	7.22	1.94	7.22	1.67	0.00	-	9.44
$LB_{tra,3}$	0.00	0.00	0.56	0.00	0.28	0.00	0.00	-

3.5.2 The impact of the processing time intervals on the performance of LB_4^N

In this phase, we focus our attention on LB_4^N. In fact, we performed an analysis of its performance according to a parameter $\alpha \in [0, 1]$ that we used to fix the processing time intervals while generating new instances. The values of α that we tested are $k \times 0.05$, k in $\{0, ..., 20\}$. For each value of α, we derived from every instance

Table 3.8 – Pairwise comparison between lower bounds on Classes 1-2.

	$LB_{bas,1}$	$LB_{bas,2}$	LB_{split}	$LB_{res,1}$	$LB_{res,2}$	$LB_{tra,1}$	$LB_{tra,2}$	$LB_{tra,3}$
$LB_{bas,1}$	-	60.83	46.67	60.83	45.83	29.17	42.50	75.00
$LB_{bas,2}$	0.00	-	39.17	0.00	38.33	7.50	10.00	52.50
LB_{split}	52.50	60.83	-	60.83	46.67	52.50	53.33	61.67
$LB_{res,1}$	0.00	0.00	39.17	-	38.33	7.50	10.00	52.50
$LB_{res,2}$	54.17	61.67	50.00	61.67	-	54.17	54.17	61.67
$LB_{tra,1}$	26.67	67.50	47.50	67.50	45.83	-	25.00	72.50
$LB_{tra,2}$	13.33	60.00	46.67	60.00	45.83	0.00	-	65.00
$LB_{tra,3}$	0.00	0.00	38.33	0.00	38.33	0.00	0.00	-

$I = (J, p_1, l, p_2)$ of the early defined classes ten new instances where the precessing times on M_1 (resp. on M_2) were randomly generated between $[(1-\alpha)\sum_{j\in J}(\frac{p_{1,j}}{n}), (1+\alpha)\sum_{j\in J}(\frac{p_{1,j}}{n})]$ (resp. $[(1-\alpha)\sum_{j\in J}(\frac{p_{2,j}}{n}), (1+\alpha)\sum_{j\in J}(\frac{p_{2,j}}{n})]$). Moreover, the time delays were drawn within the same interval as in the original class of I. We provide in Figure 3.4 the average percentage evolution of $\frac{LB_4^N - LB_{best}}{LB_4^N}$ over the instances with the same value of α referred to as Gap_{LB}. Recall that LB_4^N is a lower bound by destructive improvement that uses the value of LB_{best} as a start solution.

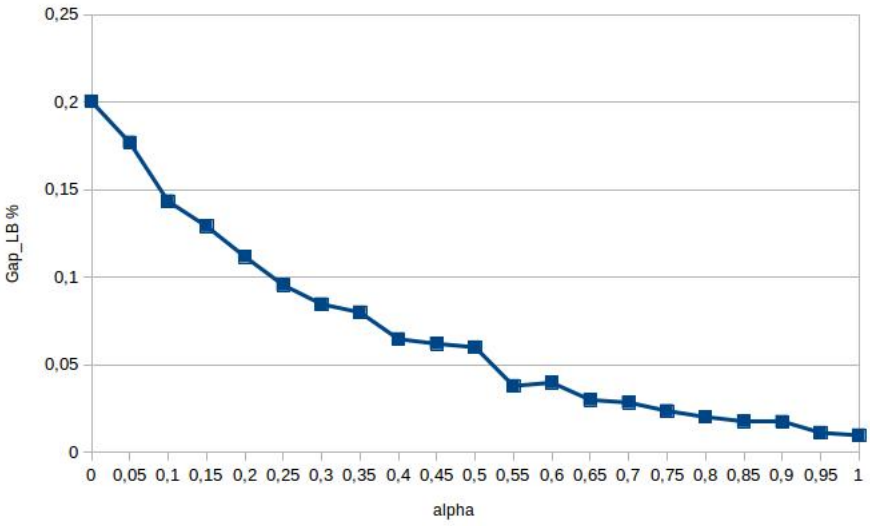


Figure 3.4 – The evolution of Gap_{LB} according to α.

We notice from Figure 3.4 that the larger is the value of α, the lower is the value of Gap_{LB}. Note that for $\alpha = 0$, the processing times are constant on each machine, and then LB_4^N is optimal in this case. However, when $\alpha > 0$, the bigger is α, the lower is the performance of LB_4^N. In fact, when the value of α increases, the intervals of processing time are large and the performance of LB_4^N decreases. As a consequence, LB_4^N yields a good performance when the intervals of processing time

are small.

3.5.3 Linear programming-based lower bounds comparison

Here, we provide a comparison between six lower bounds including LB_{best}, LB_1^N, LB_2^N and LB_4^N that aims to evaluate their performance. The lower bound LB_5^N (resp. LB_6^N) is obtained after solving the linear relaxation of the MIP model (3.16)-(3.23) (resp. (3.24)-(3.33)). In Table 3.11, we provide for each lower bound the following values:

- **Gap**: the average percentage deviation over the solved instances with the same class and size with respect to the best lower bound value. The number between parenthesis stands for the number of unsolved instances within the given 600 seconds.
- **Time**: the average computational time in seconds for the solved instances.

From Table 3.11, we make the following observations:

- LB_2^N exhibits good results within a short computational time on all classes.
- LB_2^N shows an excellent performance on Class 1 and Class 2, where it strictly dominates LB_{best} and LB_4^N. Precisely, LB_2^N provides a **Gap** of 0.01% and 0.02% on Class 1 and Class 2 compared to 1.51% and 8.32% for LB_{best}, respectively.
- LB_2^N offers a good trade-off between effectiveness and efficiency compared to LB_1^N. In fact, for $n < 100$, LB_2^N achieves the same values as LB_1^N within a very short time (0.03 seconds compared to 15.35 seconds), except on Classes C, 1 and 2. In addition, LB_1^N fails to solve 232 instances out of 240 within 600 seconds when $n \geq 100$, while LB_2^N solves all instances in an average time of 3.77 seconds.
- From the other hand, we remark that LB_4^N provides a good overall performance compared to LB_{best}. The **Gap** of LB_4^N is better than LB_{best}'s one in 6 occasions especially when $n = 10$ for Classes B, C and 2. Nevertheless, we observe that LB_4^N fails to solve 12 instances within the given time limit.

From Table 3.11, we observe that all lower bounds exhibit poor results on the new classes of instances (Class 1 and 2). This can be explained by the fact that the time delays are very large compared to processing times. Before proceeding any further, we define for each instance I of $F2|l_j|C_{max}$ the following set of pairs of jobs

$\theta(I) = \{(i,j) \in J^2; i < j \mid \ l_i \leq l_j + \max(p_{1,j}, p_{2,j})$ and $l_i \leq l_i + \max(p_{1,i}, p_{2,i})\}$. Moreover, we denote by $d(I)$ the density of the set $\theta(I)$ that is equal to $d(I) = \frac{2|\theta(I)|}{n(n-1)}$. Interestingly, we observe that the density $d(I)$ is small on the two Classes 1-2.

In what follows, we present a study to detect the density values for which the instances are hard to solve. We start by computing the average density value of each class. We then present in Figure 3.5 the evolution of the average percentage deviation of LB_2^N with respect to the best upper bound value of [Dell'Amico, 1996] over the average density of each class referred to as Gap_{UB}. Thus, each class is represented by a point that displays its average density and its average gap.

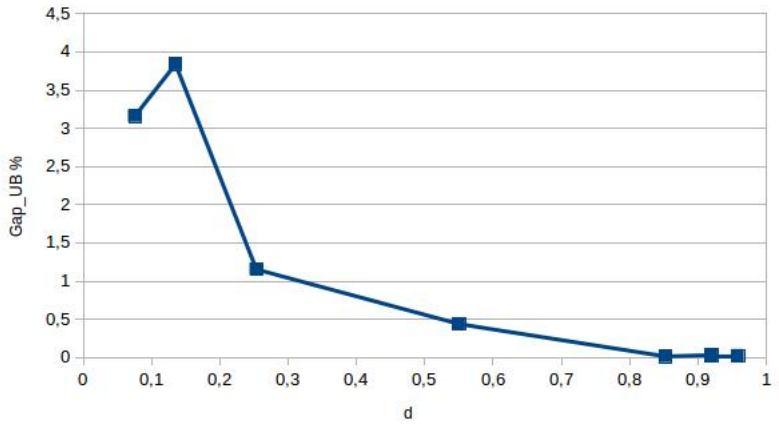


Figure 3.5 – The evolution of Gap_{UB} according to d.

From Figure 3.5, we notice the presence of three groups of classes:

- $0.08 \leq d \leq 0.25$: These classes seem to be the hardest to solve. The average gap is higher than 1% and reaches 3.84% and 3.16% for Class 1 and Class 2, respectively. From the lowest to the highest density, the three classes that appear are Class 2, Class 1 and Class C. The common feature between these classes is that the interval of the time delays is large compared to processing ones.

- $0.25 < d < 0.85$: only Class B appears in this interval. Its average gap is equal to 0.44% and so it is easier to solve compared to the first group within an average CPU time of 1.12 seconds.

- $d \geq 0.85$: These classes are the easiest to solve. LB_2^N provides the optimal solution in the majority of cases where the average gap is equal to 0.03%, 0.02%, 0.02% and 0.01% for Class E, Class F, Class D and Class A, respectively.

In order to complete Figure 3.5, we derived four new classes of instances with small density $d \leq 0.2$. The details of these new classes are provided in Table 3.9. Moreover, we present in Figure 3.6 the complete version of the results. The order of classes according to the non-decreasing order of average density is 6, 5, 2, 4, 1, 3, C, B, A, E, F and D.

Table 3.9 – Generation of new classes.

Class	3	4	5	6
$p_{1,max}$	100	100	100	100
$p_{2,max}$	100	100	100	100
l_{max}	$50\frac{n}{2}$	$90\frac{n}{2}$	$120\frac{n}{2}$	$1000\frac{n}{2}$

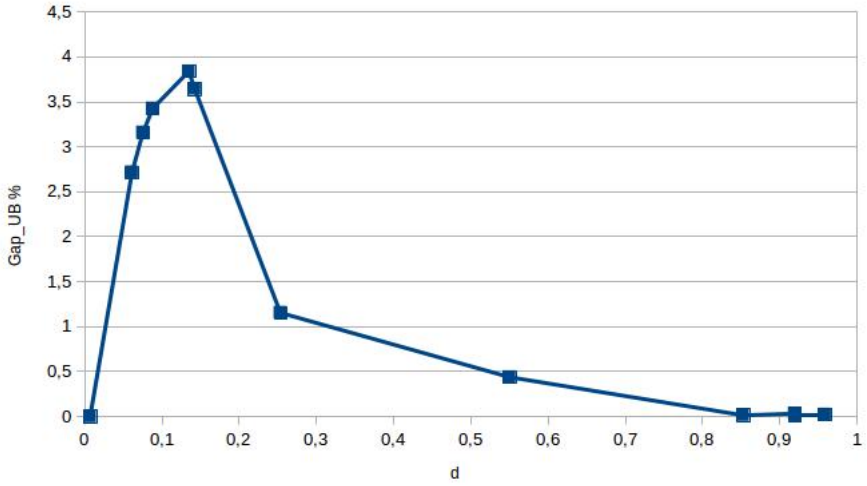


Figure 3.6 – The evolution of Gap_{UB} according to d.

From Figure 3.6, we observe that:

- The resulting graph has a Gaussian shape.
- The classes are easy to solve when $d \simeq 0$ and $d \geq 0.85$.
- The hardest classes are those with $0.07 \leq d \leq 0.14$.

3.6 Conclusion

In this paper, we addressed the two-machine flow-shop problem with time delays. A theoretical analysis of the state of the art lower bounds was given. Precisely, we proposed a new classification scheme of the different lower bounds and we enhanced

a time delay-based one. Then, a set of dominance relationships between them was proved. Moreover, we presented a set of mathematical formulations for $F2|l_j|C_{max}$. Interestingly, we identified the good performance of a linear ordering variable-based formulation. For which, we proposed a set of valid inequalities. Three linear programming-based lower bounds were proposed for $F2|l_j|C_{max}$. The experimental study shows the efficiency and the effectiveness of the proposed approaches. It is interesting to notice the good performance of the new lower bounds that yield the best values on all discussed instances.

Table 3.10 – Combinatorial lower bounds performance per class.

Class	n	$LB_{bas,1}$ Gap	$LB_{bas,2}$ Gap	LB_{split} Gap	$LB_{res,1}$ Gap	$LB_{res,2}$ Gap	$LB_{tra,1}$ Gap	$LB_{tra,2}$ Gap	$LB_{tra,3}$ Gap
A	10	0.24	0.24	20.41	0.24	22.75	1.42	0.18	0.00
	30	0.04	0.04	28.13	0.04	28.24	0.33	0.00	0.00
	50	0.04	0.00	30.71	0.00	30.73	0.03	0.03	0.00
	100	0.04	0.00	33.89	0.00	33.99	0.04	0.04	0.00
	150	0.00	0.00	32.59	0.00	32.05	0.01	0.00	0.00
	200	0.00	0.00	33.70	0.00	33.81	0.02	0.00	0.00
	AVG	0.06	0.05	29.91	0.05	30.26	0.31	0.04	0.00
B	10	0.39	0.16	14.62	0.16	17.32	1.50	0.32	0.00
	30	0.09	0.05	24.88	0.05	25.25	0.19	0.04	0.00
	50	0.13	0.05	29.02	0.05	29.02	0.10	0.07	0.00
	100	0.01	0.01	33.02	0.01	33.11	0.07	0.00	0.00
	150	0.00	0.00	31.98	0.00	31.46	0.04	0.00	0.00
	200	0.00	0.00	33.21	0.00	33.33	0.00	0.00	0.00
	AVG	0.11	0.04	27.79	0.04	28.25	0.32	0.07	0.00
C	10	11.48	2.08	6.95	2.08	8.42	11.88	11.69	0.93
	30	0.28	0.01	18.60	0.01	18.10	0.39	0.27	0.00
	50	0.10	0.06	23.59	0.06	23.78	0.13	0.05	0.00
	100	0.01	0.01	30.10	0.01	30.23	0.06	0.00	0.00
	150	0.00	0.00	29.98	0.00	29.44	0.03	0.00	0.00
	200	0.01	0.00	31.78	0.00	31.88	0.01	0.01	0.00
	AVG	1.98	0.36	23.50	0.36	23.64	2.08	2.00	0.15
D	10	0.26	0.26	26.58	0.26	26.28	0.03	0.00	0.00
	30	0.03	0.01	31.13	0.01	32.34	0.12	0.03	0.00
	50	0.04	0.00	32.48	0.00	32.73	0.05	0.04	0.00
	100	0.00	0.00	34.41	0.00	33.51	0.00	0.00	0.00
	150	0.00	0.00	35.16	0.00	35.04	0.02	0.00	0.00
	200	0.00	0.00	33.73	0.00	33.88	0.00	0.00	0.00
	AVG	0.06	0.04	32.25	0.04	32.30	0.04	0.01	0.00
E	10	0.04	0.02	47.85	0.02	39.82	0.01	0.00	0.00
	30	0.05	0.00	49.13	0.00	43.32	0.05	0.05	0.00
	50	0.05	0.00	56.28	0.00	47.87	0.05	0.04	0.00
	100	0.02	0.00	56.75	0.00	48.49	0.02	0.02	0.00
	150	0.00	0.00	55.81	0.00	48.69	0.00	0.00	0.00
	200	0.00	0.00	57.95	0.00	49.39	0.00	0.00	0.00

Table 3.10 - Continued

Class	n	$LB_{bas,1}$ Gap	$LB_{bas,2}$ Gap	LB_{split} Gap	$LB_{res,1}$ Gap	$LB_{res,2}$ Gap	$LB_{tra,1}$ Gap	$LB_{tra,2}$ Gap	$LB_{tra,3}$ Gap
	AVG	0.03	0.00	53.96	0.00	46.26	0.02	0.02	0.00
F	10	0.35	0.17	40.82	0.17	35.70	1.17	0.14	0.00
	30	0.00	0.00	51.50	0.00	45.57	0.08	0.00	0.00
	50	0.03	0.00	53.84	0.00	45.79	0.08	0.03	0.00
	100	0.00	0.00	58.34	0.00	49.59	0.01	0.00	0.00
	150	0.00	0.00	56.99	0.00	48.75	0.02	0.00	0.00
	200	0.00	0.00	58.44	0.00	49.74	0.01	0.00	0.00
	AVG	0.06	0.03	53.32	0.03	45.86	0.23	0.03	0.00
1	10	2.30	1.38	9.41	1.38	11.21	2.36	1.28	0.39
	30	0.42	0.15	8.31	0.15	9.30	0.52	0.30	0.00
	50	0.32	0.10	10.16	0.10	11.04	0.62	0.23	0.00
	100	0.44	0.04	10.28	0.04	9.96	0.41	0.39	0.00
	150	0.15	0.04	7.74	0.04	8.08	0.14	0.13	0.00
	200	0.21	0.02	7.72	0.02	8.11	0.20	0.19	0.00
	AVG	0.64	0.29	8.94	0.29	9.62	0.71	0.42	0.07
2	10	11.48	2.08	6.95	2.08	8.42	11.88	11.69	0.93
	30	9.67	4.76	2.88	4.76	1.27	10.99	10.84	4.63
	50	9.30	5.31	1.37	5.31	1.73	10.48	10.22	5.20
	100	10.21	6.58	0.81	6.58	0.39	11.87	11.83	6.55
	150	13.47	9.09	0.89	9.09	0.75	14.25	14.19	9.08
	200	12.04	9.04	1.24	9.04	0.87	12.39	12.35	9.02
	AVG	11.03	6.14	2.36	6.14	2.24	11.98	11.85	5.90

Table 3.11 – Lower bounds performance.

		LB_{best}		LB_1^N		LB_2^N		LB_4^N		LB_5^N		LB_6^N	
Class	**n**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**
Class A	10	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	26.23	0.01	71.81	0.00
	30	0.00	0.00	0.00	1.73	0.00	0.01	0.00	0.02	13.41	1.58	89.01	0.02
	50	0.00	0.00	0.00	20.35	0.00	0.05	0.00	0.06	5.73 (1)	104.40	93.05	0.07
	100	0.00	0.00	-	-	0.00	0.48	0.00	0.51	-	-	96.30	0.40
	150	0.00	0.00	-	-	0.00	1.42	0.00	1.93	-	-	97.54	1.43
	200	0.00	0.00	-	-	0.00	3.77	0.00	4.91	-	-	98.13	3.91
	AVG	0.00	0.00	0.00	7.36	0.00	0.95	0.00	1.24	15.45	32.95	90.98	0.97
Class B	10	0.15	0.00	0.00	0.01	0.00	0.00	0.13	0.00	27.71	0.01	61.05	0.01
	30	0.00	0.00	0.00	3.18	0.00	0.02	0.00	0.02	13.28	1.92	83.50	0.03
	50	0.00	0.00	0.00	41.54	0.00	0.06	0.00	0.07	5.59	44.65	89.51	0.06
	100	0.00	0.00	-	-	0.00	0.48	0.00	0.68	-	-	94.55	0.43
	150	0.00	0.00	-	-	0.00	1.84	0.00	2.84	-	-	96.26	0.99
	200	0.00	0.00	-	-	0.00	4.31	0.00	5.67	-	-	97.19	2.40
	AVG	0.03	0.00	0.00	14.91	0.00	1.12	0.02	1.55	15.53	15.53	87.01	0.65
Class C	10	3.51	0.00	0.00	0.01	0.09	0.00	3.35	0.01	42.38	0.02	31.65	0.01
	30	0.00	0.00	0.00	3.19	0.00	0.03	0.00	0.03	15.62	1.59	67.52	0.03
	50	0.00	0.00	0.00	108.75	0.00	0.10	0.00	0.10	6.28 (1)	44.56	79.00	0.08
	100	0.00	0.00	-	-	0.00	0.49	0.00	1.06	-	-	88.79	0.45
	150	0.00	0.00	-	-	0.00	1.99	0.00	6.11	-	-	92.45	0.78
	200	0.00	0.00	-	-	0.00	5.72	0.00	6.58	-	-	94.30	2.08

Table 3.11 - Continued

		LB_{best}		LB_1^N		LB_2^N		LB_4^N		LB_5^N		LB_6^N	
Class	**n**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**
	AVG	0.58	0.00	0.00	37.32	0.02	1.39	0.56	2.32	21.95	14.38	75.62	0.57
Class D	10	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	24.06	0.01	76.76	0.00
	30	0.00	0.00	0.00	1.21	0.00	0.01	0.00	0.02	13.05	1.48	91.40	0.03
	50	0.00	0.00	0.00	18.30	0.00	0.03	0.00	0.09	7.91	109.06	94.84	0.07
	100	0.00	0.00	0.00 (7)	548.68	0.00	0.17	0.00	0.55	-	-	97.24	0.46
	150	0.00	0.00	-	-	0.00	0.69	0.00	2.04	-	-	98.12	0.78
	200	0.00	0.00	-	-	0.00	1.55	0.00	4.06	-	-	98.59	2.05
	AVG	0.00	0.00	0.00	55.80	0.00	0.41	0.00	1.13	15.01	36.85	92.83	0.57
Class E	10	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01	21.51	0.01	75.84	0.00
	30	0.00	0.00	0.00	1.10	0.00	0.01	0.00	0.03	7.43	1.46	90.70	0.03
	50	0.00	0.00	0.00	17.38	0.00	0.04	0.00	0.10	4.33 (1)	105.93	94.61	0.05
	100	0.00	0.00	0.00 (9)	505.17	0.00	0.25	0.00	0.78	-	-	97.21	0.34
	150	0.00	0.00	-	-	0.00	1.05	0.00	2.50	-	-	98.03	0.91
	200	0.00	0.00	-	-	0.00	2.68	0.00	4.75	-	-	98.56	1.90
	AVG	0.00	0.00	0.00	22.26	0.00	0.67	0.00	1.36	11.32	33.38	92.49	0.54
Class F	10	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.01	54.06	0.01	80.81	0.03
	30	0.00	0.00	0.00	1.37	0.00	0.01	0.00	0.04	52.77	1.49	93.67	1.38
	50	0.00	0.00	0.00	9.78	0.00	0.04	0.00	0.10	50.68	77.66	96.25	3.04
	100	0.00	0.00	0.00 (6)	404.65	0.00	0.25	0.00	0.69	-	-	98.39	0.01
	150	0.00	0.00	-	-	0.00	1.01	0.00	2.46	-	-	98.78	0.07

Table 3.11 - Continued

		LB_{best}		LB_1^N		LB_2^N		LB_4^N		LB_5^N		LB_6^N	
Class	**n**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**	**Gap**	**Time**
	200	0.00	0.00	-	-	0.00	2.62	0.00	4.13	-	-	99.06	0.54
	AVG	0.00	0.00	0.00	50.89	0.00	0.65	0.00	1.24	52.51	26.38	94.49	0.84
Class 1	10	0.84	0.00	0.00	0.01	0.00	0.00	0.84	0.01	34.39	0.01	51.03	0.00
	30	1.30	0.00	0.00	3.62	0.03	0.04	1.30	0.02	14.32	22.55	54.04	0.02
	50	1.38	0.00	0.00	62.99	0.02	0.14	1.38	0.09	11.60	50.59	50.07	0.06
	100	0.98	0.00	-	-	0.00	1.29	0.98	1.08	-	-	52.76	0.34
	150	2.17	0.00	-	-	0.00	5.16	2.17	4.96	-	-	52.13	0.91
	200	2.37	0.00	-	-	0.00	15.29	2.37	9.44	-	-	52.35	2.20
	AVG	1.51	0.00	0.00	22.21	0.01	3.65	1.51	2.60	20.10	24.38	52.06	0.59
Class 2	10	3.51	0.00	0.00	0.01	0.09	0.00	3.35	0.01	42.38	0.01	31.65	0.01
	30	8.06	0.00	0.00	2.51	0.00	0.03	7.91	0.17	31.31	1.40	25.56	0.02
	50	9.36	0.00	0.00	71.56	0.00	0.15	9.32	0.87	25.60	64.92	24.20	0.07
	100	10.02	0.00	-	-	0.00	1.65	10.00	36.93	-	-	23.03	0.34
	150	9.35	0.00	-	-	0.00	7.97	8.99 (5)	15.38	-	-	24.68	0.96
	200	9.63	0.00	-	-	0.00	28.47	9.15 (7)	69.06	-	-	22.04	2.62
	AVG	8.32	0.00	0.00	24.69	0.02	6.38	7.88	13.83	33.10	22.11	25.19	0.67

Exact methods for $F2|l_j|C_{max}$

Contents

4.1 Introduction

In this chapter, we investigate exact methods for the $F2|l_j|C_{max}$ problem. As far we know, a unique exact method based on the branch-and-bound scheme was introduced for $F2|l_j|C_{max}$ by [Dell'Amico, 1996]. Furthermore, several authors addressed numerous variants of the problem by providing exact algorithms based on a branch-and-bound scheme. A non-exhaustive list includes the two-machine flow-shop problem with time delays and unit-time operations ([Moukrim et al., 2014]), the problem of minimizing the total completion time in a two-machine permutation flow-shop environment with time delays ([Msakni et al., 2016]) and the m-machine permutation flow-shop problem with minimal and maximal time lags ([Fondrevelle et al., 2006]). For more details, we refer the reader to a recent survey of [Emmons and Vairaktarakis, 2013].

The contribution of this chapter consists in:

- Introducing a branch-and-cut algorithm based on a linear ordering variable-based model. This method includes the consideration of a set of dominance rules and clique-based valid inequalities. Moreover, a critical path-based branching scheme is considered. We also apply a new heuristic method and a node pruning procedure.

- Proposing a branch-and-bound algorithm. For which, we extend the best lower bounds to be used inside the search tree and we introduce a local search-based upper bound and three dominance rules. One of the dominance rules includes the consideration of the no-good list technique of [Schiex and Verfaillie, 1993].

- Providing extensive experiments that were carried out on different classes of instances. We present a comprehensive experimental analysis of the combinatorial lower bounds within our branch-and-bound algorithm. We then study the impact of the proposed components on the performance of the two exact methods. Finally, a comparison with the literature exact method is given.

The remainder of this chapter is organized as follows. Section 4.2 provides a representation of the problem as a disjunctive graph. We also present a set of dominance rules. In Section 4.3, we introduce a linear programming-based exact method. In Section 4.4, we provide a detailed description of the branch-and-bound algorithm. Section 4.5 displays the results of a computational study that was carried out on randomly generated instances. Finally, the last section provides concluding remarks.

4.2 Preliminaries

In this section, we represent the problem using a disjunctive graph. Then, we introduce new dominance rules.

4.2.1 Problem representation

Here, we represent each instance of $F2|l_j|C_{max}$ using a disjunctive graph model of [Roy and Sussmann,]. This graph is used to implement the critical path-based branching scheme for our branch-and-cut algorithm. Given an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$, we define by $G = (V, C \cup D)$ a disjunctive graph on I as follows. The set V contains the nodes where each job is modeled using three nodes:

- one node for each operation having a weight that is equal to the processing time of the operation.
- one node for the time delay having a weight that is equal to the time delay value.

In addition, two extra nodes are defined including a source and a sink. A weight that is equal to zero is attributed to both extra nodes.

The set C contains the conjunctive arcs that reflect the executing order of operations and time delay of the same job. The executing order of operations and time delay of job j can be given by four conjunctive arcs:

- an arc from the source to $O_{1,j}$'s node,
- an arc from $O_{1,j}$'s node to the time delay node of j,
- an arc from the time delay node of j to $O_{2,j}$'s node,
- finally an arc from $O_{2,j}$'s node to the sink.

The set D contains the undirected arcs, referred to as disjunctive arcs. These arcs link every pair of operations that are executed on the same machine. The precedence relationships between operations on the same machine can be defined using a selection of disjunctive arcs. If operation $O_{m,j}$ is processed before operation $O_{m,i}$ on M_m, then this can be done by transforming the undirected arc between $O_{m,j}$ and $O_{m,i}$ into directed one, i and j in J; m in $\{1, 2\}$. The set of all direct disjunctive arcs are referred to as a selection.

Every schedule S of $F2|l_j|C_{max}$ can be given by a selection φ that is obtained after turning the undirected arcs of D into direct ones. This schedule is called feasible if and only if all disjunctive arcs are fixed and the obtained graph $G(S) = (V, C \cup \varphi)$ is acyclic. Moreover, the makespan value of S is given by the length of the path with the highest cumulative weight from the source to the sink. This path is called *critical path.*

4.2.2 Dominance rules

Hereafter, we present two dominance rules that allow to reduce the solution research space. First, we recall a dominance rule that was presented in Chapter 3. We then introduce a new one. The two dominance rules are given by the following two lemmas.

Lemma 4.1. *Consider an instance I of $F2|l_j|C_{max}$ and two jobs i and j where $p_{1,j} + l_j \leq p_{1,i} + l_i$, $l_i \leq l_j + p_{2,j}$ and $p_{1,j} \leq p_{2,j}$. If a schedule where i and j are adjacent on M_1 exists, then j must be executed first.*

Proof. See Lemma 3.1. □

Lemma 4.2. *Consider two jobs i, j in J where $p_{1,j} \leq p_{1,i}$, $p_{2,i} \leq p_{2,j}$, $p_{1,j} + l_j \leq p_{1,i} + l_i$ and $p_{2,j} + l_j \geq p_{2,i} + l_i$, then j must precede i on both machines.*

Proof. Let σ_1 be a sub-sequence on M_1 ($J_{\sigma_1} \subset J \backslash \{i, j\}$) and S an arbitrary schedule of $\Omega_{\sigma_1 \oplus (i)}$ where job i is executed directly after σ_1 on M_1. On the basis of S, we can derive a schedule $S^{'}$ of $\Omega_{\sigma_1 \oplus (j)}$ such that j precedes i on M_2, without increasing the makespan. For $S^{'}$, the job sequence on M_1 is obtained by simply interchanging i and j and keeping the same executing order for the rest of the jobs, like in S. Therefore, it holds that $t_{1,j}(S^{'}) = t_{1,i}(S)$ and $t_{1,i}(S^{'}) + p_{1,i} = t_{1,j}(S) + p_{1,j}$. Since $p_{1,j} \leq p_{1,i}$, we have that $t_{1,v}(S^{'}) \leq t_{1,v}(S)$, for all v in $J \backslash \{i\}$. Moreover, we identify two cases:

Case 1: j precedes i on M_2.

In this case, the job sequence on M_2 of $S^{'}$ is similar to that of S, where $t_{2,v}(S^{'}) = t_{2,v}(S)$, for all v in J. Since $t_{1,v}(S^{'}) \leq t_{1,v}(S)$ for all v in $J \backslash \{i\}$, v is at the disposal of M_2 before its starting time. Moreover, it holds that:

$$t_{1,i}(S^{'}) + p_{1,i} + l_i \quad = \quad t_{1,j}(S) + p_{1,j} + l_i.$$

Since $l_i \leq l_j + p_{2,j}$, we obtain:

$$\begin{aligned} t_{1,i}(S^{'}) + p_{1,i} + l_i &\leq t_{1,j}(S) + p_{1,j} + l_j + p_{2,j} \\ &\leq t_{2,j}(S) + p_{2,j} \\ &\leq t_{2,i}(S^{'}). \end{aligned}$$

On the basis of the above remarks, we conclude that all jobs are available for processing on M_2 before their starting times in $S^{'}$.

Case 2: i precedes j on M_2.

In this case, the job sequence on M_2 of $S^{'}$ is derived from that of S by simply interchanging i and j on M_2, where the same jobs are executed between i and j as in S. Then, it holds that $t_{2,j}(S^{'}) = t_{2,i}(S)$ and $t_{2,i}(S^{'}) + p_{2,i} = t_{2,j}(S) + p_{2,j}$. For each job v in $J\backslash\{i,j\}$, we observe that $t_{1,v}(S^{'}) \leq t_{1,v}(S)$ and $t_{2,v}(S^{'}) \geq t_{2,v}(S)$ since $p_{1,j} \leq p_{1,i}$ and $p_{2,j} \geq p_{2,i}$. Therefore, each job of $J\backslash\{i,j\}$ is available on M_2 before its starting time. We recall that:

$$t_{1,j}(S^{'}) + p_{1,j} + l_j = t_{1,i}(S) + p_{1,j} + l_j.$$

Since $p_{1,j} + l_j \leq p_{1,i} + l_i$, it holds that:

$$\begin{aligned} t_{1,j}(S^{'}) + p_{1,j} + l_j &\leq t_{1,i}(S) + p_{1,i} + l_i \\ &\leq t_{2,i}(S) \\ &\leq t_{2,j}(S^{'}). \end{aligned}$$

Moreover, since $l_i \leq l_j + p_{2,j} - p_{2,i}$, we have:

$$\begin{aligned} t_{1,i}(S^{'}) + p_{1,i} + l_i &= t_{1,j}(S) + p_{1,j} + l_i \\ &\leq t_{1,j}(S) + p_{1,j} + l_j + p_{2,j} - p_{2,i} \\ &\leq t_{2,j}(S) + p_{2,j} - p_{2,i} \\ &\leq t_{2,i}(S^{'}). \end{aligned}$$

On the basis of the above remarks, we conclude that all jobs are available for processing on M_2 before their starting times in $S^{'}$. □

4.2.3 Pre-processing procedure

In this section, we introduce a pre-processing procedure based on that of [Carlier and Pinson, 1990]. This procedure is implemented in the two exact methods. It aims at fixing additional precedence relationships between operations on the two machines. The pre-processing procedure can play a key role in improving the performance of the two exact methods. In fact, a quick increase of the fixed precedence relationships is substantive for the two exact methods for many reasons. First, we can reduce the search tree by pruning nodes that differ with at least one of the determined precedence relationships. Moreover, the additional precedence information can be used to enhance the lower bounds.

For each job j in J and m in $\{1, 2\}$, we introduce two sets of jobs $\varphi_{m,j}$ and $\psi_{m,j}$. The set $\varphi_{m,j}$ (resp. $\psi_{m,j}$) contains the jobs of J that should be processed before (resp. after) j on M_m.

The pre-processing method is given by the following lemma:

Lemma 4.3. *([Carlier and Pinson, 1990]) Let I be an instance of $F2|l_j|C_{max}$ and UB an upper bound on the makespan of I. For each pair of jobs i and j, if we have:*

$$\sum_{v \in \varphi_{1,j}} p_{1,v} + p_{1,j} + l_j + p_{2,j} + p_{2,i} + \sum_{v \in \psi_{2,i}} p_{2,v} \geq UB,$$

then i is a predecessor of j on M_2. Moreover, if it holds that:

$$\sum_{v \in \varphi_{1,i}} p_{1,v} + p_{1,i} + p_{1,j} + l_j + p_{2,j} + \sum_{v \in \psi_{2,j}} p_{2,v} \geq UB,$$

then j is a predecessor of i on M_1.

4.3 Mixed Integer Linear Programming-based exact method

In this section, we start by modeling our problem using a MIP formulation. Then, we present the different components proposed for the branch-and-cut algorithm.

4.3.1 MIP formulation

Throughout this chapter, we use the linear ordering variable-based MIP formulation that was proposed in Section 3.3. This choice can be explained by the absence of

pseudo-polynomial number of variables in this model where it is equal to $O(n^2)$. Moreover, we do not need to introduce big-M to model some constraints, which decreases the relaxation quality. In addition, the number of required constraints is polynomial in our case, where it is equal to $O(n^3)$.

For sake of clarity, we recall here the formulation. This formulation involves the concept of precedence-flow decision variables. Precisely, the linear ordering variables are defined for each pair of jobs $i,\ j$ in J, where $X^m_{i,j}$ takes the value 1 if i precedes j on M_m, m in $\{1,2\}$ and 0 otherwise. Moreover, the completion time of job j on M_1 (resp. M_2) is represented by a continuous variable $C_{1,j}$ (resp. $C_{2,j}$). Finally, the makespan is described by a continuous variable Z. The problem variables are linked together view a set of constraints, which are presented as follows:

$$\textbf{min} \qquad Z \tag{4.1}$$

$$\text{s.t.} \qquad X^m_{i,j} + X^m_{j,i} = 1, \qquad \forall i,\ j \in J\ i \neq j; m \in \{1,2\} \tag{4.2}$$

$$X^m_{i,j} \geq X^m_{i,v} + X^m_{v,j} - 1, \qquad \forall i,j,v \in J; m \in \{1,2\} \tag{4.3}$$

$$C_{1,j} = \textstyle\sum_{i=1}^{n} p_{1,i}.X^1_{i,j} + p_{1,j}, \quad \forall j \in J \tag{4.4}$$

$$C_{2,j} \geq C_{1,j} + l_j + p_{2,j}, \qquad \forall j \in J \tag{4.5}$$

$$C_{2,j} = Z - \textstyle\sum_{i=1}^{n} p_{2,i}.X^2_{j,i}, \quad \forall j \in J \tag{4.6}$$

$$Z \geq 0,\ C_{m,j} \geq 0, \qquad \forall j \in J, m \in \{1,2\} \tag{4.7}$$

$$X^m_{i,j} \in \{0,1\}, \qquad \forall i,\ j \in J, m \in \{1,2\} \tag{4.8}$$

4.3.2 Valid inequalities

To improve the MIP formulation, we proposed a set of valid inequalities to increase its relaxation quality and to improve the convergence of the resolution process. The basic idea of using dominance rules, within the framework of the linear programming, is to be able to reduce the solution space (the variables domain) and to construct optimal solutions directly. These rules can be easily added to the model by injecting new constraints.

At first, we recall the two valid inequalities presented in Section 3.4.1. For the first valid inequality, we take into consideration the extra time that a job has to wait before being assigned to M_1 or to M_2. The second valid inequality consists in

injecting partial optimal solutions into our model. We start by modeling the original problem instance using an interval graph, and we compute maximal cliques on it. A set of sub-instances is identified after considering the obtained maximal cliques. These sub-instances are then optimally solved, and for each one of them, a set of constraints is injected into the formulation.

Furthermore, we provide in the following proposition a new valid inequality.

Proposition 4.1. *Let us consider an instance $I = (J, p_1, l, p_2)$ of $F2|l_j|C_{max}$ and two jobs $i,\ j \in J$ where $p_{1,i} \leq p_{1,j}$, $p_{1,i}+l_i \leq p_{1,j}+l_j$, $p_{2,i} \geq p_{2,j}$ and $p_{2,i}+l_i \geq p_{2,j}+l_j$. As a consequence of Lemma 4.2, i must precede j on the two machines. This constraint can be given by: $X^1_{i,j} = X^2_{i,j} = 1$.*

4.3.3 Branching scheme

During the exact resolution of our MIP formulation, we adopt three branching schemes, namely, a critical path-based branching scheme, a dichotomic-based branching scheme and a quaternary-based branching scheme.

Critical path-based branching scheme. This branching scheme is based on the approach of [Grabowski et al., 1986]. We consider the disjunctive graph early described in Section 4.2.1 referred to as $G = (V \cup D)$. Given a feasible schedule S, we define by $G(S)$ the obtained graph after adding the appropriate selection and by P the obtained critical path on $G(S)$. On the basis of P, we define at most two blocks of operations referred to as B_m, $0 \leq m \leq 2$. Each block is constituted of the maximum number of successive nodes (at least two nodes) that are assigned to the same machine.

The branching scheme is based on the following theorem.

Theorem 4.1. *([Brucker et al., 1994]) Let I be an instance of $F2|l_j|C_{max}$ and S a valid schedule of I. If another feasible schedule S' exists for I such that $C_{max}(S') < C_{max}(S)$, then at least one operation from one block B_m in $G(S)$ is scheduled before the first operation or after the last operation of B_m, $0 \leq m \leq 2$.*

Corollary 4.1. *Given two valid schedules S and S' such that $C_{max}(S') < C_{max}(S)$, at least one of the following two conditions holds:*

- *at least one operation of block B_m in $G(S)$, which is different from the first operation in B_m, has to precede all other operations of the same block in $G(S')$, $0 \leq m \leq 2$.*

- *at least one operation of block B_m in $G(S)$, which is different from the last operation in B_m, has to follow all other operations of the same block in $G(S')$,* $0 \leq m \leq 2$.

Let us consider a node N of the search tree of our branch-and-cut and a feasible schedule S with respect to N. If we consider the critical path P computed on $G(S)$, we determine the blocks B_m on P, $0 \leq m \leq 2$. Hereafter, we suppose that block B_m is dedicated to the set of operations executed on M_m, m in $\{1, 2\}$. For each block B_m, we introduce two sets of operations E_m^B and E_m^A where E_m^B (resp. E_m^A) contains all operations of B_m except the first scheduled one (resp. the last scheduled one).

The branching scheme is described as follows. For each operation $O_{m,j}$ of E_m^B (resp. E_m^A), a descendant node of N is generated where this operation is fixed before (resp. after) the block E_m^B (resp. E_m^A) by fixing the variables $X_{j,i}^m$ (resp. $X_{u,j}^m$) to 1, for all $O_{m,i} \in E_m^B \setminus \{O_{m,j}\}$ (resp. $O_{m,u} \in E_m^A \setminus \{O_{m,j}\}$).

Hereafter, we define by $(E_1, ..., E_{2m})$ a permutation of all sets E_m^B and E_m^A, $0 \leq m \leq 2$. For the adopted branching strategy, we define a hypothetical branching order. In which, a descendant node from N is not generated for $O \in E_k$ unless all operations of $E_1 \cup E_2 \cup ... \cup E_{k-1}$ are handled, k in $\{1, ..., 2m\}$. Therefore, additional variables can be fixed in the node generated for $O \in E_k$:

- for all ℓ in $\{1, ..., k-1\}$, if $E_\ell = E_m^B$ and the operation $O_{m,j}$ is the first scheduled operation of B_m, then we fix the set of variables $X_{j,i}^m$ to 1 for all $O_{m,i} \in E_m^B \setminus \{O_{m,j}\}$.
- for all ℓ in $\{1, ..., k-1\}$, if $E_\ell = E_m^A$ and the operation $O_{m,j}$ is the last scheduled operation of B_m, then we fix the set of variable $X_{i,j}^m$ to 1 for all $O_{m,i} \in E_m^A \setminus \{O_{m,j}\}$.

As a consequence of the above Theorem and the arguments given above, all possible solutions on node N are investigated in the descendant nodes. We also observe that the intersection of solution spaces of each pair of descendant nodes of N is empty.

In order to fix a large number of disjunctive arcs, the sets E_m^A and E_m^B are ordered in the permutation $(E_1, ..., E_{2m})$ according to the non-increasing order of their cardinality, $0 \leq m \leq 2$.

Dichotomic-based branching scheme. In this branching scheme, we branch on a unique decision variable at a time. Thus, two nodes are derived, for which the decision variable is fixed to 1 for the first node and to 0 for the second.

As a consequence of Observation 1.1, we enumerate job sequences on the first machine. To do so, we determine the precedence relationships between all operations on M_1 by branching on the set of decision variables $X^1_{i,j}$, (i,j) in J^2. Precisely, we branch on a unique variable of $X^1_{i,j}$ at a time, (i,j) in J^2.

The branching strategy is described as follows. First, we sort all pairs of jobs in a non-decreasing order of

$$\begin{cases} \min(l_j - l_i - \max(p_{1,i}, p_{2,i}), l_i - l_j - \max(p_{1,j}, p_{2,j})), & \text{if } l_j > l_i + \max(p_{1,i}, p_{2,i}) \text{ and} \\ & l_i > l_j + \max(p_{1,j}, p_{2,j}) \\ l_j - l_i - \max(p_{1,i}, p_{2,i}), & \text{if } l_j > l_i + \max(p_{1,i}, p_{2,i}) \\ l_i - l_j - \max(p_{1,j}, p_{2,j}), & \text{otherwise.} \end{cases}$$

Let $\varepsilon = ((\varepsilon^1_1, \varepsilon^2_1), ..., (\varepsilon^1_{n^2-n}, \varepsilon^2_{n^2-n}))$ be the obtained order. In our method, we define a hypothetical order where we do not branch on the variable $X^1_{\varepsilon^1_k, \varepsilon^2_k}$ until all variables $X^1_{\varepsilon^1_1, \varepsilon^2_1}, ..., X^1_{\varepsilon^1_{k-1}, \varepsilon^2_{k-1}}$ are fixed.

Quaternary-based branching scheme. In this branching scheme, we branch on two decision variables simultaneously. Thus, four nodes are derived at a time.

Here, we enumerate job sequences on the first and the second machines in the same time. To that purpose, we determine the precedence relationships between operations on the two machines by branching on the set of variables $X^m_{i,j}$, i, j in J and m in $\{1, 2\}$. At a time, for a pair of jobs i, j in J, four descendant nodes are derived as follows:

- The first node is obtained after fixing $X^1_{i,j}$ and $X^2_{i,j}$ to 1.
- The second node is obtained after fixing $X^1_{i,j}$ and $X^2_{i,j}$ to 0.
- The third node is obtained after fixing $X^1_{i,j}$ to 1 and $X^2_{i,j}$ to 0.
- The fourth node is obtained after fixing $X^1_{i,j}$ to 0 and $X^2_{i,j}$ to 1.

The Branching strategy is described as follows. We sort all pairs of jobs as it is given in the dichotomic-branching scheme. Let $\varepsilon = ((\varepsilon^1_1, \varepsilon^2_1), ..., (\varepsilon^1_{n^2-n}, \varepsilon^2_{n^2-n}))$ be the obtained order. We then define a hypothetical order where we do not branch on a pair of jobs $(\varepsilon^1_k, \varepsilon^2_k)$ until we handle the pairs $(\varepsilon^1_1, \varepsilon^2_1), ..., (\varepsilon^1_{k-1}, \varepsilon^2_{k-1})$.

4.3.4 Heuristic method

The quality of the upper bound plays a main role in improving the effectiveness and the efficiency of the proposed branch-and-cut algorithm. Here, we introduce a heuristic method that constructs step by step a solution and then applies a local search procedure on it in order to improve its quality.

Applied on a node N, the heuristic method first determines a job sequence σ of all jobs on M_1. To do so, we provide in the following the different steps:

- Compute a set Π of operations on the disjunctive graph $G = (V, C \cup D)$ obtained with respect to N. This set contains the operations on M_1 with no unscheduled predecessors.
- Let $O_{1,j}$ be an operation of Π with the minimal relaxation value of variable $C_{1,j}$, j in J.
- $\sigma = \sigma \oplus j$.
- Remove $O_{1,j}$ from the set Π and investigate its successors for possible addition to Π.

This process is iterated until we obtain a complete job sequence on M_1.

On the basis of σ, we schedule the jobs on M_2 according to the non-decreasing order of their arrival times with respect to the fixed precedence relationships on M_2. Then by using the obtained job sequence on M_2, σ is updated according to the arrival times of jobs on M_1 with respect to the fixed precedence relationships. This process is iterated until no modification is observed on the job sequence σ between two successive iterations or $iter_{max}$ iterations are performed.

4.3.5 Node pruning procedure

We apply on each node of the branch-and-cut algorithm a procedure that aims at pruning nodes at early stage of the search tree. Let N be a node of the search tree such that a fixed job sequence σ (resp. π) of all jobs on M_1 (resp. M_2) exists. On the basis of Observation 1.1, we can stop branching on N. Although that some decision variables are not fixed, we can already determine the optimal solution of Ω_σ (resp. Ω_π) in $O(n \log n)$-time. As a consequence, we inject the feasible solution before pruning the node.

The implementation of the node pruning procedure is described as follows. Applied on a node N, an easy way to implement the procedure consists in verifying if

one of the two sets of variables $X^1_{i,j}$ or $X^2_{i,j}$ is fixed, for all i and j in J; $i \neq j$. However, this needs $O(n^2)$-time. Interestingly, we implement this procedure in two steps that require $O(n \log n)$-time. First, we verify the existence of a total order on one of the two machines using the relaxation values of variables $C_{m,j}$, m in $\{1, 2\}$ and j in J. Let σ (resp. π) be the obtained job sequence on M_1 (resp. M_2) after considering the jobs according to the non-decreasing order (resp. non-increasing order) of the relaxation values of variables $C_{1,j}$ (resp. $C_{2,j}$), j in J. A total order exists on M_1 (resp. M_2) if and only if $C_{1,\sigma_{[k]}} = C_{1,\sigma_{[k-1]}} + p_{1,\sigma_{[k]}}$ (resp. $C_{2,\pi_{[k-1]}} = C_{2,\pi_{[k]}} + p_{2,\pi_{[k-1]}}$) for all k in $\{1, ..., n\}$. In case a total order exists on M_1 (resp. M_2), we ensure in the second step that the set of variables $X^1_{\sigma_{[k]},\sigma_{[k+1]}}$ (resp. $X^2_{\pi_{[k+1]},\pi_{[k]}}$) are fixed to 1 for all k in $\{1, ..., n\}$. If it is the case, then the node N is pruned.

4.4 Branch-and-bound algorithm

In this section, we present an exact method for $F2|l_j|C_{max}$ based on a branch-and-bound enumeration scheme. We describe the considered branching scheme in detail as well as the different components used to improve the efficiency and effectiveness of the proposed method.

4.4.1 Branching scheme

As a consequence of Observation 1.1, the branching scheme consists in enumerating job sequences on M_1. Starting from the root node, where no job is scheduled yet, a descendant node is derived for each job where it is processed at the first position on M_1. This process is iterated such that at a node N_{σ_1} with a level k, a partial job sequence σ_1 of k jobs is first fixed on M_1. In order to discard nodes from the search tree, we evaluate the node N_{σ_1} using the following features:

- A pre-processing procedure.
- A lower bound of $F2|l_j|C_{max}$.
- A local search-based heuristic.
- Dominance rules.

Note that we adopt the depth-first node selection strategy.

4.4.2 Lower bounds

We incorporate the most competitive lower bounds in the literature in our branch-and-bound after considering the results of Section 3.2.2. However, in order to use

them in an efficient way inside the search tree, we extend them after considering the obtained partial schedules.

Before proceeding further, we consider the following notation. Let $\sigma = \sigma_1 \oplus \sigma_2$ be a complete job sequence of all jobs on M_1 that is composed from a fixed sub-sequence σ_1 and an arbitrary sub-sequence of $J \setminus J_{\sigma_1}$ called σ_2. Moreover, β_j stands for the starting time of job j of J_{σ_1} on M_2. To compute β_j for all j in J_{σ_1}, we suppose that the jobs of J_{σ_1} are executed on M_2, like in Observation 1.1.

Let us consider the following proposition.

Proposition 4.2. *Given a node N_{σ_1} of the search tree, we define a new time delay $l_j^{\sigma_1}$ for each job j of J. This time delay is given by:*

$$l_j^{\sigma_1} = \begin{cases} \beta_j - t_{1,j}(S) - p_{1,j}, \text{ if } j \in J_{\sigma_1}; S \in \Omega_{\sigma_1} \\ l_j, \text{ otherwise.} \end{cases} \tag{4.9}$$

Proof. Let us consider a node N_{σ_1} and a schedule S of Ω_{σ_1}. Note that $\beta_j \geq t_{1,j}(S) + p_{1,j} + l_j$, j in J_{σ_1}. Moreover, if we schedule the jobs of $J \setminus J_{\sigma_1}$ after σ_1 on M_1, we observe that the arrival times of J_{σ_1} remain unchanged. However, their starting times can only increase or remain the same since the jobs of $J \setminus J_{\sigma_1}$ are processed on M_2 with respect to Observation 1.1. Consequently, it holds that $t_{2,j}(S) \geq \beta_j$, for all j in J_{σ_1}. Therefore, the time delay that is observed by job j of J_{σ_1} is at least $\beta_j - t_{1,j}(S) - p_{1,j}$. □

As a consequence of Observation 1.1 and Proposition 4.2, we describe in the following corollary and remark how the lower bounds of Section 3.2 are invoked in the internal nodes of the branch-and-bound search tree.

Corollary 4.2. *Given a node N_{σ_1} of the search tree, it follows that for each schedule S of Ω_{σ_1}:*

$$C_{max}(S) \geq \max(LB(I_{\sigma_1}^u), \sum_{j \in J_{\sigma_1}} p_{1,j} + LB(I_{\sigma_1}^r)). \tag{4.10}$$

where LB is a valid lower bound, $I_{\sigma_1}^u = (J, p_1, l^{\sigma_1}, p_2)$ and $I_{\sigma_1}^r = (J_{\sigma_2}, p_1, l, p_2)$ are two instances of $F2|l_j|C_{max}$.

Remark 4.1. *Unfortunately, the relaxation scheme presented in (4.10) does not allow us to fully exploit the benefits of the precedence relationships deduced from the partial schedule. Indeed, consideration of these precedence relationships drastically increases the computation burden of the lower bounds. Except for $LB_{res,1}$, an*

extension of this lower bound referred to as $LB_{res,3}$ can be computed by solving a one machine scheduling problem with release dates and precedence constraints.

4.4.3 Upper bound

In this section, we describe a heuristic method that is intended to be used as an upper bound on the makespan. This heuristic is based on a local search exploration of a job sequence. At first, a job sequence σ of all jobs is determined on M_1 using the constructive heuristics of [Dell'Amico, 1996]. On the basis of σ, we schedule the jobs on M_2 according to the non-decreasing order of their arrival times, like in Observation 1.1. Then, by considering the job sequence π obtained on M_2, we update σ according to the arrival times of jobs. We iterate this process until σ is unchanged between two successive iterations or a certain number of iterations $iter_{max}$ is performed.

Moreover, we apply a local search procedure on σ and π in which we use two neighborhood operators:

- *Shift operator:* evaluates the job sequence that is obtained by removing a job j from its current position and relocates it at another position in the sequence.
- *Swap operator:* evaluates the job sequence that is obtained by exchanging two jobs i and j in the sequence.

The local search procedure randomly selects a neighborhood to be applied to the permutation. If the obtained job sequence has a better makespan value, then the procedure is iterated over the new job sequence. Otherwise, we repeat the same processing until all neighborhoods are applied without an improvement. If we consider all possible movements of the neighborhood operators, the computational time will drastically increase. Thus, we limit the number of neighborhood operations that occur per type to a certain number k_{ls}. We present a detailed description of the procedure in Algorithm 4.1.

While handling internal nodes of the search tree, we adjust the heuristic method to make it less time-consuming. First, the job sequence σ is now obtained as follows. At a given node N_{σ_1}, we complete the fixed sub-sequence σ_1 by applying the constructive heuristics of [Dell'Amico, 1996] on the instance constituted of jobs $J \backslash J_{\sigma_1}$. Moreover, we do not use the local search procedure, and we change the stop condition of the heuristic. In fact, the heuristic is terminated when no improvement on the makespan is observed between two successive iterations.

Algorithm 4.1: Local search exploration.

Result: UB
Using the heuristics of [Dell'Amico, 1996], determine a job sequence σ of J;
With respect to σ, derive a job sequence π on M_2 where jobs are scheduled according to Observation 1.1. Let $S = (\sigma, \pi)$ be the corresponding schedule;
$UB \leftarrow C_{max}(S)$;
$iter \leftarrow 0$;
repeat
  $iter \leftarrow iter + 1$;
  $\sigma' \leftarrow \sigma$;
  Apply the two operators swap and shift on π;
  Starting from π, update the job sequence σ by executing the jobs on M_1 with respect to Observation 1.1;
  Apply the two operators swap and shift on σ;
  Starting from σ, update the job sequence π by executing the jobs on M_2 with respect to Observation 1.1;
  Update schedule S;
  if $UB > C_{max}(S)$ **then**
    $UB \leftarrow C_{max}(S)$;
until $\sigma = \sigma'$ *or* $iter \geq iter_{max}$;

4.4.4 Dominance rules

Hereafter, we introduce three dominance rules that make it possible to reduce the solution search space. The dominance rules are introduced in the following two propositions and lemma.

Proposition 4.3. *Consider an instance I of $F2|l_j|C_{max}$ and a pair of jobs (i, j) where $p_{1,j} + l_j \leq p_{1,i} + l_i$, $l_i \leq l_j + p_{2,j}$ and $p_{1,j} \leq p_{2,j}$. If a node where j is directly fixed after i on M_1 exists, then this node can be fathomed.*

Proof. This proposition is a direct consequence of Lemma 4.1. □

Proposition 4.4. *Consider a node N_{σ_1} of the search tree where the job sequence σ_1 is fixed first on M_1. If there are two jobs i in J_{σ_1} and j in $J \setminus J_{\sigma_1}$ where $p_{1,j} \leq p_{1,i}$, $p_{2,i} \leq p_{2,j}$, $p_{1,j} + l_j \leq p_{1,i} + l_i$ and $p_{2,j} + l_j \geq p_{2,i} + l_i$, then N_{σ_1} can be fathomed.*

Proof. This proposition is a direct consequence of Lemma 4.2.

□

Let us consider the following notation. We define $\Delta(S', S, t_1, t_2)$ as the difference between the idle time value observed on M_2 in the interval $[t_1, t_2]$ in S' and that in S.

Lemma 4.4. *Let S and $S^{'}$ be two partial schedules obtained at two nodes N_{σ_1} and $N_{\sigma_1^{'}}$, respectively, such that $J_{\sigma_1} = J_{\sigma_1^{'}}$. If it holds that:*

$$\Delta(S^{'}, S, t, \infty) \geq 0, \ \forall t \in [0, \infty[, \tag{4.11}$$

then we can discard the node N_{σ_1}. We also say that $\sigma_1^{'}$ dominates σ_1.

Proof. It should be noted that if (4.11) holds then $C_{max}(S^{'}) \leq C_{max}(S)$. Otherwise, it contradicts the assumption that $\Delta(S^{'}, S, C_{max}(S), \infty) \geq 0$. In this proof, given an arbitrary job sequence σ_2 of $J \setminus J_{\sigma_1}$, we show that $\sigma_1^{'} \oplus \sigma_2$ dominates $\sigma_1 \oplus \sigma_2$. To do so, we iteratively prove that for all k in $\{1, ..., |J_{\sigma_2}|\}$ $\sigma_1^{'} \oplus \sigma_{2[1]} \oplus ... \oplus \sigma_{2[k]}$ dominates $\sigma_1 \oplus \sigma_{2[1]} \oplus ... \oplus \sigma_{2[k]}$. For job $j = \sigma_{2[1]}$, let S_j (resp. $S_j^{'}$) be a partial schedule of $\Omega_{\sigma_1 \oplus j}$ (resp. $\Omega_{\sigma_1^{'} \oplus j}$). We denote the arrival time of job j on M_2 by $\gamma_j = t_{1,j}(S_j) + p_{1,j} + l_j = t_{1,j}(S_j^{'}) + p_{1,j} + l_j$, and τ (resp. $\tau^{'}$) represents the minimal time such that the total idle time observed between γ_j and τ (resp. $\tau^{'}$) on M_2 in S (resp. $S^{'}$) is equal to $p_{2,j}$. The job sequences on M_2 of S_j and $S_j^{'}$ are obtained as follows. We start by removing all jobs that are scheduled between γ_j and τ (resp. $\tau^{'}$) on M_2 in S (resp. $S^{'}$). Then, we continuously process j followed by the removed jobs from S (resp. $S^{'}$) such that they end their processing at time τ (resp. $\tau^{'}$) on M_2 in S_j (resp. $S_j^{'}$). Since j is processed after γ_j on M_2, we observe that the idle time value of S_j (resp. $S_j^{'}$) is equal to the idle time value of S (resp. $S^{'}$) minus $p_{2,j}$ time units. Therefore, S_j and $S_j^{'}$ verify that:

$$\Delta(S_j^{'}, S_j, t, \infty) = \Delta(S^{'}, S, t, \infty) \geq 0, \ \forall t \in [0, \gamma_j].$$

Moreover, we notice that the schedule is the same in S_j and in S (resp. in $S_j^{'}$ and in $S^{'}$) after the instant $\max(\tau, \tau^{'})$. Therefore, it holds that:

$$\Delta(S_j^{'}, S_j, t, \infty) \geq 0, \ \forall t \in [\max(\tau, \tau^{'}), \infty[.$$

Interestingly, if the second machine for one of the two schedules $S_j^{'}$ and S_j is not idle during an interval $[t_1, t_2]$, then $\Delta(S_j^{'}, S_j, t, \infty)$ consists in determining the difference between a constant and a non-increasing value for all t in $[t_1, t_2]$. Therefore, $\Delta(S_j^{'}, S_j, t, \infty)$ is monotonic in the interval $[t_1, t_2]$. For S_j (resp. $S_j^{'}$), it holds that M_2 is not idle in the interval $[\gamma_j, \tau]$ (resp. $[\gamma_j, \tau^{'}]$). Consequently, M_2 is not idle in the interval $[\gamma_j, \max(\tau, \tau^{'})]$ at least in one of the two schedules S_j and $S_j^{'}$. Therefore, $\Delta(S_j^{'}, S_j, t, \infty)$ is monotonic in the interval $[\gamma_j, \max(\tau, \tau^{'})]$. In addition, since $\Delta(S_j^{'}, S_j, \gamma_j, \infty) \geq 0$ and $\Delta(S_j^{'}, S_j, \max(\tau, \tau^{'}), \infty) \geq 0$, it holds that:

$$\Delta(S_j^{'}, S_j, t, \infty) \geq 0, \ \forall t \in [\gamma_j, \max(\tau, \tau^{'})].$$

This process is reiterated on job $\sigma_{2[2]}$ with $\sigma_1 = \sigma_1 \oplus j$ and $\sigma_1^{'} = \sigma_1^{'} \oplus j$, and so on. □

We can apply the last dominance rule at every node N_σ with level k of the search tree by searching for a dominant job sequence π with respect to the lemma conditions. Clearly, an exponential number of permutations of J_σ exists. However, in this work, we limit ourselves to a polynomial number of permutations obtained using a neighborhood function. Applied on each node of the search tree, the neighborhood function enumerates a set of permutations that are obtained as follows.

- The insertion of the job $\sigma_{[k]}$ in the ℓ-th position of σ, ℓ in $\{1, ..., |\sigma| - 1\}$. We obtain here $|\sigma| - 1$ partial job sequences.
- Switching the job $\sigma_{[k]}$ with every job $\sigma_{[\ell]}$, ℓ in $\{1, ..., |\sigma| - 1\}$. We obtain here another $|\sigma| - 1$ partial job sequences.

If one of the obtained job sequences dominates σ, then the node N_σ can be discarded.

As a consequence of the adopted branching scheme, we may visit different nodes with the same set of fixed jobs. It is therefore interesting to check the validity of the above dominance rule in this case. To do this, we use the no-good list technique of [Schiex and Verfaillie, 1993]. While exploring the search tree, we record a sub-sequence per set of jobs. This sub-sequence is updated each time a dominance relationship occurs. A sub-sequence is characterized by three parameters:

- the set of scheduled jobs,
- the fixed sub-sequence,
- the idle time periods on M_2.

If a new node N_σ is considered, we verify first the existence of a job sequence π with the same set of jobs as σ in the no-good list. We identify two cases:

- In case no job sequence is found, we apply the neighborhood function on σ. If one of the obtained job sequences $\sigma^{'}$ dominates σ, then $\sigma^{'}$ is stored in the no-good list and N_σ is discarded. Otherwise, σ is stored in the no-good list and N_σ is developed.
- In case π exists, we study the presence of a dominance relationship between σ and π. Four cases are possible:

- ◇ $\pi = \sigma$. In this case, the node N_σ is developed.
- ◇ σ is not dominated by π. In this case, we apply the neighborhood function on σ. If one of the obtained job sequences $\sigma^{'}$ dominates σ, then the node N_σ is discarded and the dominance property between $\sigma^{'}$ and π is checked in order to update the no-good list. Otherwise, the node N_σ is developed.
- ◇ π dominates σ. In this case, N_σ is pruned.
- ◇ σ dominates π. In this case, we apply the neighborhood function on σ. If one of the obtained job sequences $\sigma^{'}$ dominates σ, then $\sigma^{'}$ replaces π in the no-good list and the node N_σ is discarded. Otherwise, the no-good list is updated by replacing π by σ and N_σ is developed.

4.5 Computational results

In this section, we study the performance of the proposed approaches. First, we report the results of the lower bounds within our branch-and-bound mehod. We then show some parameter tunning for the upper bound. Moreover, we analyze the impact of the proposed components on the performance of the two exact methods. Finally, a comparison with the exact method of [Dell'Amico, 1996] is given.

The tests were carried out on a set of eight classes of instances including six classes that were proposed by [Dell'Amico, 1996]. Moreover, two hard classes were introduced by us. For each class, the processing times on M_1 and M_2 and the time delays are generated between the intervals $[1, ..., p_{1,max}]$, $[1, ..., p_{2,max}]$ and $[1, ..., l_{max}]$, respectively. The values of $p_{1,max}$, $p_{2,max}$ and l_{max} per class are given in Table 4.1.

Table 4.1 – Classes generation.

	[Dell'Amico, 1996]						New classes	
	A	B	C	D	E	F	1	2
$p_{1,max}$	100	100	100	200	100	200	20	100
$p_{2,max}$	100	100	100	200	200	100	20	100
l_{max}	100	200	500	100	100	100	$\frac{n}{2}10$	$\frac{n}{2}100$

The problem size n was chosen from the set $\{10, 30, 50, 100, 150, 200\}$ (resp. $\{10, 15, 20, 25, 30, 50\}$) for Classes A-F (resp. Classes 1-2). For each pair of class and n, 10 instances were randomly generated. All algorithms were coded in C++ and compiled under CentOS 6.6. Moreover, we implemented the no-good list using a hash table (code available at http://burtleburte.net/ bob/hash/doobs.html). The experiments were conducted on an Intel(R) Xeon(R) @ 2.60GHz processor.

4.5.1 Lower bounds performance

In Section 3.5.1, we provided a detailed image of the performance of the lower bounds LB_{split}, $LB_{res,2}$, $LB_{bas,2}$, $LB_{tra,1}$ and $LB_{tra,3}$ at the root node. However, to better appraise the worth of these lower bounds, it is more interesting to study their performance within an exact method. To that aim, we analyze the impact of the incorporation of the lower bounds $LB_{res,1}$, LB_{split}, $LB_{res,2}$, $LB_{tra,2}$, $LB_{tra,3}$ and $LB_{res,3}$ within our branch-and-bound method. We implemented six versions of our branch-and-bound method for which a unique lower bound is activated per version. We denote by B&B(LB) the branch-and-bound version of the lower bound LB. To bring a better image of the quality of these lower bounds, we use neither of the pre-processing procedure, the heuristic method and the dominance rules. A time limit of 60 seconds is set to all versions. In Table 4.2, we provide for each version the following measures:

- USI: the number of unsolved instances within the given time limit.
- Nodes: the average number of the explored nodes for the solved instances.
- Time: the average computational time in seconds for the solved instances.

From Table 4.2, we observe that:

- B&B($LB_{res,3}$) provides the best results with only 28 (resp. 79) unsolved instances in an average time of 0.62 (resp. 5.73) seconds on Classes A-F (resp. Classes 1-2).
- $LB_{res,3}$ is more efficient than $LB_{tra,3}$ within our branch-and-bound method. Precisely, B&B($LB_{res,3}$) solves 51 additional instances compared to B&B($LB_{tra,3}$). This fact may be due to the adapted branching scheme that plays a main role in improving the performance of $LB_{res,3}$.
- B&B($LB_{res,2}$) and B&B(LB_{split}) exhibit a poor performance. Both of them fail to solve 400 instances over 480 possible ones.

To conclude, we are confident that $LB_{res,3}$ provides the best performance from both effectiveness and efficiency viewpoints. Therefore, we assume in the implementation of the branch-and-bound method that $LB_{res,3}$ is invoked in each internal node of the search tree.

4.5.2 Parameter tuning for the upper bound

The aim of this section is to determine the couple of parameters $(iter_{max}, k_{ls})$ that offers a good trade-off between effectiveness and efficiency for the upper bound. Thus, we performed an experiment in which we test the performance of the upper bound for a variety of combinations between $iter_{max}$ and k_{ls}. The experiment was conducted on the earlier described classes where we varied both $iter_{max}$ and k_{ls} in the set of values $\{1, 5, 10, 15, 20, 30, 40\}$. In Figure 4.1, we present the evolution of the average percentage deviation with respect to the maximal obtained upper bound value and the average CPU time in terms of $iter_{max}$ and k_{ls}.

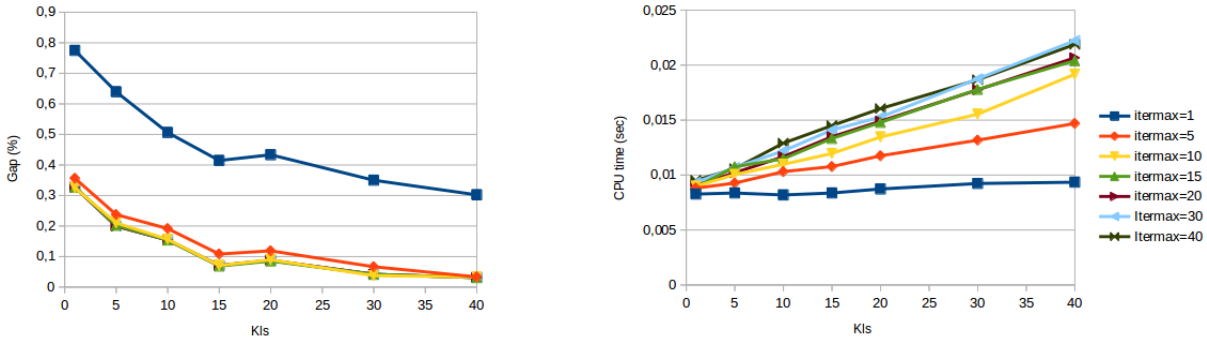


Figure 4.1 – Performance of the upper bound in terms of k_{ls} and $iter_{max}$.

On the basis of Figure 4.1, we observe that:

- The couple of values $(iter_{max} = 15, k_{ls} = 40)$ provides the best average gap with 0.03% while requiring a very short time that is equal to 0.02 seconds.
- For each value of $iter_{max}$, the average gap value decreases when the value of k_{ls} increases. This may be due to the intensive use of the neighborhood operators that improves the quality of the permutation. Not surprisingly, this leads to the increase of the CPU time.
- For $iter_{max} \geq 15$, the different versions of the upper bound exhibit the same performance in terms of average gap. However, the versions with $iter_{max} = 15$ are less time-consuming.

To conclude, we assume in our implementation that $iter_{max} = 15$ and $k_{ls} = 40$.

4.5.3 Branch-and-bound performance

The aim of this section is to better appraise the worth of the proposed components on the performance of our branch-and-bound method by analyzing the impact of their incorporation within this method. For that purpose, we implemented three versions of our branch-and-bound method:

- B&B$_{v1}$: Only $LB_{res,3}$ and the pre-processing procedure are applied.
- B&B$_{v2}$: The heuristic method, $LB_{res,3}$ and the pre-processing procedure are activated in each node.
- B&B$_{v3}$: The heuristic method, $LB_{res,3}$, the pre-processing procedure and the dominance rules are used.

In Table 4.3, we provide a comparison between the three versions of our branch-and-bound method on all classes. For each version, we set a time limit of 60 seconds and we provide the following values for each version: USI, Nodes and Time.

On the basis of Table 4.3, we make the following observations:

- Interestingly, B&B$_{v3}$ solves all the instances of Classes A-F in a very short time.
- B&B$_{v3}$ exhibits better results while requiring less time than B&B$_{v1}$ and B&B$_{v2}$ with only 64 unsolved instances compared to 104 and 71 unsolved instances, respectively.
- B&B$_{v2}$ outperforms B&B$_{v1}$. Indeed, the incorporation of the new heuristic method enhances the results where 33 additional instances are solved for Classes B, C, 1 and 2.
- An enhancement of performance is also observed when we incorporate the dominance rules. Precisely, B&B$_{v3}$ exhibits a good performance and outperforms B&B$_{v2}$. Actually, B&B$_{v3}$ solves seven new instances compared to B&B$_{v2}$ including five instances for Class 1 and two instances for Class 2.
- For both Classes 1 and 2, we observe that B&B$_{v3}$ exhibits the best results with only 64 unsolved instances compared to 75 and 71 unsolved instances for B&B$_{v1}$ and B&B$_{v2}$, respectively.
- Unfortunately, for $n > 30$ (resp. $n \geq 25$), the best version B&B$_{v3}$ fails to solve all instances of Class 1 (resp. Class 2). This may be due to the fact that the lower bounds perform poorly when the time delays are very large compared to the processing times.

Pushing our analysis a step further, we compare the performance of the best version B&B$_{v3}$ to the branch-and-bound algorithm of [Dell'Amico, 1996]. For equity reason, we set a time limit of 4 seconds for the best version of our branch-and-bound algorithm compared to 2000 seconds for B&B$_{Dell}$. Indeed, the experiments of [Dell'Amico, 1996] were conducted on a PC486 with a clock at 33 Mhz. We used

Super Pi, which is a single threaded program that calculates π using the Gauss-Legendre algorithm, as a crude estimate of CPU speed of the processor used by [Dell'Amico, 1996]. According to the scores available at [CPU,], Super Pi takes 41 minutes for a similar processor of Dell'Amico but with a clock at 100 Mhz. The same procedure needs 14.94 seconds on our machine. Therefore, a crude estimate is that our machine is about $3 \times 2460/14.94 \simeq 492$ times faster.

We report the results of this study in Table 4.4 and we observe the following points:

- Interestingly, B&B$_{v3}$ solves all the state of the art instances except one while requiring a very short computational time. Indeed, B&B$_{v3}$ outperforms B&B$_{Dell}$ after solving 359 literature instances out of 360 compared to four unsolved instances for B&B$_{Dell}$.

- The two methods easily manage to solve Classes A, B, D, E and F, on which they basically show a similar performance. However, on Class C, the instances become harder to solve since B&B$_{Dell}$ (resp. B&B$_{v3}$) fails to solve four instances (resp. one instance). Moreover, Nodes and Time increase ten times for B&B$_{v3}$ compared to the other classes.

4.5.4 MIP model performance

We study here the impact of the incorporation of the different components on the performance of our branch-and-cut method. First, we discuss the effect of the branching scheme on the overall performance of our method. Then, we bring a more detailed image of the impact of the proposed components.

We provide in Table 4.5 the results obtained by four versions of our branch-and-cut method that are obtained after the exact resolution of the MIP model (4.1)-(4.8). These version are presented as follows.

- MIP$_{critical}$: we adopt the critical path-based branching scheme.

- MIP$_{bin}$: we adopt the dichotomic-based branching scheme.

- MIP$_{bincplx}$: we adopt the standard branching scheme of CPLEX. CPLEX determines the variable on which we are branching.

- MIP$_{quat}$: we adopt the quaternary-based branching scheme.

We illustrate for each version the following measures: the number of unsolved instances (USI) within the time limit, the average number of explored nodes (Nodes) and the average CPU time in seconds (Time). Note that Nodes and CPU are only computed on the solved instances. A time limit of 2000 seconds is set to each version.

From Table 4.5, we made the following observations:

- $\text{MIP}_{critical}$ provides the best results with only 78 unsolved instances compared to 123, 133 and 125 for MIP_{bin}, $\text{MIP}_{bincplx}$ and MIP_{quat}, respectively.
- In term of CPU time and average number of explored nodes, $\text{MIP}_{critical}$ dominates the other versions on all classes. Except on Class 2, $\text{MIP}_{bincplx}$ solves the same number of instances as $\text{MIP}_{critical}$ while exploring less nodes in a shorter time. Note that for some classes the comparison seems to be irrelevant since $\text{MIP}_{critical}$ solves more instances.

As a consequence, we adopt the critical path-based branching scheme.

In Table 4.6, we report the results of an analysis where we study the impact of the valid inequalities, the pre-processing method, the heuristic method and the node pruning procedure on the performance of our MIP model. We implemented the following four versions of our branch-and-cut method:

- MIP_{v0}: The exact resolution of the MIP model (4.1)-(4.8).
- MIP_{v1}: The exact resolution of the MIP model (4.1)-(4.8) after adding the valid inequalities of Section 4.3.2.
- MIP_{v2}: The exact resolution of the MIP model (4.1)-(4.8) after adding the valid inequalities of Section 4.3.2 and the pre-processing method.
- MIP_{v3}: The exact resolution of the MIP model (4.1)-(4.8) after adding the valid inequalities of Section 4.3.2, the pre-processing method, the heuristic method, and the node-pruning procedure.

In Table 4.6, we see that:

- MIP_{v3} exhibits the best results in term of unsolved instances with only 45 compared to 78, 74 and 51 for MIP_{v0}, MIP_{v1} and MIP_{v2}, respectively.
- MIP_{v1} outperforms MIP_{v0}. The incorporation of the valid inequalities allows us to solve four additional instances. Precisely, one, four and one additional instances are solved for Class C, Class 1 and 2, respectively. However, two instances ($n \in \{50, 150\}$) of Class C become unsolved by MIP_{v1}.

- MIP_{v2} outperforms MIP_{v1}. Indeed, MIP_{v2} provides a better performance after solving all instances of the literature comparing to seven unsolved instances for MIP_{v1}. In addition, MIP_{v2} manages to solve eight additional instances per class for Classes 1-2.

- MIP_{v3} outperforms MIP_{v2}. An enhancement of performance is observed when we incorporate the heuristic method and the node pruning procedure. The two methods show basically the same performance on Classes A-F. However, on Class 1 and 2, MIP_{v3} solves four and two new instances respectively compared to MIP_{v2}.

4.5.5 MIP and B&B comparison

Finally, we exhibit a comparison between the two best versions of the proposed branch-and-bound method and the MIP model on the early discussed classes. A time limit of 2000 seconds is set to each method. We provide the results of this study in Table 4.7.

We notice that MIP_{v3} exhibits better results than $\text{B\&B}_{v3}$ with only 45 unsolved instances compared to 54 ones. On the Classes A-F, the two methods manage to solve all instances with an advantage to $\text{B\&B}_{v3}$ in term of average number of nodes. On Class 1, the two methods exhibit the same performance with 19 unsolved instances. However, we observe a difference of performance per size where for $n = 30$, $\text{B\&B}_{v3}$ solves two additional instances compared to MIP_{v3}. From the other hand, for $n = 50$, MIP_{v3} solves more two instances than $\text{B\&B}_{v3}$. Nevertheless, MIP_{v3} yields the best results on Class 2 after solving nine additional instances compared to $\text{B\&B}_{v3}$.

4.6 Conclusion

This chapter addressed the two-machine flow-shop problem with time delays. We introduced two exact methods for $F2|l_j|C_{max}$. The first method consists of a branch-and-cut method. This method includes the consideration of a critical path-based branching scheme, a pre-processing procedure, a heuristic method and a node pruning procedure. Moreover, we presented a branch-and-bound algorithm for $F2|l_j|C_{max}$. For which, we introduced a new heuristic method and proposed three dominance rules. A computational study, which was carried out on a set of 480 instances, shows that our approaches outperform the exact method of [Dell'Amico, 1996].

Table 4.2 – Lower bound performance comparison within a branch-and-bound method.

Classes A-F																		
	$B\&B(LB_{res,1})$			$B\&B(LB_{res,2})$			$B\&B(LB_{split})$			$B\&B(LB_{tra,2})$			$B\&B(LB_{tra,3})$			$B\&B(LB_{res,3})$		
Size	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
10	0	249435.34	1.49	0	1774908.12	7.55	0	2997781.75	10.98	0	133903.45	0.93	0	69410.40	0.77	0	637.33	0.01
30	14	0.00	0.00	60	-	-	60	-	-	13	0.00	0.00	12	0.00	0.00	7	3108.49	0.47
50	16	0.00	0.00	60	-	-	60	-	-	18	0.00	0.00	16	0.00	0.00	6	3139.02	1.48
100	14	0.00	0.01	60	-	-	60	-	-	15	0.00	0.01	13	0.00	0.00	4	364.14	0.84
150	11	0.00	0.02	60	-	-	60	-	-	10	0.00	0.01	10	0.00	0.02	8	213.17	0.80
200	8	0.00	0.03	60	-	-	60	-	-	9	0.00	0.03	8	0.00	0.03	3	17.96	0.20
AVG	63	50390.98	0.31	300	1774908.12	7.55	300	2997781.75	10.98	65	27234.60	0.20	59	13835.96	0.16	28	1219.87	0.62
Classes 1-2																		
10	0	410746.25	3.06	0	39186.60	0.59	0	492281.44	3.30	0	255899.75	2.36	0	193775.45	2.36	0	1968.45	0.04
15	20	-	-	20	-	-	20	-	-	19	0.00	0.00	19	0.00	0.00	6	278261.72	12.06
20	20	-	-	20	-	-	20	-	-	20	-	-	20	-	-	14	156988.83	10.60
25	20	-	-	20	-	-	20	-	-	20	-	-	20	-	-	19	19099.00	1.90
30	20	-	-	20	-	-	20	-	-	20	-	-	20	-	-	20	-	-
50	20	-	-	20	-	-	20	-	-	20	-	-	20	-	-	20	-	-
AVG	100	410746.25	3.06	100	39186.60	0.59	100	492281.44	3.30	99	243714.05	2.25	99	184548.05	2.24	79	119416.22	5.73

Table 4.3 – The branch-and-bound algorithm performance t=60s.

		B&B$_{v1}$			B&B$_{v2}$			B&B$_{v3}$		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
Class A	10	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	30	0	0.00	0.00	0	0.00	0.01	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.01	0	0.00	0.01
	100	0	0.00	0.01	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	0.02	0	0.00	0.04	0	0.00	0.04
	200	0	0.00	0.04	0	0.00	0.08	0	0.00	0.08
	AVG	0	0.00	0.01	0	0.00	0.03	0	0.00	0.02
Class B	10	0	65.90	0.01	0	56.00	0.01	0	13.40	0.00
	30	0	622.60	0.70	0	0.00	0.00	0	0.00	0.00
	50	0	34.80	0.03	0	0.00	0.01	0	0.00	0.01
	100	1	137.44	0.82	0	0.00	0.03	0	0.00	0.03
	150	2	0.00	0.02	0	0.00	0.05	0	0.00	0.05
	200	0	0.00	0.06	0	0.00	0.10	0	0.00	0.10
	AVG	3	148.60	0.27	0	9.33	0.04	0	2.23	0.03
Class C	10	0	98.80	0.01	0	50.00	0.01	0	33.40	0.01
	30	6	3039.75	6.05	0	158.20	0.22	0	17.00	0.02
	50	6	2266.50	14.74	0	85.80	0.30	0	24.00	0.08
	100	3	1400.29	9.41	0	0.30	0.04	0	0.00	0.03
	150	8	0.00	0.03	0	0.00	0.07	0	0.00	0.07
	200	3	113.57	6.00	0	9.40	0.84	0	8.90	0.85
	AVG	26	965.00	5.62	0	50.62	0.25	0	13.88	0.18
Class D	10	0	1.00	0.00	0	0.10	0.00	0	0.10	0.00
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.01	0	0.00	0.01
	100	0	29.60	0.16	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	0.01	0	0.00	0.03	0	0.00	0.03
	200	0	0.00	0.01	0	0.00	0.04	0	0.00	0.04
	AVG	0	5.10	0.03	0	0.02	0.02	0	0.02	0.02
Class E	10	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	9.80	0.01	0	0.00	0.01	0	0.00	0.01
	100	0	0.00	0.01	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	0.01	0	0.00	0.04	0	0.00	0.04
	200	0	0.00	0.02	0	0.00	0.05	0	0.00	0.06

Table 4.3 - Continued.

		B&B$_{v1}$			B&B$_{v2}$			B&B$_{v3}$		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
	AVG	0	1.63	0.01	0	0.00	0.02	0	0.00	0.02
Class F	10	0	2.30	0.00	0	0.30	0.00	0	0.30	0.00
	30	0	0.00	0.00	0	0.00	0.01	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.01	0	0.00	0.01
	100	0	0.00	0.00	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	0.01	0	0.00	0.04	0	0.00	0.04
	200	0	0.00	0.03	0	0.00	0.06	0	0.00	0.06
	AVG	0	0.38	0.01	0	0.05	0.02	0	0.05	0.02
Class 1	10	0	69.50	0.01	0	55.60	0.01	0	31.20	0.01
	15	0	793.50	0.30	0	606.20	0.23	0	144.90	0.07
	20	4	4955.67	3.08	4	3197.17	2.23	1	5558.56	5.01
	25	9	1096.00	1.38	7	2.00	0.00	6	2782.00	4.74
	30	10	-	-	9	11271.00	29.76	8	2277.50	4.97
	50	10	-	-	10	-	-	10	-	-
	AVG	33	1461.48	0.85	30	1235.93	1.52	25	1927.74	2.14
Class 2	10	0	98.80	0.01	0	50.00	0.01	0	33.40	0.01
	15	2	3253.75	1.27	1	14455.11	7.11	0	7197.80	4.54
	20	10	-	-	10	-	-	9	30577.00	28.98
	25	10	-	-	10	-	-	10	-	-
	30	10	-	-	10	-	-	10	-	-
	50	10	-	-	10	-	-	10	-	-
	AVG	42	1501.00	0.57	41	6873.47	3.37	39	4899.29	3.55

Table 4.4 – Comparison with the state of the art exact method.

		B&B$_{Dell}$			B&B$_{v3}$					
					time limit=4 seconds			time limit=60 seconds		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
Class A	10	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	1.00	0.05	0	0.00	0.01	0	0.00	0.01
	100	0	1.00	0.11	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	0.01	0	0.00	0.04	0	0.00	0.04
	200	0	0.00	0.02	0	0.00	0.08	0	0.00	0.08
	AVG	0	0.33	0.03	0	0.00	0.02	0	0.00	0.02
Class B	10	0	8.40	0.02	0	13.40	0.00	0	13.40	0.00
	30	0	1.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	1.00	0.02	0	0.00	0.01	0	0.00	0.01
	100	0	1.00	0.09	0	0.00	0.03	0	0.00	0.03
	150	0	1.00	0.67	0	0.00	0.05	0	0.00	0.05
	200	0	1.00	1.41	0	0.00	0.10	0	0.00	0.10
	AVG	0	2.23	0.37	0	2.23	0.03	0	2.23	0.03
Class C	10	0	10.20	0.01	0	33.40	0.01	0	33.40	0.01
	30	2	23.60	0.04	0	17.00	0.02	0	17.00	0.02
	50	1	1.00	0.01	0	24.00	0.08	0	24.00	0.08
	100	0	3.10	0.28	0	0.00	0.03	0	0.00	0.03
	150	0	1.00	0.57	0	0.00	0.07	0	0.00	0.07
	200	1	1.00	1.86	1	0.00	0.13	0	8.90	0.85
	AVG	4	6.65	0.46	1	12.61	0.06	0	13.88	0.18
Class D	10	0	0.00	0.00	0	0.10	0.00	0	0.10	0.00
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.01	0	0.00	0.01
	100	0	0.00	0.00	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	0.00	0	0.00	0.03	0	0.00	0.03
	200	0	1.00	1.32	0	0.00	0.04	0	0.00	0.04
	AVG	0	0.17	0.22	0	0.02	0.02	0	0.02	0.02
Class E	10	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	30	0	1.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.01	0	0.00	0.01
	100	0	0.00	0.00	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	0.00	0	0.00	0.04	0	0.00	0.04
	200	0	1.00	1.14	0	0.00	0.06	0	0.00	0.06
	AVG	0	0.33	0.19	0	0.00	0.02	0	0.00	0.02
Class F	10	0	0.00	0.00	0	0.30	0.00	0	0.30	0.00
	30	0	1.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.01	0	0.00	0.01
	100	0	0.00	0.00	0	0.00	0.02	0	0.00	0.02
	150	0	0.00	0.00	0	0.00	0.04	0	0.00	0.04
	200	0	0.00	0.06	0	0.00	0.06	0	0.00	0.06
	AVG	0	0.17	0.01	0	0.05	0.02	0	0.05	0.02

Table 4.5 – The branching scheme influence on the MIP model performance t=2000s.

		$MIP_{Critical}$			MIP_{bin}			$MIP_{bincplx}$			MIP_{quat}		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
Class A	10	0	2.30	0.00	0	8044.20	2.83	0	98.20	0.15	0	4204.10	1.34
	30	0	6007.40	4.57	1	0.00	0.00	1	0.00	0.00	1	0.00	0.00
	50	0	0.20	0.03	1	0.00	0.00	1	0.00	0.00	1	0.00	0.00
	100	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	150	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	200	0	216.50	22.96	1	0.00	0.00	1	0.00	0.00	1	0.00	0.00
	AVG	0	1037.73	4.59	3	1411.26	0.50	3	17.23	0.03	3	737.56	0.24
Class B	10	0	645.70	0.12	0	21768.40	7.11	0	279.30	0.45	0	34685.10	11.56
	30	0	26.40	0.05	1	1357.44	2.54	4	0.00	0.00	1	2239.22	3.23
	50	0	1636.80	3.87	4	0.00	0.00	4	0.00	0.00	4	0.00	0.00
	100	0	119.50	2.32	2	0.75	3.22	4	0.00	0.00	2	0.75	2.23
	150	0	127.50	7.17	1	0.00	0.00	1	0.00	0.00	1	0.00	0.00
	200	0	0.40	1.85	1	0.00	0.00	1	0.00	0.00	1	0.00	0.00
	AVG	0	426.05	2.57	9	4507.98	2.35	14	60.72	0.10	9	7196.27	3.18
Class C	10	0	488.00	0.11	0	11819.70	4.17	0	557.00	0.85	0	69500.80	22.97
	30	3	16.00	0.07	3	86463.00	188.39	8	0.00	0.00	3	1480.71	3.31
	50	3	14.57	0.25	6	1056.75	7.49	9	0.00	0.00	6	44.25	1.36
	100	0	43.90	1.70	2	1.12	3.39	5	0.00	0.00	2	1.12	3.37
	150	0	15.60	6.01	0	899.20	125.09	8	0.00	0.00	1	2.33	17.04
	200	0	1063.30	106.50	3	1.71	32.71	7	0.00	0.00	3	1.71	28.69

Table 4.5 - Continued.

		$MIP_{Critical}$			MIP_{bin}			$MIP_{bincplx}$			MIP_{quat}		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
	AVG	6	302.26	21.21	14	16014.74	62.99	37	242.17	0.37	15	15679.82	14.21
Class D	10	0	6.40	0.00	0	11718.50	3.11	0	97.40	0.15	0	3574.80	1.21
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	100	0	0.40	0.19	1	0.00	0.00	1	0.00	0.00	1	0.00	0.00
	150	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	200	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	AVG	0	1.13	0.03	1	1986.19	0.53	1	16.51	0.03	1	605.90	0.20
Class E	10	0	0.20	0.00	0	51074.90	17.82	0	167.40	0.23	0	6.20	0.00
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	3.50	0.07	2	0.00	0.00	2	0.00	0.00	2	0.00	0.00
	100	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	150	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	200	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	AVG	0	0.62	0.01	2	8806.02	3.07	2	28.86	0.04	2	1.07	0.00
Class F	10	0	39.20	0.01	0	54718.70	18.80	0	362.00	0.38	0	84962.10	33.69
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	100	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	150	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00

Table 4.5 - Continued.

		$\text{MIP}_{Critical}$			MIP_{bin}			$\text{MIP}_{bincplx}$			MIP_{quat}		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
	200	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	AVG	0	6.53	0.00	0	9119.78	3.13	0	60.33	0.06	0	14160.35	5.61
Class 1	10	0	691.30	0.14	0	18460.70	5.93	0	608.90	1.06	0	140281.80	50.46
	30	0	21834.70	6.40	6	269943.25	213.51	0	15003.00	118.03	7	6363.33	3.40
	50	6	1160784.00	482.25	9	481.00	0.64	9	25077.00	993.13	9	245.00	0.32
	100	9	3.00	0.05	9	5.00	0.86	10	-	-	9	5.00	0.59
	150	10	-	-	10	-	-	10	-	-	10	-	-
	200	10	-	-	10	-	-	10	-	-	10	-	-
	AVG	35	194735.95	79.78	44	79054.12	57.18	39	8628.38	104.00	45	94810.53	34.38
Class 2	10	0	488.00	0.11	0	11819.70	3.88	0	557.00	0.77	0	69500.80	22.55
	15	0	69238.50	21.75	10	-	-	0	4485.70	40.50	10	-	-
	20	7	2498106.25	1137.00	10	-	-	7	16586.33	809.07	10	-	-
	25	10	-	-	10	-	-	10	-	-	10	-	-
	30	10	-	-	10	-	-	10	-	-	10	-	-
	50	10	-	-	10	-	-	10	-	-	10	-	-
	AVG	37	356155.81	157.81	50	11819.70	3.88	37	4355.91	123.48	50	69500.80	22.55

Table 4.6 – The MIP model performance t=2000s.

		MIP_{v0}			MIP_{v1}			MIP_{v2}			MIP_{v3}		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
Class A	10	0	2.30	0.00	0	1.10	0.00	0	0.00	0.00	0	0.00	0.00
	30	0	6007.40	4.57	0	10.60	0.02	0	3.70	0.02	0	3.70	0.02
	50	0	0.20	0.03	0	0.20	0.04	0	0.00	0.00	0	0.00	0.00
	100	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	150	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	200	0	216.50	22.96	0	10.70	6.07	0	0.00	0.00	0	0.00	0.00
	AVG	0	1037.73	4.59	0	3.77	1.02	0	0.62	0.00	0	0.62	0.00
Class B	10	0	645.70	0.12	0	276.90	0.08	0	37.40	0.02	0	44.40	0.03
	30	0	26.40	0.05	0	1.30	0.05	0	0.00	0.01	0	0.00	0.01
	50	0	1636.80	3.87	0	3.90	0.20	0	0.00	0.02	0	0.00	0.02
	100	0	119.50	2.32	0	20.50	1.82	0	0.30	0.34	0	0.30	0.32
	150	0	127.50	7.17	0	51.10	7.16	0	0.00	0.00	0	0.00	0.00
	200	0	0.40	1.85	0	0.30	1.84	0	0.00	0.00	0	0.00	0.00
	AVG	0	426.05	2.57	0	59.00	1.86	0	6.28	0.06	0	7.45	0.06
Class C	10	0	488.00	0.11	0	326.40	0.12	0	38.70	0.03	0	39.50	0.04
	30	3	16.00	0.07	2	10642.38	19.14	0	111.40	0.29	0	115.10	0.31
	50	3	14.57	0.25	4	7.33	0.39	0	49.40	0.69	0	253.30	2.06
	100	0	43.90	1.70	0	5.60	1.34	0	1.10	0.78	0	1.10	0.79
	150	0	15.60	6.01	1	1.78	4.81	0	1.00	4.11	0	1.00	4.35
	200	0	1063.30	106.50	0	22.60	19.75	0	1.80	7.73	0	1.80	7.92

Table 4.6 - Continued.

		MIP_{v0}			MIP_{v1}			MIP_{v2}			MIP_{v3}		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
	AVG	6	302.26	21.21	7	1674.43	7.75	0	33.90	2.27	0	68.63	2.58
Class D	10	0	6.40	0.00	0	3.30	0.00	0	1.30	0.00	0	1.30	0.00
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	100	0	0.40	0.19	0	0.20	0.26	0	0.00	0.18	0	0.00	0.18
	150	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	200	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	AVG	0	1.13	0.03	0	0.58	0.04	0	0.22	0.03	0	0.22	0.03
Class E	10	0	0.20	0.00	0	0.20	0.00	0	0.00	0.00	0	0.00	0.00
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	3.50	0.07	0	1.80	0.06	0	0.00	0.02	0	0.00	0.02
	100	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	150	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	200	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	AVG	0	0.62	0.01	0	0.33	0.01	0	0.00	0.00	0	0.00	0.00
Class F	10	0	39.20	0.01	0	2.00	0.00	0	0.60	0.00	0	1.10	0.00
	30	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	100	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	150	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00

Table 4.6 - Continued.

		MIP_{v0}			MIP_{v1}			MIP_{v2}			MIP_{v3}		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time	USI	Nodes	Time
	200	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00	0	0.00	0.00
	AVG	0	6.53	0.00	0	0.33	0.00	0	0.10	0.00	0	0.18	0.00
Class 1	10	0	691.30	0.14	0	326.60	0.10	0	42.00	0.03	0	42.20	0.04
	15	0	21834.70	6.40	0	5822.60	3.15	0	180.30	0.15	0	188.10	0.16
	20	6	1160784.00	482.25	3	620112.44	546.23	1	13694.00	12.93	1	11883.33	10.63
	25	9	3.00	0.05	8	122820.00	153.13	4	333702.34	471.09	3	413291.84	493.65
	30	10	-	-	10	-	-	9	732074.00	1429.39	7	627044.69	1116.12
	50	10	-	-	10	-	-	9	3036.00	10.80	8	8716.00	39.68
	AVG	35	194735.95	79.78	31	160273.06	143.53	23	77372.78	118.51	19	119533.22	170.27
Class 2	10	0	488.00	0.11	0	326.40	0.11	0	38.70	0.03	0	39.50	0.03
	15	0	69238.50	21.75	0	32701.40	20.74	0	1902.40	1.33	0	1546.60	1.03
	20	7	2498106.25	1137.00	6	1490245.25	1315.35	0	225193.70	233.21	0	106274.70	93.04
	25	10	-	-	10	-	-	8	608800.50	860.58	6	1092271.25	1119.21
	30	10	-	-	10	-	-	10	-	-	10	-	-
	50	10	-	-	10	-	-	10	-	-	10	-	-
	AVG	37	356155.81	157.81	36	262135.80	227.91	28	109029.66	127.09	26	160226.27	159.35

Table 4.7 – Comparison between the B&B method and the MIP model.

		$B\&B_{v3}$			MIP_{v3}		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Time
Class A	10	0	0.00	0.00	0	0.00	0.00
	30	0	0.00	0.00	0	3.70	0.02
	50	0	0.00	0.00	0	0.00	0.00
	100	0	0.00	0.01	0	0.00	0.00
	150	0	0.00	0.02	0	0.00	0.00
	200	0	0.00	0.05	0	0.00	0.00
	AVG	0	0.00	0.01	0	0.62	0.00
Class B	10	0	13.40	0.00	0	44.40	0.03
	30	0	0.40	0.00	0	0.00	0.01
	50	0	0.00	0.00	0	0.00	0.02
	100	0	0.00	0.01	0	0.30	0.32
	150	0	0.00	0.03	0	0.00	0.00
	200	0	0.00	0.06	0	0.00	0.00
	AVG	0	2.30	0.02	0	7.45	0.06
Class C	10	0	33.40	0.00	0	39.50	0.04
	30	0	64.10	0.09	0	115.10	0.31
	50	0	33.50	0.11	0	253.30	2.06
	100	0	0.20	0.01	0	1.10	0.79
	150	0	0.00	0.04	0	1.00	4.35
	200	0	15.00	1.23	0	1.80	7.92
	AVG	0	24.37	0.25	0	68.63	2.58
Class D	10	0	0.10	0.00	0	1.30	0.00
	30	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.00
	100	0	0.10	0.01	0	0.00	0.18
	150	0	0.00	0.01	0	0.00	0.00
	200	0	0.00	0.02	0	0.00	0.00
	AVG	0	0.03	0.01	0	0.22	0.03
Class E	10	0	0.00	0.00	0	0.00	0.00
	30	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.02
	100	0	0.00	0.01	0	0.00	0.00
	150	0	0.00	0.02	0	0.00	0.00
	200	0	0.00	0.03	0	0.00	0.00

Table 4.7 - Continued.

		B&B$_{v3}$			MIP$_{v3}$		
Class	**Size**	USI	Nodes	Time	USI	Nodes	Timee
	AVG	0	0.00	0.01	0	0.00	0.00
Class F	10	0	0.30	0.00	0	1.10	0.00
	30	0	0.00	0.00	0	0.00	0.00
	50	0	0.00	0.00	0	0.00	0.00
	100	0	0.00	0.01	0	0.00	0.00
	150	0	0.00	0.02	0	0.00	0.00
	200	0	0.00	0.03	0	0.00	0.00
	AVG	0	0.05	0.01	0	0.18	0.00
Class 1	10	0	31.20	0.00	0	42.20	0.04
	15	0	134.60	0.06	0	188.10	0.16
	20	1	5662.44	4.89	1	11883.33	10.63
	25	3	192723.58	256.00	3	413291.84	493.65
	30	5	217680.00	567.01	7	627044.69	1116.12
	50	10	-	-	8	8716.00	39.68
	AVG	19	60733.78	113.95	19	119533.22	170.27
Class 2	10	0	33.40	0.01	0	39.50	0.03
	15	0	7796.00	4.57	0	1546.60	1.03
	20	5	431107.19	552.06	0	106274.70	93.04
	25	10	-	-	6	1092271.25	1119.21
	30	10	-	-	10	-	-
	50	10	-	-	10	-	-
	AVG	35	89353.20	112.24	26	160226.27	159.35

Conclusion and future works

In this dissertation, we introduced several methods and strategies to solve two variants of the flow-shop scheduling problem. The solution approaches was based on:

- Introducing an efficient non trivial Integer Linear Programming (ILP) formulation for $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$. We also presented a pre-processing procedure that aims at reducing the number of decision variables of the proposed formulation.

- Providing theoretical and experimental analyses of the lower bounds of the literature. We presented a new classification scheme of the different lower bounds and we enhanced a time delay-based one. Finally, we elucidated dominance relationships between them.

- Covering three integer programming-based lower bounds for $F2|l_j|C_{max}$. The first scheme was obtained after solving the LP relaxation of a linear ordering variable-based formulation with the consideration of clique-based inequalities. In the second and the third scheme, we relaxed the $F2|l_j|C_{max}$ to $F2|p_{1,j} = a, p_{2,j} = b, l_j|C_{max}$ and to a particular assignment problem, respectively. Then, we applied an ILP formulation on the two relaxed instances.

- Investigating two exact methods for $F2|l_j|C_{max}$. First, we introduced a linear programming-based exact method that includes the consideration of dominance rules, valid inequalities, a critical path-based branching scheme, a new heuristic method, and a node pruning procedure. Then, we proposed an exact algorithm based on a branch-and-bound enumeration scheme. To improve its performance, we introduced a heuristic method based on a local search technique and three dominance rules.

Moreover, we provided extensive experimentations and comparisons with the state of the art best methods. We retained the following observations:

- On the $F2|p_{i,j} = 1, l_j|C_{max}$ instances, the exact resolution of the ILP model solves 279 more instances while requiring less time than the branch-and-bound method of [Moukrim et al., 2014].
- On the $F2|l_j|C_{max}$ instances, the linear relaxation of the linear ordering variable-based model together with the consideration of clique-based inequalities exhibits good results. Precisely, it yields the best lower bound value on all instances compared to the literature lower bounds.
- Our branch-and-bound outperforms the exact method of [Dell'Amico, 1996] on the literature instances. Moreover, we introduce new hard classes, on which our branch-and-cut method yields the best results.

Future Work

On the basis of the observations made in this thesis, we hereafter outline some directions for further research developments.

The 2-machine flow-shop problem with time delays. Future research can be focused on improving the proposed approaches for the 2-machine flow-shop problem with time delays. An interesting perspective consists in investigating new dominance properties and developing lower bound methods for the case where the time delays are very large comparing to processing times.

An interesting issue that deserves investigation is the 2-machine flow-shop problem with maximal and minimal time delays. In addition to the time delays (minimal delays) studied in this thesis, maximal time delays are considered. The maximal time delays arise when the waiting time between the two operations of the same job must not exceed a certain duration. A possible start would be trying to adapt the proposed MIP formulation of Chapter 4 to model this problem. Heuristic approaches may also be investigated.

Furthermore, the case where $m > 2$ machines are available can be studied. New dominance properties, lower bounds and advanced heuristic are needed.

The integrated scheduling problem of container handling system in a maritime terminal. A promising avenue for future research consists in studying the integrated scheduling problem of container handling system in a maritime terminal. In such a problem, we dispose of a set of n containers to be loaded or unloaded from

or onto a ship. The loading (resp. unloading) activity is constituted of three steps. The first step consists in unloading containers from the ship (resp. the storage yard) using *Quay Cranes* (resp. *Yard Cranes*). The second step ensures the transfer of containers to the storage yard (resp. quayside) using *Yard Vehicles*. Finally, the last step is to store (resp. load) containers in the yard (resp. onto the ship) using *Yard Cranes* (resp. *Quay Cranes*). The objective is to minimize the time it takes to serve n containers in the maritime terminal.

Bibliography

[CPU,] Cpu speed of pc486. `http://hwbot.org/benchmark/superpi_-_1m/rankings?hardwareTypeId=processor_4138&cores=1#start=0#interval=20`.

[Ageev and Baburin, 2016] Ageev, A. A. and Baburin, A. E. (2016). Approximating two-machine flow shop problem with delays when processing times depend only on machines. In *DOOR*.

[Allahverdi and Al-Anzi, 2002] Allahverdi, A. and Al-Anzi, F. S. (2002). Using two-machine flowshop with maximum lateness objective to model multimedia data objects scheduling problem for WWW applications. *Computers & OR*, 29(8):971–994.

[An et al., 2016] An, Y.-J., Kim, Y.-D., and Choi, S.-W. (2016). Minimizing makespan in a two-machine flowshop with a limited waiting time constraint and sequence-dependent setup times. *Computers & Operations Research*, 71(Supplement C):127 – 136.

[Asil and Naralan, 2016] Asil, H. and Naralan, A. (2016). Impact of information technology on management in small and medium industries. *Journal of Telecommunications System & Management*, 5(3):1–3.

[Brucker et al., 1994] Brucker, P., Jurisch, B., and Sievers, B. (1994). A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49(1):107 – 127.

[Carlier and Pinson, 1990] Carlier, J. and Pinson, E. (1990). A practical use of jackson's preemptive schedule for solving the job shop problem. *Annals of Operations Research*, 26(1-4):269 – 287.

[Chu and Proth, 1996] Chu, C. and Proth, J. (1996). Single machine scheduling with chain structured precedence constraints and separation time windows. *IEEE Transactions on Robotics and Automation*, 12:835âĂŞ843.

[Codato and Fischetti, 2006] Codato, G. and Fischetti, M. (2006). Combinatorial benders' cuts for mixed-integer linear programming. *Operations Research*, 54(4):756–766.

[Conway et al., 1967] Conway, R., Maxwell, W., and Miller, L. (1967). *Theory of Scheduling*. Addison-Wesley Publishing Company.

[Dell'Amico, 1996] Dell'Amico, M. (1996). Shop problems with two machines and time lags. *Operations Research*, 44(5):777–787.

[Dell'Amico and Vaessens, 1996] Dell'Amico, M. and Vaessens, R. (1996). In *Flow and open shop scheduling on two machines with transportation times and machine-independant processing times is NP-hard*. Dipartimento di Economia politica, Università di Modena.

[Desaulniers et al., 2006] Desaulniers, G., Desrosiers, J., and Solomon, M. M. (2006). Column generation.

[Emmons and Vairaktarakis, 2013] Emmons, H. and Vairaktarakis, G. (2013). *Flow Shop Scheduling*, volume 182. Springer US.

[Fondrevelle et al., 2006] Fondrevelle, J., Oulamara, A., and Portmann, M. (2006). Permutation flowshop scheduling problems with maximal and minimal time lags. *Computers & Operations Research*, 33(6):1540 – 1556.

[Garey et al., 1976] Garey, M. R., Johnson, D. S., and Sethi, R. (1976). The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

[Grabowski et al., 1986] Grabowski, J., Nowicki, E., and ZdrzaÅĆka, S. (1986). A block approach for single-machine scheduling with release dates and due dates. *European Journal of Operational Research*, 26(2):278 – 285.

[Graham et al., 1979] Graham, R., Lawler, E., Lenstra, J., and Kan, A. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. In P.L. Hammer, E. J. and Korte, B., editors, *Discrete Optimization II*, volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. Elsevier.

[Hall, 1994] Hall, L. A. (1994). A polynomial approximation scheme for a constrained flow-shop scheduling problem. *Mathematics of Operations Research*, 19(1):68–85.

[Hodson et al., 1985] Hodson, A., Muhlemann, A. P., and Price, D. H. R. (1985). A microcomputer based solution to a practical scheduling problem. *The Journal of the Operational Research Society*, 36(10):903–914.

[Johnson, 1954] Johnson, S. M. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68.

[Karuno and Nagamochi, 2003] Karuno, Y. and Nagamochi, H. (2003). A better approximation for the two-machine flowshop scheduling problem with time lags. In *Algorithms and Computation, 14th International Symposium, ISAAC 2003, Kyoto, Japan, December 15-17, 2003, Proceedings*, pages 309–318.

[Keha et al., 2009] Keha, A., Khowala, K., and Fowler, J. (2009). Mixed integer programming formulations for single machine scheduling problems. *Computers & Industrial Engineering*, 56(1):357 – 367.

[Khachiyan, 1980] Khachiyan, L. (1980). Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20(1):53 – 72.

[Kim et al., 1996] Kim, Y.-D., Lim, H.-G., and Park, M.-W. (1996). Search heuristics for a flowshop scheduling problem in a printed circuit board assembly process. *European Journal of Operational Research*, 91(1):124 – 143.

[Kovalyov and Werner, 1997] Kovalyov, M. Y. and Werner, F. (1997). A polynomial approximation scheme for problem $f2/r_j/c_{max}$. *Oper. Res. Lett.*, 20(2):75–79.

[Lawler et al., 1993] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. (1993). Sequencing and scheduling: Algorithms and complexity. *Handbooks in operations research and management science*, 4:445–522.

[Lenstra et al., 1977] Lenstra, J., Kan, A. R., and Brucker, P. (1977). Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343 – 362. Studies in Integer Programming.

[Miller et al., 1960] Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329.

[Mitten, 1959] Mitten, L. (1959). Sequencing n jobs on two machines with arbitrary time lags. *Management Science*, 5(3):293–298.

[Mkadem et al., 2016a] Mkadem, M. A., Moukrim, A., and Serairi, M. (2016a). Lower bounds for the two-machine flow shop problem with time delays. In *Annual International Conference of the German Operations Research Society*, Hambourg, Germany.

[Mkadem et al., 2016b] Mkadem, M. A., Moukrim, A., and Serairi, M. (2016b). Le problème flow-shop à deux machines avec temps de transport. In *Roadef*, Compiègne, France.

[Mkadem et al., 2017a] Mkadem, M. A., Moukrim, A., and Serairi, M. (2017a). An exact method for solving the two-machine flow-shop problem with time delays. In *The 13th Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP 2017)* , Proceedings of the 13th Workshop on Models and Algorithms for Planning and Scheduling Problems, pages 175–176, Seeon Abbey, Germany.

[Mkadem et al., 2017b] Mkadem, M. A., Moukrim, A., and Serairi, M. (2017b). A branch-and-bound algorithm for the two-machine flow-shop problem with time delays. In *2017 International Conference on Control, Decision and Information Technologies (CoDIT)*.

[Mkadem et al., 2018] Mkadem, M. A., Moukrim, A., and Serairi, M. (2018). *Lower Bounds for the Two-Machine Flow Shop Problem with Time Delays*, pages 527–533. Springer International Publishing, Cham.

[Moukrim et al., 2014] Moukrim, A., Rebaine, D., and Serairi, M. (2014). A branch and bound algorithm for the two-machine flowshop problem with unit-time operations and time delays. *RAIRO - Operations Research*, 48(2):235–254.

[Msakni et al., 2016] Msakni, M., Khallouli, W., Al-Salem, M., and Ladhari, T. (2016). Minimizing the total completion time in a two-machine flowshop problem with time delays. *Engineering Optimization*, 48(7):1164–1181.

[Nawaz et al., 1983] Nawaz, M., Enscore, J., and Ham, I. (1983). A heuristic algorithm for the n-job, m-machine sequencing problem. *Management Science*, 16/B:630âĂŞ637.

[Pinedo, 2008] Pinedo, M. L. (2008). *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition.

[Potts, 1985] Potts, C. N. (1985). Analysis of heuristics for two-machine flow-shop sequencing subject to release dates. *Mathematics of Operations Research*, 10(4):576–584.

[Rayward-Smith and Rebaine, 2008] Rayward-Smith, V. and Rebaine, D. (2008). Analysis of heuristics for the uet two-machine flow shop problem with time delays. *Computers & Operations Research*, 35(10):3298 – 3310. Part Special Issue: Search-based Software Engineering.

[Roy and Sussmann,] Roy, D. and Sussmann, B. les problèmes d'ordonnancement avec contraintes disjonctives. *Note Ds no. 9 bis*.

[Schiex and Verfaillie, 1993] Schiex, T. and Verfaillie, G. (1993). Nogood recording for static and dynamic constraint satisfaction problems. In *Proceedings of 1993 IEEE Conference on Tools with Al (TAI-93)*, pages 48–55.

[Soukhal et al., 2005] Soukhal, A., Oulamara, A., and Martineau, P. (2005). Complexity of flow shop scheduling problems with transportation constraints. *European Journal of Operational Research*, 161(1):32 – 41. IEPM: Focus on Scheduling.

[Su, 2003] Su, L.-H. (2003). A hybrid two-stage flowshop with limited waiting time constraints. *Computers & Industrial Engineering*, 44(3):409 – 424.

[Turing, 1936] Turing, A. (1936). On computable numbers with an application to the âĂİ EntscheidungsproblemâĂİ. *Proceeding of the London Mathematical Society*.

[Yang and Chern, 1995] Yang, D.-L. and Chern, M.-S. (1995). A two-machine flowshop sequencing problem with limited waiting time constraints. *Computers & Industrial Engineering*, 28(1):63 – 70.

[Ye et al., 2017] Ye, S., Zhao, N., Li, K., and Lei, C. (2017). Efficient heuristic for solving non-permutation flow-shop scheduling problems with maximal and

minimal time lags. *Computers & Industrial Engineering*, 113(Supplement C):160 – 184.

[Yu, 1996] Yu, W. (1996). *The two-machine flow shop problem and the one-machine total tardiness problem.* PhD thesis, Eindhoven University of Technology, The Netherlands.

[Yu et al., 2004] Yu, W., Hoogeveen, H., and Lenstra, J. (2004). Minimizing makespan in a two-machine flow shop with delays and unit-time operations is np-hard. *Journal of Scheduling*, 7(5):333 – 348.

[Zhang and van de Velde, 2010] Zhang, X. and van de Velde, S. (2010). Polynomial-time approximation schemes for scheduling problems with time lags. *Journal of Scheduling*, 13(5):553–559.