Decomposition methods for the preemptive Flexible Job-Shop Scheduling Problem

Carla Juvin, Laurent Houssin, Pierre Lopez

Decomposition methods for the preemptive Flexible Job-Shop Scheduling Problem*

Carla Juvin[1] Laurent Houssin[1,2] Pierre Lopez[1]

[1] LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France
[2] ISAE-SUPAERO, Toulouse, France
e-mails: {carla.juvin, laurent.houssin, pierre.lopez}@laas.fr

Abstract

In this paper, we focus on exact methods to solve the preemptive Flexible Job-Shop Scheduling Problem with makespan and total completion time minimisation objective functions. Mathematical and constraint programming models enable the resolution of this problem for small instances. However, as an NP-hard problem, the cost of solving grows rapidly when considering larger instances. In this regard, we propose a logic-based Benders decomposition that relies on an efficient branch-and-bound procedure to solve the subproblem representing a pure (non-flexible) preemptive job-shop scheduling problem. Computational experiments are carried out and show the very good performance of our proposals.

Keywords: Flexible job-shop scheduling, preemption, Benders decomposition, mixed-integer programming, constraint programming.

1 Introduction

The job-shop scheduling problem (JSSP) is a well studied and NP-hard problem [2]. In the JSSP, a set of jobs are to be processed on a set of machines. Each job is composed of a sequence of operations that must be processed on machines with given processing time in a given job-dependent order, and each machine can process only one operation at a time. An extensive literature has been devoted to the JSSP, including the consideration of various additional constraints and different objective

*An updated version focusing on the makespan minimisation criterion was published in the journal Computers & Operations Research, see [1].

functions, even if makespan minimisation is mostly studied [3]. Nevertheless, very few works consider the job-shop problem with preemption. Le Pape and Baptiste [4] were the first to solve the pJSSP. Then, genetic algorithm and fuzzy logic are used in Yun [5]. To our knowledge, Ebadi and Moslehi are among the latest authors to have solved the problem to optimality. In particular, they proposed a MILP model [6] and an efficient branch-and-bound procedure [7].

The flexible job-shop problem (FJSSP) is a generalisation of the JSSP in the sense that, for each operation, there exists a set of eligible machines. This makes the problem more difficult to solve, as it consists of both a machine allocation and a sequencing problem. In the FJSSP literature, the most often studied optimisation criterion is the makespan minimisation; see for example the survey of Chaudhry and Khan [8]. Due to the complexity of the problem, many heuristics and metaheuristics have been developed, e.g., Tabu search [9, 10], genetic algorithms [11, 12] or simulated annealing [13, 14]. With the improvement of the performance of solvers, exact methods have also been designed, including a majority of mathematical models [15, 16]. Meng et al. [17] compared four mixed-integer linear programming models and one constraint programming model. They showed that the constraint programming model outperforms all other studied exact methods. Numerous variants of the FJSSP exist and have been also studied in the literature, e.g., including sequence-dependent setup times [9] or workers [18].

Studies on the preemptive flexible job-shop scheduling problem (pFJSSP) are scarce: three metaheuristics are presented in Zhang and Yang [19], while Jansen et al. [20] proposed several approximation algorithms for the FJSSP including one for the particular case of the pFJSSP. To the best of our knowledge, no exact method is reported in the literature to solve this problem. The purpose of the present article is to develop and compare several exact methods for the pFJSSP with makespan minimisation, namely: Mixed-Integer Linear Programming (MILP), Constraint programming (CP), and Logic-Based Benders decomposition (LBBD). We also extend these three methods to a second optimisation criterion: the total completion time minimisation.

LBBD was introduced by Hooker et al. [21] in the context of logic circuit verification and Hooker [22] formalised the concept. This approach has been applied to a large variety of scheduling problems, especially those including assignment decisions to make: Hooker and Ottosson [23] tackle a parallel-machine problem with release and delivery dates with the objective to minimise processing costs, Hooker [24] addresses the cumulative scheduling problem, while Tan and Terekhov [25] solve the flexible flow-shop problem with an LBBD algorithm. Naderi and Roshanaei [26] also proposed an LBBD scheme to solve the FJSSP without preemption.

The remainder of the paper is organised as follows. In the next section, the preemptive flexible job-shop scheduling problem is formally defined; MILP and CP models for the pFJSSP with makespan minimisation are also presented. In Section 3, we propose an LBBD scheme based on the use of an efficient branch-and-bound procedure for solving a pure scheduling subproblem. Computational experiments are carried out in Section 4 for two objective functions: the minimisation of the makespan and the minimisation of the total completion time. We draw a conclusion in Section 5.

2 Problem definition and modelling

An instance of the pFJSSP is defined as a set of jobs $\mathcal{J}$ and a set of machines $\mathcal{M}$. A job is not subject to an initial release date. Each job $i \in \mathcal{J}$ consists of a sequence of n_i operations. The j^{th} operation $O_{i,j} \in \mathcal{O}_i$ of a job i must be performed by one of the machine from the set of eligible machines $\mathcal{M}_{i,j} \subseteq \mathcal{M}$. Let $p_{i,j,m}$ denote the processing time of operation $O_{i,j}$ that is processed on machine $m \in \mathcal{M}_{i,j}$. Each machine can process at most one operation at a time and preemption is allowed: the processing of operations can be interrupted and resumed later without penalty. Although an operation can be interrupted (*preemption*), it is assumed that it must be fully processed by one and the same machine (no *migration*).

Notations and definitions

$\mathcal{J}$	Set of jobs
$\mathcal{M}$	Set of machines
n_i	Number of operations in job i ($i \in \mathcal{J}$)
$\mathcal{O}_i$	Set of operations in job i ($i \in \mathcal{J}$)
$O_{i,j}$	j^{th} operation of job i ($i \in \mathcal{J}, j \in \{1, \ldots, n_i\}$)
$\mathcal{M}_{i,j}$	Set of eligible machines for operation $O_{i,j}$ ($i \in \mathcal{J}, O_{i,j} \in \mathcal{O}_i$)
$\mathcal{I}_m$	Set of operations that can be processed by machine m ($m \in \mathcal{M}$)
$p_{i,j,m}$	Processing time of operation $O_{i,j}$ on machine m ($i \in \mathcal{J}, O_{i,j} \in \mathcal{O}_i, m \in \mathcal{M}_{i,j}$)
$p_{i,j}^{min}$	Shortest processing time of operation $O_{i,j}$, $p_{i,j}^{min} = \min_{m \in \mathcal{M}_{i,j}} p_{i,j,m}$ ($i \in \mathcal{J}, O_{i,j} \in \mathcal{O}_i$)
$r_{i,j}^{min}$	Shortest release date of operation $O_{i,j}$, $r_{i,j}^{min} = \sum_{j'<j} p_{i,j'}^{min}$ ($i \in \mathcal{J}, O_{i,j} \in \mathcal{O}_i$)
$q_{i,j}^{min}$	Shortest delivery time of operation $O_{i,j}$, $q_{i,j}^{min} = \sum_{j'>j} p_{i,j'}^{min}$ ($i \in \mathcal{J}, O_{i,j} \in \mathcal{O}_i$)

Example 1. *Consider a pFJSSP with 4 jobs and 4 machines. The processing times* $p_{i,j,m}$ *of operations* $\{O_{i,j} \in \mathcal{O}_i, j = 1, \ldots, n_i\}_{i \in \mathcal{J}}$*, on each eligible machine* $m \in \mathcal{M}_{i,j}$ *are given in Table 1.*

		$M1$	$M2$	$M3$	$M4$
$J1$	$O_{1,1}$	**2**	3		
	$O_{1,2}$		4	**4**	5
	$O_{1,3}$	4		3	**2**
$J2$	$O_{2,1}$	**3**	4		
	$O_{2,2}$		**1**		2
	$O_{2,3}$	**3**		2	
$J3$	$O_{3,1}$	3		5	**3**
	$O_{3,2}$		**4**	6	
	$O_{3,3}$			3	**2**
$J4$	$O_{4,1}$		**2**	3	
	$O_{4,2}$	2			**2**
	$O_{4,3}$			**4**	4

Table 1: Numerical example of an instance of the pFJSSP: processing times

A solution of this instance of the pFJSSP is to consider the following assignments: operations $O_{1,1}$, $O_{2,1}$ and $O_{2,3}$ on machine $M1$, operations $O_{2,2}$, $O_{3,2}$, $O_{4,1}$ on machine $M2$, operations $O_{1,2}$, $O_{4,3}$ on machine $M3$, and operations $O_{1,3}$, $O_{3,1}$, $O_{3,3}$, $O_{4,2}$ on machine $M4$. These assignments are associated with the processing times noted in bold in Table 1. Figure 1 depicts such a solution. On machine $M2$, operation $O_{3,2}$ is interrupted at $t = 5$ by operation $O_{2,2}$ and resumed on the same machine at $t = 6$. The makespan is 10.

2.1 Mathematical model

Let us introduce a mixed-integer program to solve the pFJSSP. It is based on a time-indexed formulation proposed by Bowman [27] to solve the preemptive job-shop problem. We adapt it to integrate the notion of resource flexibility.

Let $\mathcal{H} = \{1, 2, 3, \ldots, h\}$ be the time horizon. The decision variables are defined as follows:

$$x_{i,j,m} = \begin{cases} 1 & \text{if operation } O_{i,j} \text{ is processed on machine } m; \\ 0 & \text{otherwise.} \end{cases}$$

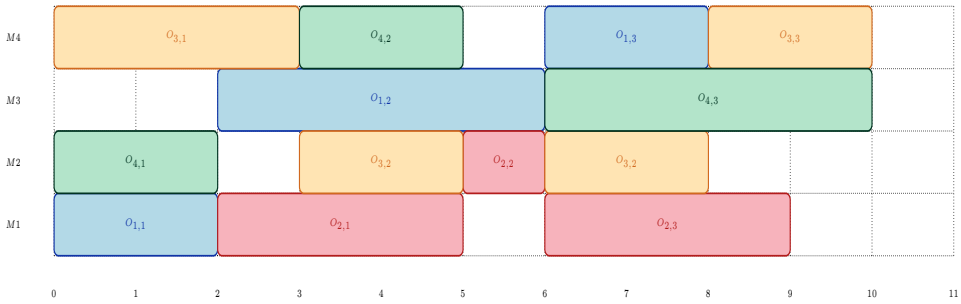

Figure 1: Gantt chart of a solution for Example 1 (4-job×4-machine pFJSSP, makespan 10)

$$y_{i,j,t} = \begin{cases} 1 & \text{if operation } O_{i,j} \text{ is in process at time } t; \\ 0 & \text{otherwise.} \end{cases}$$

Hence for the mathematical model, it yields:

$$\min C_{\max} \tag{1}$$

$$s.t. \quad \sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} = 1 \quad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i \tag{2}$$

$$\sum_{t=1}^{h} y_{i,j,t} \geq \sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} \times p_{i,j,m} \quad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i \tag{3}$$

$$\sum_{t'=t}^{h} y_{i,j,t'} \leq \max_{m \in \mathcal{M}_{i,j}} p_{i,j,m} \times (1 - y_{i,j+1,t}) \quad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i \setminus \{O_{i,n_i}\},\, t \in \mathcal{H} \tag{4}$$

$$\sum_{i \in \mathcal{J}} \sum_{j=1}^{n_i} x_{i,j,m} \times y_{i,j,t} \leq 1 \quad \forall m \in \mathcal{M},\, t \in \mathcal{H} \tag{5}$$

$$C_{\max} \geq (t+1) \times \sum_{i \in \mathcal{J}} y_{i,n_i,t} \quad \forall t \in \mathcal{H} \tag{6}$$

$$x_{i,j,m} \in \{0,1\} \quad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i, m \in \mathcal{M}_{i,j} \tag{7}$$

$$y_{i,j,t} \in \{0,1\} \quad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i,\, t \in \mathcal{H} \tag{8}$$

The objective (1) is the makespan minimisation. Constraints (2) ensure that each operation is processed by exactly one of the eligible machines. The duration of each operation is satisfied thanks to constraints (3). Constraints (4) express the precedence constraints between operations of the same job. Let $i \in \mathcal{J}$ be a job, $O_{i,j}$

and $O_{i,j+1}$ be two successive operations and t be a given period. If operation $O_{i,j+1}$ is processed at time t (i.e., $y_{i,j+1,t} = 1$), then operation $O_{i,j}$ must be completed before t (i.e., no longer be processed from t, $\forall t' \geq t$, $y_{i,j,t'} = 0$). Constraints (5) represent the disjunctive constraints on each machine. Finally, constraints (6) allow us to calculate the makespan.

Note that constraints (5) are nonlinear, but can be easily linearised since variables $x_{i,j,m}$ and $y_{i,j,t}$ are binary, the previous mathematical model thus becoming a MILP model by adding the variables:

$$z_{i,j,m,t} = \begin{cases} 1 & \text{if operation } O_{i,j} \text{ is processed on machine } m \text{ at time } t; \\ 0 & \text{otherwise} \end{cases}$$

and replacing constraints (5) by the following constraint set:

$$z_{i,j,m,t} \geq x_{i,j,m} + y_{i,j,t} - 1 \qquad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i,\, m \in \mathcal{M}_{i,j},\, t \in \mathcal{H} \tag{9}$$

$$z_{i,j,m,t} \leq x_{i,j,m} \qquad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i,\, m \in \mathcal{M}_{i,j},\, t \in \mathcal{H} \tag{10}$$

$$z_{i,j,m,t} \leq y_{i,j,t} \qquad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i,\, m \in \mathcal{M}_{i,j},\, t \in \mathcal{H} \tag{11}$$

$$\sum_{i \in \mathcal{J}} \sum_{j=1}^{n_i} z_{i,j,m,t} \leq 1 \qquad \forall m \in \mathcal{M},\, t \in \mathcal{H} \tag{12}$$

2.2 Constraint programming

To introduce the CP model, we use the IBM CP Optimizer solver, which allows the use of specific decision variables and constraints. In particular, we use interval variables, whose main feature is to be optional. This concept is relevant to our problem since each operation is performed on a machine within a set of eligible machines.

There are two possible ways to model preemptive operations of duration p with interval variables [28]:

1) by a set of optional interval variables with unfixed duration and imposing that the sum of their durations is equal to p;

2) by a chain of p unit-duration intervals.

These two possibilities are respectively illustrated in Figure 2 for an operation of duration 3.

In the context of preemptive multi-skill resource-constrained project scheduling, Polo-Mejía et al. [28] show that the second approach is more efficient. We have

Figure 2: Two possible models with interval variables for preemptive operations of duration 3

also conducted experiments leading to this conclusion for our specific pFJSSP. For this purpose, each preemptive operation is divided into unit-duration parts in our constraint programming model. We introduce the following decision variables:

- $task_{i,j}$: interval variable between the start and the end of the processing of operation $O_{i,j}$;
- $mode_{i,j,m}$: interval variable between the start and the end of the processing of operation $O_{i,j}$ on machine m (since the operations have multiple eligible machines and must be executed by exactly one of them, this is an optional variable);
- $part_{i,j,k,m}$: interval variable of unit duration of the processing of the k^{th} part of operation $O_{i,j}$ on machine m (optional variable).

$$\min C_{\max} \tag{13}$$

$$s.t. \quad C_{\max} \geq task_{i,n_i}.end \qquad \forall i \in \mathcal{J} \tag{14}$$

$$EndBeforeStart(task_{i,j}, task_{i,j+1}) \qquad \forall i \in \mathcal{J}, \; O_{i,j} \in \mathcal{O}_i \setminus \{O_{i,n_i}\} \tag{15}$$

$$EndBeforeStart(part_{i,j,k,m}, part_{i,j,k+1,m}) \qquad \forall i \in \mathcal{J}, \; O_{i,j} \in \mathcal{O}_i, \; m \in \mathcal{M}_{i,j}, 1 \leq k \leq p_{i,j,m} - 1 \tag{16}$$

$$Span(mode_{i,j,m}, part_{i,j,k,m} : \forall 1 \leq k \leq p_{i,j,m}) \qquad \forall i \in \mathcal{J}, \; O_{i,j} \in \mathcal{O}_i, \; m \in \mathcal{M}_{i,j} \tag{17}$$

$$Alternative(task_{i,j}, mode_{i,j,m} : \forall m \in \mathcal{M}_{i,j}) \qquad \forall i \in \mathcal{J}, \; O_{i,j} \in \mathcal{O}_i \tag{18}$$

$$PresenceOf(mode_{i,j,m}) \Rightarrow PresenceOf(part_{i,j,k,m}) \qquad \forall i \in \mathcal{J}, \; O_{i,j} \in \mathcal{O}_i, \; m \in \mathcal{M}_{i,j}, \; 1 \leq k \leq p_{i,j,m} \tag{19}$$

$$NoOverlap(part_{i,j,k,m} : \forall O_{i,j} \in \mathcal{I}_m, \; 1 \leq k \leq p_{i,j,m}) \qquad \forall m \in \mathcal{M} \tag{20}$$

The objective function (13) is to minimise the makespan. Constraints (14) define the makespan. The global constraint *EndBeforeStart* is used to model the precedence constraints, as in the two following constraint sets. Constraints (15) ensure that the operations of the same job will be processed with respect to the job sequence. Constraints (16) aim at ordering the parts of the operation and so avoid symmetries, and ensure that each part is treated one after the other. With the *Span* global constraint, constraints (17) are used to ensure that operation interval spans over all its processing parts (i.e., each operation starts with its first part and ends with its last part). Constraints (18) use the *Alternative* global constraint that ensures each operation to be processed by exactly one of the eligible machines. From the presence status (*PresenceOf* function) of mode interval variable, constraints (19) define the duration of each operation. With the *NoOverlap* global constraint, constraints (20) forbid the overlapping of operations processed on the same machine.

3 Logic-based Benders decomposition

Hybrid methods for optimisation, especially those combining MILP and CP, have been widely used since the late 90's. LBBD falls into this category. The LBBD approach decomposes a given problem into a master problem, usually a MILP, and one or several subproblems. It iterates between the master problem and the subproblems until an optimal solution is found by means of constraint generation techniques. At each iteration h, a subset of variables, say x, are fixed by the master problem, the resolution of the resulting subproblem provides the objective value z^h implied by the current solution $\bar{x}^h$. This bound is then used to generate cuts that are added to the master problem, and so on. A state-of-the-art about LBBD can be found in [29].

3.1 Decomposition scheme

The pFJSSP can be decomposed into an assignment master problem and a scheduling subproblem. In the master problem, each operation is assigned to exactly one of its eligible machines. Solving the master problem enables to find a lower bound LB for the global problem. Once this assignment $\bar{x}^h$ is fixed, the subproblem consists in a preemptive JSSP (i.e., without flexibility), of which a solution is an upper bound UB of the global problem and so a feasible solution of our problem. Solving this subproblem also allows the deduction of optimality cuts. Then, these cuts are added to the master problem, which is solved again. This general processing is illustrated in Figure 3.

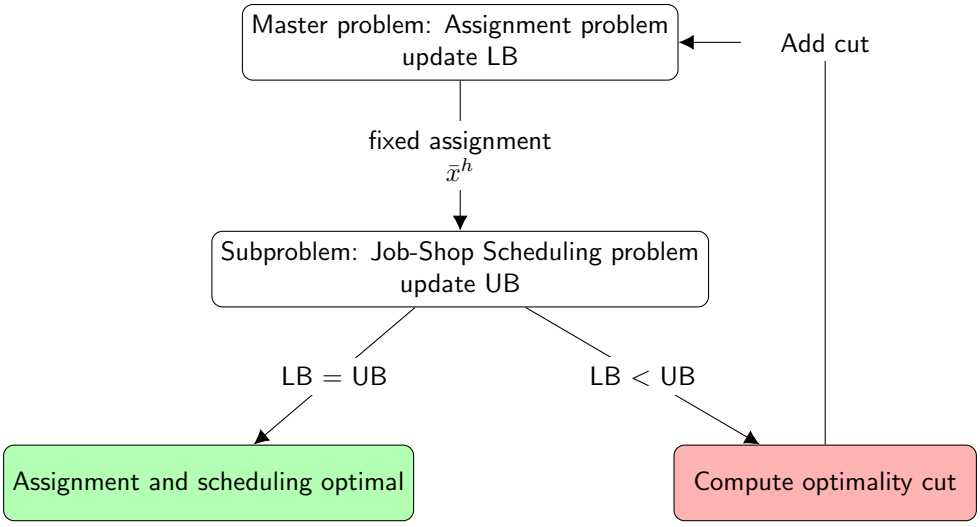

Figure 3: A decomposition scheme for the pFJSSP

In the following, a MILP model is proposed for the master problem. The scheduling subproblems are solved using two different methods: constraint programming for the first version and a dedicated branch-and-bound algorithm for the second version. These methods are named $LBBD_{CPO}$ and $LBBD_{B\&B}$, respectively. Then the question of the generation of optimality cuts is addressed. Finally, we present an extension of the methods for the consideration of the total completion time minimisation problem.

3.2 Master problem

The master problem is a relaxation of the pFJSSP as it only consists in the assignment of operations to machines (i.e., the constraints only involve the assignment variables $x_{i,j,m}$).

Let $m \in \mathcal{M}$ be a machine, $\mathcal{I}'_m \subseteq \mathcal{I}_m$ be a subset of operations that can be processed by m, $r'_m = \min_{O_{i,j} \in \mathcal{I}'_m} r^{min}_{i,j}$ be the shortest release starting date of operations from $\mathcal{I}'_m$ (minimum head) and $q'_m = \min_{O_{i,j} \in \mathcal{I}'_m} q^{min}_{i,j}$ be the shortest delivery time

after $\mathcal{I}'_m$ (minimum tail). The master problem is as follows:

$$\min C_{\max} \tag{21}$$

$$s.t. \quad \sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} = 1 \quad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i \tag{22}$$

$$C_{\max} \geq \sum_{j=1}^{n_i} \sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} \times p_{i,j,m} \quad \forall i \in \mathcal{J} \tag{23}$$

$$C_{\max} \geq r'_m + \sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} \times p_{i,j,m} + q'_m \quad \forall m \in \mathcal{M}, \mathcal{I}'_m \subseteq \mathcal{I}_m \tag{24}$$

$$x_{i,j,m} \in \{0,1\} \quad \forall i \in \mathcal{J},\, O_{i,j} \in \mathcal{O}_i,\, m \in \mathcal{M}_{i,j} \tag{25}$$

Constraints (22) are identical to constraints (2) (processing of each operation by one of the eligible machines). Constraints (23) and (24) express the relaxation of the subproblem as they help to estimate the objective value $C_{\max}$ according to the assignment variables $x_{i,j,m}$. Having a tighter lower bound increases the chances of obtaining high-quality subproblem. Constraints (23) restrict the makespan to be at least equal to the sum of the processing times of operations of the same job. Constraints (24) come from an idea first proposed by Carlier [30] to compute a lower bound for the JSSP: Given a machine m and a subset of operations $\mathcal{I}'_m$, the makespan is at least equal to the sum of the processing times of operations from this subset assigned to m, plus the minimum head and tail.

In practice, for this last set of constraints (24), it is not useful to consider all the possible subsets of $\mathcal{I}_m$. Indeed, based on Theorem 1 below, it is enough to study the sets $\mathcal{I}_a$ created as follows:
for each machine m and for each pair of operations $a = (O_{i,j}, O_{i',j'}) \in \mathcal{I}_m \times \mathcal{I}_m$, let $r_a = \min(r^{min}_{i,j}, r^{min}_{i',j'})$ be the minimum head and $q_a = \min(q^{min}_{i,j}, q^{min}_{i',j'})$ be the minimum tail, and $\mathcal{I}_a = \{O_{i,j} \in \mathcal{I}_m \,|\, r^{min}_{i,j} > r_a,\, q^{min}_{i,j} > q_a\}$ be the set of operations that can be processed by m and of which the shortest release date and the shortest delivery time are larger than the minimum head and tail, respectively.

Theorem 1. *Let $\mathcal{I}'_m$ and $\mathcal{I}''_m$ be two subsets, such that $\mathcal{I}''_m \subset \mathcal{I}'_m$ and $\exists\, O_{i,j} \in \mathcal{I}''_m$ with $r^{min}_{i,j} = r'_m$ and $\exists\, O_{i',j'} \in \mathcal{I}''_m$ with $q^{min}_{i',j'} = q'_m$. Constraint $C_{\max} \geq r'_m + \sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} \times p_{i,j,m} + q'_m$ dominates constraint $C_{\max} \geq r''_m + \sum_{O_{i,j} \in \mathcal{I}''_m} x_{i,j,m} \times p_{i,j,m} + q''_m$.*

Proof. To prove that a constraint $C1$ dominates another one constraint $C2$, it must be shown that $C1$ implies $C2$.

By assumption $r'_m = r''_m$, $q''_m = q'_m$ and, for any assignment, $\sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} \times p_{i,j,m} \geq \sum_{O_{i,j} \in \mathcal{I}''_m} x_{i,j,m} \times p_{i,j,m}$. Hence, $r'_m + \sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} \times p_{i,j,m} + q'_m \geq r''_m +$

$\sum_{O_{i,j} \in \mathcal{I}''_m} x_{i,j,m} \times p_{i,j,m} + q''_m$.

It follows that:
$C_{\max} \geq r'_m + \sum_{O_{i,j} \in \mathcal{I}'_m} x_{i,j,m} \times p_{i,j,m} + q'_m \implies C_{\max} \geq r''_m + \sum_{O_{i,j} \in \mathcal{I}''_m} x_{i,j,m} \times p_{i,j,m} + q''_m$. □

Example 2. *Consider again Example 1 and let suppose that:* $x_{1,1,1} = x_{1,2,3} = x_{1,3,4} = x_{2,1,1} = x_{2,2,2} = x_{2,3,1} = x_{3,1,4} = x_{3,2,2} = x_{3,3,4} = x_{4,1,2} = x_{4,2,4} = x_{4,3,3} = 1$.

Regarding constraints (23), it yields:

- *duration of job 1:* $p_{1,1,1} + p_{1,2,3} + p_{1,3,4} = 2 + 4 + 2 = 8 \implies C_1 \geq 8$;
- *duration of job 2:* $p_{2,1,1} + p_{2,2,2} + p_{2,3,1} = 3 + 1 + 3 = 7 \implies C_2 \geq 7$;
- *duration of job 3:* $p_{3,1,4} + p_{3,2,2} + p_{3,3,4} = 3 + 4 + 2 = 9 \implies C_3 \geq 9$;
- *duration of job 4:* $p_{4,1,2} + p_{4,2,4} + p_{4,3,3} = 2 + 2 + 4 = 8 \implies C_4 \geq 8$.

Therefore, the makespan of the incumbent solution is at least 9, *that is,* $C_{max} \geq 9$.

About constraints (24), we detail the calculations for only one pair $a = (O_{1,1}, O_{2,1})$ *on machine* $M1$.

(1) Identify the head r_a *and tail* q_a*:*

$$r_a = \min(r^{min}_{1,1}, r^{min}_{2,1}) = \min(0, 0) = 0$$
$$q_a = \min(q^{min}_{1,1}, q^{min}_{2,1}) = \min(p^{min}_{1,2} + p^{min}_{1,3}, p^{min}_{2,2} + p^{min}_{2,3}) = \min(4 + 2, 1 + 2) = 3$$

(2) Form subset I_a*:*

$$I_a = \{O_{1,1}, O_{2,1}, O_{3,1}, O_{4,2}\}$$

(3) Deduce a lower bound for the makespan:

$$\begin{aligned} C_{max} &\geq 0 + p_{1,1,1} \times x_{1,1,1} + p_{2,1,1} \times x_{2,1,1} + p_{3,1,1} \times x_{3,1,1} + p_{4,2,1} \times x_{4,2,1} + 3 \\ &\geq 0 + 2 \times 1 + 3 \times 1 + 3 \times 0 + 4 \times 0 + 3 = 8 \end{aligned}$$

Doing the calculations extensively on all machines and for all eligible pairs, we obtain $C_{max} \geq 10$ *(for* $I_{a'} = \{O_{1,2}, O_{4,3}\}$ *on machine* $M3$*).*

At each iteration h, we get a solution of the master problem, i.e., a set of operation-machine assignments. The following additional notations will be useful for the sequel: let $\bar{x}^h_{i,j,m}$ be the value of variable $x_{i,j,m}$ at iteration h. Let $\mathcal{P}^h_m = \{O_{i,j} \in \mathcal{O}_i,\ i \in \mathcal{J} \mid \bar{x}^h_{i,j,m} = 1\}$ be the set of operations assigned to machine m at iteration h.

3.3 Subproblem

The subproblem is a pJSSP. To solve it, two different ways are proposed. We first present a CP model, then we describe a branch-and-bound algorithm that has its origin in a proposal made in Ebadi and Moslehi [7].

3.3.1 Constraint programming: $LBBD_{CPO}$

In the $LBBD_{CPO}$ method, the subproblem is solved using a constraint programming model. This model is similar to the one proposed in Section 2.2. However, in this case, it aims at solving a non-flexible pJSSP since the assignments are already fixed. The variables involved in this model are the following:

- $task_{i,j}$: interval variable between the start and the end of the processing of operation $O_{i,j}$;
- $part_{i,j,k}$: interval variable of unit duration of the processing of the k^{th} part of operation $O_{i,j}$.

Since, the set of eligible machines is reduced to a single machine, unlike the model presented in Section 2.2, the $mode_{i,j,m}$ variables does not appear and $part_{i,j,k}$ are non-optional variables. The subproblem is described as follows:

$$\min z^h \tag{26}$$

$$s.t. \quad z^h \geq task_{i,n_i}.end \qquad \forall i \in \mathcal{J} \tag{27}$$

$$EndBeforeStart(task_{i,j}, task_{i,j+1}) \qquad \forall i \in \mathcal{J},\ O_{i,j} \in \mathcal{O}_i \setminus \{O_{i,n_i}\} \tag{28}$$

$$EndBeforeStart(part_{i,j,k}, part_{i,j,k+1}) \qquad \forall i \in \mathcal{J},\ O_{i,j} \in \mathcal{O}_i,\ 1 \leq k \leq p^h_{i,j} - 1 \tag{29}$$

$$Span(task_{i,j}, part_{i,j,k} : \forall 1 \leq k \leq p^h_{i,j}) \qquad \forall i \in \mathcal{J},\ O_{i,j} \in \mathcal{O}_i \tag{30}$$

$$NoOverlap(part_{i,j,k} : \forall O_{i,j} \in \mathcal{P}^h_m,\ 1 \leq k \leq p^h_{i,j}) \qquad \forall m \in \mathcal{M} \tag{31}$$

where $p^h_{i,j} = \sum_{m \in \mathcal{M}_{i,j}} p_{i,j,m} \times \bar{x}^h_{i,j,m}$.

3.3.2 Branch-and-bound algorithm: $LBBD_{B\&B}$

The resolution algorithm used for the $LBBD_{B\&B}$ method is inspired by Ebadi and Moslehi [7]. In this study, the authors solve the pJSSP using a branch-and-bound algorithm developed on the basis of a disjunctive graph representation. By exploiting the properties of the problem, they introduce effective dominance rules and lower

bounds that make the method efficient. In particular, they have shown that the set of schedules of which preemption is only allowed by a newly available operation is dominant for the problem. The proposed algorithm is briefly presented in the following.

Each operation $O_{i,j}$ of duration $p_{i,j}$ is divided into $p_{i,j}$ operations $\{O_{i,j,1}, \ldots, O_{i,j,p_{i,j}}\}$ of unit duration named unit operations. At each level of the branch-and-bound, the set $\mathcal{V}_a$ of available unit operations (i.e., whose preceding operations have already been scheduled) is formed. The machine m^* processing the unit operation with the earliest availability date (the availability date is determined by the longest path between node *Start* and the unit operation in the partial disjunctive graph) ES is selected. The set of unit operations available in ES on m^* is denoted by $\mathcal{V}'_a$. A lower bound X for the date of the next available unit operation on m^* is calculated. For each unit operation $O_{i,j,k}$ in $\mathcal{V}'_a$, a new node is created where $O_{i,j,k}$ and the $(X-1)$ unit operations following are scheduled from ES. The disjunctive graph is updated (a disjunctive arc between the last unit operation $O_{i,j,p_{i,j}}$ of operation $O_{i,j}$ and the set of other unit operations in $\mathcal{V}'_a$ is added); hence, a lower bound LB of the partial scheduling is obtained to evaluate this new node. If LB is greater than the objective value of the best solution obtained so far, the node is pruned. Otherwise, the algorithm continue at the next level.

Ebadi and Moslehi [7] make the assumption that each job visits each machine at most once (in shop scheduling this is known as *no recirculation*). However, this is not true in our case when solving a subproblem of the pFJSSP. Indeed, during the assignment phase, nothing prevents two operations of the same job from being performed by the same machine (cf. Example 1). This is why we have to adapt the solving algorithm to our specific problem.

3.4 Benders cuts

If z^h is the minimum makespan found by the subproblem, then a trivial Benders cut requires the makespan to be at least z^h, whenever all assignments remain the same. In other words, any solution that achieves a better makespan assigns at least one of the operations to another machine. Thus, the following cut is created and added to the master problem:

$$C_{\max} \geq z^h \Big(1 - \sum_{m \in \mathcal{M}} \sum_{O_{i,j} \in \mathcal{P}^h_m} (1 - \bar{x}^h_{i,j,m})\Big) \tag{32}$$

Inequality (32) is valid in the sense that it does not remove any feasible solution. However, this cut involves all the assignments and leads to the exclusion of only one

solution of the master problem, since the cut becomes useless when even one of the operation assignments is changed. The goal is therefore to strengthen the cut by identifying a subset of assignments responsible for the value of the makespan.

For this purpose, we adapt our algorithm (see Algorithm 1). Once the master problem is solved (*Step* 1), instead of going directly to the subproblem (*Step* 3), the idea proposed by Carlier [30] is again used, as in the set of constraints (24), as a relaxation of the subproblem (*Step* 2). However, at this stage of the search, the assignments are fixed; this allows us to calculate a better approximation of the value of the objective function. Indeed, the duration of each operation $O_{i,j}$ is known: $p^h_{i,j} = \sum_{m \in \mathcal{M}_{i,j}} p_{i,j,m} \times \bar{x}^h_{i,j,m}$, thus it is possible to redefine the shortest release date (resp. shortest delivery time) of an operation $O_{i,j}$ at iteration h as $r^{min,h}_{i,j} = \sum_{j'<j} p^h_{i,j}$ (resp. $q^{min,h}_{i,j} = \sum_{j'>j} p^h_{i,j}$).

Let $m \in \mathcal{M}$ be a machine, $a = (O_{i,j}, O_{i',j'}) \in \mathcal{P}^h_m \times \mathcal{P}^h_m$ be a pair of operations processed by machine m. Let us define $r_a = min(r^h_{i,j}, r^h_{i',j'})$ as the minimum head, $q_a = min(q^h_{i,j}, q^h_{i',j'})$ as the minimum tail, and $\mathcal{P}_a = \{O_{i,j} \in \mathcal{P}^h_m \,|\, m \in \mathcal{M}_{i,j},\, r^h_{i,j} > r_a,\, q^h_{i,j} > q_a\}$ as the set of operations that are processed by m and of which the shortest release date and the delivery time are smaller than the minimum head and the minimum tail, respectively. $LB_a = r_a + \sum_{O_{i,j} \in \mathcal{P}_a} p_{i,j,m} + q_a$ is a lower bound on the optimum value for the z^h objective value in the pJSSP subproblem.

Therefore, if LB_a is superior to the incumbent best solution found, it is useless to solve the subproblem since it is impossible to improve the best known solution. Instead of solving the subproblem, it is required to add the following cut:

$$C_{\max} \geq LB_a \Big(1 - \sum_{m \in \mathcal{M}} \sum_{O_{i,j} \in \mathcal{S}_a \cap \mathcal{P}^h_m} (1 - \bar{x}^h_{i,j,m})\Big) \tag{33}$$

where $\mathcal{S}_a$ is the set of operations belonging to $\mathcal{P}_a$, but also the operations of jobs with at least one operation belonging to this set (because they have an influence on the calculation of r_a or q_a).

Example 3. *Consider Example 1 again and assume the same assignments as in Example 2, that is:* $\bar{x}^h_{1,1,1} = \bar{x}^h_{1,2,3} = \bar{x}^h_{1,3,4} = \bar{x}^h_{2,1,1} = \bar{x}^h_{2,2,2} = \bar{x}^h_{2,3,1} = \bar{x}^h_{3,1,4} = \bar{x}^h_{3,2,2} = \bar{x}^h_{3,3,4} = \bar{x}^h_{4,1,2} = \bar{x}^h_{4,2,4} = \bar{x}^h_{4,3,3} = 1$.

Algorithm 1: Logic-based Benders decomposition algorithm

Initialisation Master constraints ←{constraint sets (22), (23), (24)} ;
$incumbent\_solution \leftarrow$ solution found using a heuristic ;
$h \leftarrow 0$;
Step 1 *Master problem*
  Solve the assignment master problem until a new feasible solution $\mathcal{P}^h$ is found ;
Step 2 *Subproblem relaxation*
  foreach $m \in \mathcal{M}$ **do**
    foreach *pair* $a = (O_{i,j}, O_{i',j'}) \in \mathcal{P}_m^h \times \mathcal{P}_m^h$ **do**
      $r_a \leftarrow min(r_{i,j}^h, r_{i',j'}^h)$;
      $q_a \leftarrow min(q_{i,j}^h, q_{i',j'}^h)$;
      $\mathcal{P}_a \leftarrow \{O_{i,j} \in \mathcal{P}_m^h \mid r_{i,j}^h > r_a,\, q_{i,j}^h > q_a\}$;
      $LB_a \leftarrow r_a + \sum_{O_{i,j} \in \mathcal{P}_a} p_{i,j,m} + q_m^a$;
      if $LB_a > incumbent\_solution.value$ **then**
        Generate cut (33) and add it to master constraints ;
        Go back to Step 1 ;
Step 3 *Subproblem*
  $current\_solution \leftarrow$ Solve the pJSSP subproblem with $\mathcal{P}^h$ assignments ;
  if $current\_solution.value < incumbent\_solution.value$ **then**
    $incumbent\_solution \leftarrow current\_solution$;
  Generate cut (32) and add it to master constraints ;
  Go back to Step 1

We focus on machine $M1$ and we take into account the pair $a = (O_{1,1}, O_{2,1})$:

$$\begin{aligned} LB_a = r_a & + \sum_{O_{i,j} \in \mathcal{P}_a} p_{i,j,1} + q_a \\ = \min(r^h_{1,1}, r^h_{2,1}) & + p_{1,1,1} + p_{2,1,1} + \min(q^h_{1,1}, q^h_{2,1}) \\ = \min(r^h_{1,1}, r^h_{2,1}) & + p_{1,1,1} + p_{2,1,1} + \min(p^h_{1,2} + p^h_{1,3}, p^h_{2,2} + p^h_{2,3}) \\ = \min(0, 0) & + 2 + 3 + \min(4 + 2, 1 + 3) = 9 \end{aligned}$$

This leads to the following cut:

$$C_{\max} \geq 9 \times \Big(1 - \sum_{m \in \mathcal{M}} \sum_{O_{i,j} \in (\mathcal{O}_1 \cup \mathcal{O}_2) \cap \mathcal{P}^h_m} (1 - \bar{x}^h_{i,j,m})\Big)$$

In Example 3, although the considered subset of operations assigned to machine $M1$ is identical to the one studied in Example 2, the bound obtained is tighter because the values r_a and q_a are calculated differently. Since the assignments are fixed, only the current assignment selection $\mathcal{P}^h$ is taken into consideration to calculate the shortest release date $r^h_{i,j}$ and shortest delivery time $q^h_{i,j}$ of an operation $O_{i,j}$ (i.e., there is no need to consider all the possible assignments of previous and following operations).

3.5 Total completion time

Considering the total completion time (i.e., the sum of the completion times of all the jobs) as the objective function to minimise, the decomposition scheme presented in Section 3.1 remains identical. However, adjustments have to be done to solve both the master problem and the subproblems.

3.5.1 Master problem

The master problem needs to be adapted to the new objective function. First of all, the criterion (21) is replaced by:

$$\min C_{\text{total}} \tag{34}$$

where C_{total} is the total completion time.

Then we used two relaxations for the master problem. The first one is based on the idea that the completion time C_i of a job i is at least equal to the sum of the

processing times of operations belonging to job i, $C_i \geq \sum_{m\in\mathcal{M}} \sum_{j=1}^{n_i} x_{i,j,m} \times p_{i,j,m}$, $\forall i \in \mathcal{J}$. Constraints (23) are thus replaced by the following:

$$C_{\text{total}} \geq \sum_{i\in\mathcal{J}} \sum_{j=1}^{n_i} \sum_{m\in\mathcal{M}_{i,j}} x_{i,j,m} \times p_{i,j,m} \tag{35}$$

The second relaxation of the subproblem is based on the consideration of a single-machine problem for each machine. On the one hand, by definition, the completion time of a job is not less than the completion time of one of its operations plus the tail of this operation: $C_i \geq C_{i,j} + q_{i,j}^{min}, \forall i \in \mathcal{J}, O_{i,j} \in \mathcal{O}_i$. Hence, we deduce: $C_i \geq \frac{1}{n_i} \times \sum_{j=1}^{n_i}(C_{i,j} + q_{i,j}^{min}), \forall i \in \mathcal{J}$ and $C_{\text{total}} \geq \sum_{i\in\mathcal{J}} \frac{1}{n_i} \times \sum_{j=1}^{n_i}(C_{i,j} + q_{i,j}^{min})$. Let $\underline{C}_m$ be a lower bound on the total completion time of operations on machine m. According to the previous statement, we can say that:

$$C_{\text{total}} \geq \frac{1}{\max_{i\in\mathcal{J}} n_i} \times \Big(\sum_{m\in\mathcal{M}} \underline{C}_m + \sum_{i\in\mathcal{J}} \sum_{j=1}^{n_i} q_{i,j}^{min} \Big) \tag{36}$$

On the other hand, the Shortest Processing Time first (SPT) rule is known to be optimal for the single-machine problem when the objective is to minimise the total completion time. Thus, for each machine m, let π_m be a permutation of $\mathcal{I}_m$ such that $p_{\pi_m(1)} \leq p_{\pi_m(2)} \leq \cdots \leq p_{\pi_m(|\mathcal{I}_m|)}$. Hence, $\underline{C}'_m = \sum_{k=0}^{|\mathcal{I}_m|} C_{\pi_m(k)}$ where

$$C_{\pi_m(k)} \geq \Big(\sum_{l=0}^{k} p_{\pi_m(l)} \times x_{\pi_m(l)} \Big) \times x_{\pi_m(k)} \tag{37}$$

is a lower bound on the total completion time of operations on machine m. Terms $C_{\pi_m(k)}$ occur only for those operations that are assigned to machine m, that is why we introduce the assignment variables $x_{\pi_m(k)}$. We linearise by writing :

$$C_{\pi_m(k)} \geq \sum_{l=0}^{k} (p_{\pi_m(l)} \times x_{\pi_m(l)}) - B_{\pi_m(k)} \times (1 - x_{\pi_m(k)}) \tag{38}$$

and $C_{\pi_m(k)} \geq 0$, the big-M term $B_{\pi_m(k)}$ is given by:

$$B_{\pi_m(k)} = \sum_{l=0}^{k-1} p_{\pi_m(l)} \tag{39}$$

Finally, we sum over all machines and constraints (24) are replaced by constraints (38), (39), and the following:

$$C_{\text{total}} \geq \frac{1}{\max_{i \in \mathcal{J}} n_i} \Big(\sum_{m \in M} \sum_{k=0}^{|\mathcal{I}_m|} C_{\pi_m(k)} + \sum_{i \in \mathcal{J}} \sum_{j=1}^{n_i} q_{i,j}^{min} \Big) \tag{40}$$

3.5.2 Subproblem

The subproblem must also fit the objective function. In particular, in the subproblem of the $LBBD_{CPO}$ version, it is enough to replace constraints (27) of the CP model by:

$$z^h \geq \sum_{i \in \mathcal{J}} task_{i,n_i}.end \tag{41}$$

Regarding the $LBBD_{B\&B}$ version, some adaptations must be made in the branch-and-bound algorithm. Specifically, the objective function value is given by the total length of the longest paths from node *Start* to the last unit operation of each job, in the disjunctive graph. At each node, a lower bound is calculated in a similar way in the partial disjunctive graph.

4 Numerical experiments

For computational tests, all experiments are performed on three cluster nodes with Intel Xeon E5-2695 v4 CPU at 2.1 GHz. All algorithms presented are implemented in C++, using CPLEX 12.10 for the MILP models and CP Optimizer (CPO) 12.10 for the CP models. CPU time and RAM were respectively limited to 1 hour and 16 GB.

A naive heuristic allows us to obtain an upper bound for the objective value. This solution is used at the beginning of the search in the decomposition methods. For the purpose of comparison between methods, and for the sake of fairness, we also use this solution as a warm start for the CP method. Therefore, the instances are solved using the CP model presented in Section 2 with warm start solution (method $warm\_CPO$) and without (CPO), and by the two versions of the decomposition proposed in Section 3 ($LBBD_{CPO}$ and $LBBD_{B\&B}$).

To evaluate and compare the diverse methods, we consider various performance measures:

- the average optimality gap, computed as follows:

$$gap^{method} = \frac{UB^{method} - LB^{method}}{UB^{method}} \tag{42}$$

- the proportion of optima found;
- the relative percentage deviation compared to the best solution found. It aims to evaluate the quality of the upper bounds found by each method. It is therefore computed as follows:

$$RD^{method} = \frac{UB^{method} - UB^{*}}{UB^{*}} \tag{43}$$

  where UB^{*} is the best bound found among all tested methods;
- the proportion of instances for which the best solution (among the tested methods) is found.

Moreover, two objective functions are considered: the minimisation of the makespan and the minimisation of the total completion time.

4.1 Instances

Numerous benchmark instances for the FJSSP without preemption are available in the literature. These instances serve as a reference to evaluate our propositions in the preemptive case.

Brandimarte [31] introduced instances, referred to here as "*BrandimarteMk*", of which data are randomly generated using a uniform distribution within a given limit. Hurink et al. [10] provided an adaptation of some of the classical JSSP instances [32, 33, 34, 35, 36] and generate three data sets, denoted by "*HurinkEdata*", "*HurinkRdata*" and "*HurinkVdata*", with increasing flexibility, by expanding the pool of eligible machines for some of the operations. Dauzère-Pérès and Paulli [37] designed a data set "*DPpaulli*", in which the number of operations per job is greater than the number of machines, and whose parameters are randomly generated using a uniform distribution within given limits. Barnes and Chambers [38] created a set of instances "*ChambersBarnes*" based on three instances of the classical JSSP problem [32, 33] transformed into FJSSP instances by duplicating some machines. Kacem et al. [39] created the instances "*Kacem*" with total flexibility (i.e., each machine is able to process each operation). Furthermore, for this benchmark, it is worth mentioning that it is also distinguished by the very small number of instances it contains

(4). Lastly, Fattahi et al. [40] proposed randomly generated small- and medium-size problems ("*Fattahi*").

Characteristics of these benchmarks are summarised in Table 2. For each benchmark, we report the number of instances (276 in total) that compose it, the size of these instances (in terms of number of jobs and number of machines), as well as the flexibility degree of the instance, defined as the average number of eligible machines per operation. Detailed information on these instances is presented in Behnke and Geiger [41].

Data set	Reference	Number of instances	Size		Flexibility
			Jobs	Machines	
BrandimarteMk	Brandimarte [31]	15	[10 ... 30]	[4 ... 15]	[1.4 ... 4.1]
HurinkEdata	Hurink et al. [10]	66	[6 ... 30]	[4 ... 15]	[1.1 ... 1.2]
HurinkVdata	Hurink et al. [10]	66	[6 ... 30]	[4 ... 15]	[2 ... 6.7]
DPpaulli	Dauzère... [37]	18	[10 ... 20]	[5 ... 10]	[1.1 ... 5]
ChambersBarnes	Barnes... [38]	21	[10 ... 15]	[11... 18]	[1 ... 1.3]
Kacem	Kacem et al. [39]	4	[4 ... 15]	[5 ... 10]	[5 ... 10]
Fattahi	Fattahi et al. [40]	20	[2 ... 12]	[2 ... 8]	[1.5 ... 2.7]
Total		*276*	*[2 ... 30]*	*[2 ... 18]*	*[1 ... 10]*

Table 2: FJSSP instances characteristics

4.2 Makespan

Figures 4 to 7 show a graphical visualisation of the performance indicators for the minimisation of the makespan. In these figures, the results are grouped by type of benchmark. Figure 4 presents the average optimality gap for each method. We observe that the proposed decomposition methods outperform CP methods for all benchmarks. Globally, if we consider together the 276 instances, CPO method reaches in average an optimality gap of 23% (22% for $warm\_CPO$) in comparison with 10% for the $LBBD_{CPO}$ method and 6% for the $LBBD_{B\&B}$.

Figure 5 illustrates the proportion of optima found for each benchmark. Overall, $LBBD_{B\&B}$ is able to solve to optimality a larger number of instances within the limited time: 104 compared to 51 for $LBBD_{CPO}$, 36 for CPO and 41 for $warm\_CPO$. Note that none of the methods succeeds in solving even a single instance for benchmarks from Dauzère-Pérès and Paulli [37] and Barnes and Chambers [38].

Figure 6 provides the proportion of best solutions found among all methods. In contrast to the optimality gap, if we consider the number of best solutions found

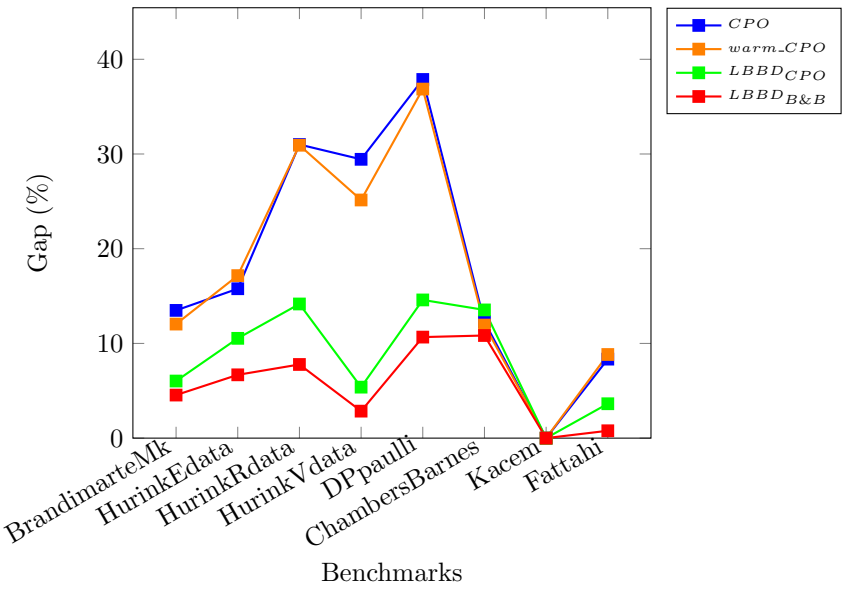

Figure 4: Makespan minimisation – Average optimality gap for each method according to benchmarks

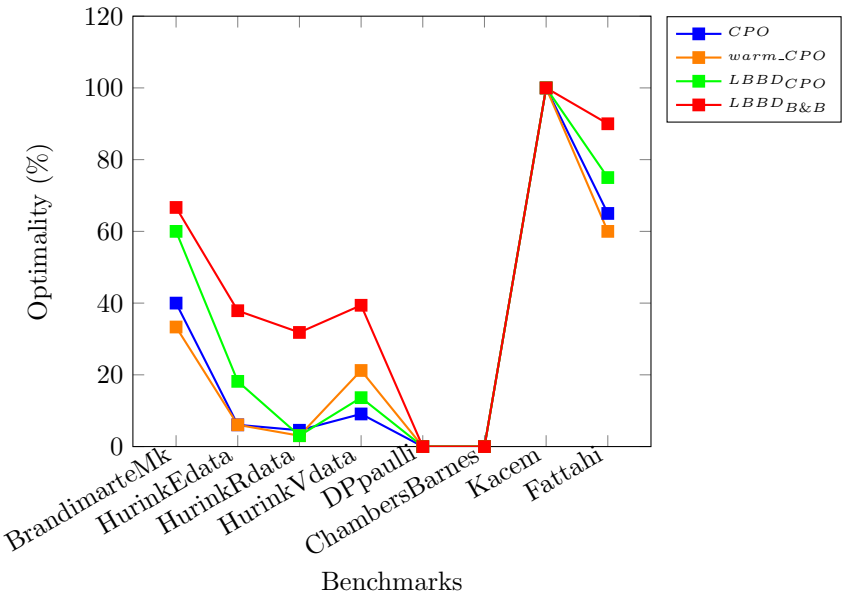

Figure 5: Makespan minimisation – Proportion of optima found for each method according to benchmarks

as an indicator of performance, method $LBBD_{CPO}$ (with 65 best solutions found over all instances) does not clearly dominate CPO (with 56 best solutions) and gets even worse results than $warm\_CPO$ (with 90 best solutions). However, method $LBBD_{B\&B}$ still outperforms all these methods (with 194 best solutions) and finds at least half of the best solutions for each benchmark.

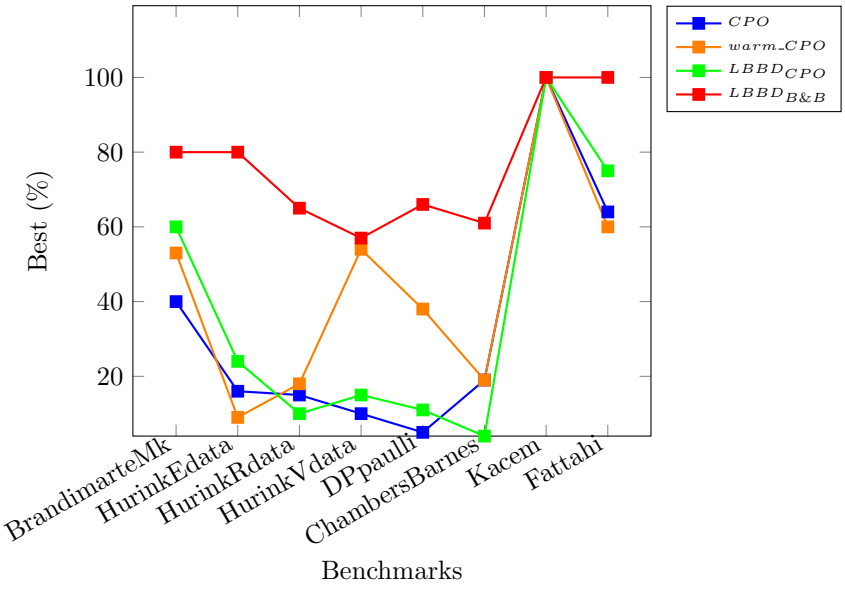

Figure 6: Makespan minimisation – Proportion of best solutions found for each method according to benchmarks

We illustrate the relative percentage deviation compared to the best solution found in Figure 7. It shows that $LBBD_{B\&B}$ always gets high-quality solutions and never exceeds an average of 2% deviation for each benchmark. For the other methods, the quality of the solutions obtained is quite variable depending on the benchmark studied.

Based on these results we deduce that the proposed decomposition $LBBD_{B\&B}$ outperforms the CP model for makespan minimisation. For all studied benchmarks, the $LBBD_{B\&B}$ method obtains a greater number of optimal solutions (Figure 5), a greater number of best solutions (Figure 6), a smaller average gap (Figure 4), and a smaller relative deviation to the best solution found (Figure 7).

Although the $LBBD_{B\&B}$ method achieves better performances, we notice a great variability of the results depending on the benchmarks. Therefore, we propose to fo-

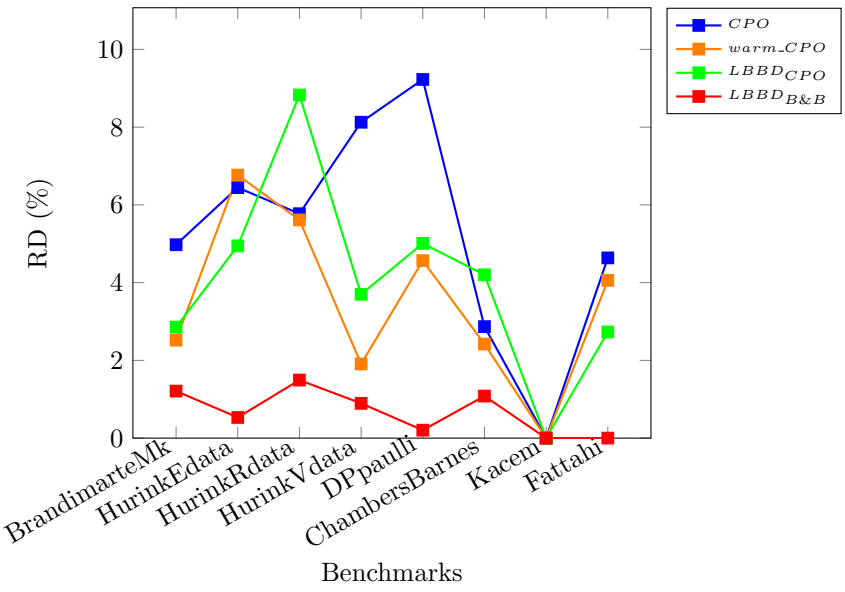

Figure 7: Makespan minimisation – Average relative deviation to the best solution found for each method according to benchmarks

cus on the possible factors. In particular, Figure 8 permits us to study the optimality gap according to the average number of operations per machine. Each point of this graph represents an instance of the FJSSP solved by one of the four studied methods, with as abscissa the average number of operations per machine of the instance and as ordinate the obtained optimality gap for it with the involved method.

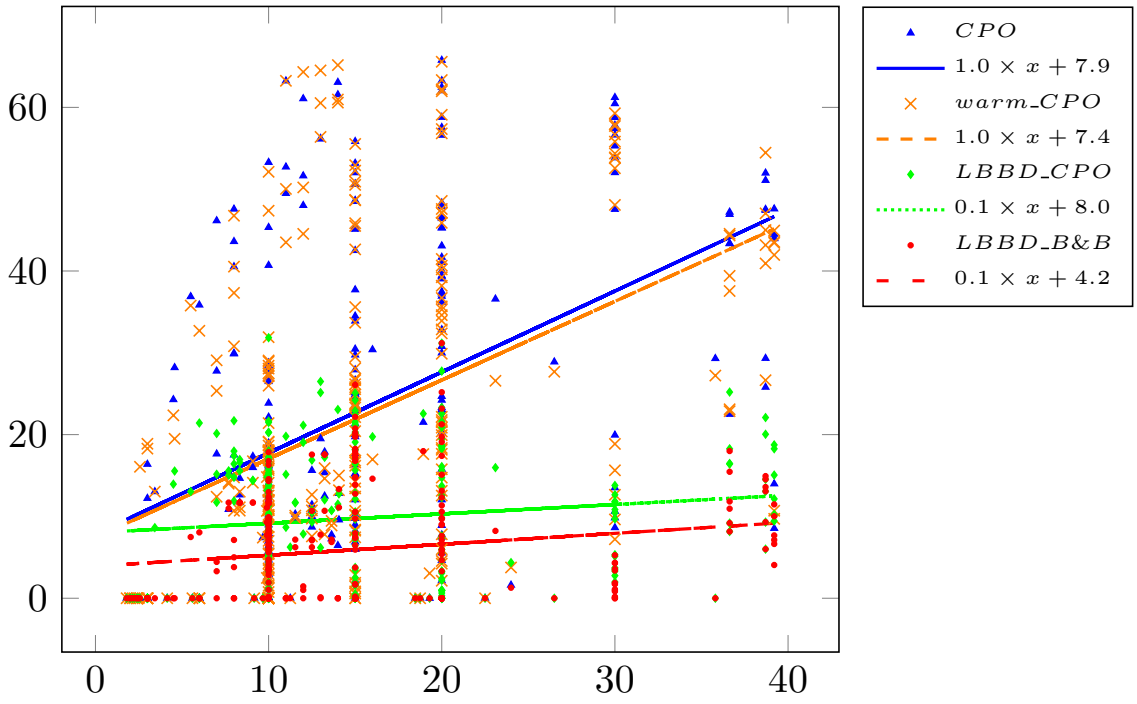

Figure 8: Makespan minimisation – Gap according to the average number of operations per machine for each method

The observation of Figure 8 confirms what can be seen in Figure 4, namely that the decomposition-based methods achieve a better optimality gap than pure constraint programming methods. Moreover, using the linear regression performed for each method, we notice that problems become more difficult to solve as the number of operations per machine increases. However, the efficiency of the decomposition methods is less sensitive to that factor than the CP methods. Indeed, while the slope of the linear regression relating the optimality gap obtained by the CP-based methods (*warm_CPO* and *CPO*) according to the number of operations per machine is

1.0, this same coefficient goes down to 0.1 for the LBBD-based methods ($LBBD_{CPO}$ and $LBBD_{B\&B}$).

Finally, all the results presented above show the superiority of the $LBBD_{B\&B}$ method over the $LBBD_{CPO}$ method. In Table 3, we report, for these two methods, the proportion of time spent in solving the subproblems (SP(%)), the proportion of time spent in solving the master problems (MP(%)), the average number of subproblems solved (SP number), and the average time spent to solve a subproblem (time per SP) for each studied benchmark. As expected, the time spent to solve the subproblems is much smaller for methods using the branch-and-bound algorithm, which explains its better performance. We also notice that for the most complicated instances (*DPpaulli* and *ChambersBarnes*), very few subproblems are considered (only 1 in most cases), which means that, whatever the method used, the imposed time limit of 1 hour is not enough to optimally solve these subproblems. By crossing this information with Figure 6, we note that even for these complicated instances method $LBBD_{B\&B}$ obtains better quality solutions. We deduce that, in addition to proving the optimality of a solution more efficiently, the branch-and-bound algorithm allows finding better solutions in a given time than the CP method for the considered objective.

Table 3: Makespan minimisation – Performances of decomposition methods for each benchmark

Benchmarks	$LBBD_{CPO}$				$LBBD_{B\&B}$			
	SP(%)	MP(%)	SP number	time per SP	SP(%)	MP(%)	SP number	time per SP
BrandimarteMk	99.94	0.07	3	502.9	99.81	0.2	69	16.3
HurinkEdata	100	0.01	3	861.8	100	0.01	42	54
HurinkRdata	99.99	0.02	35	100.1	93.19	6.82	2855	0.9
HurinkVdata	99.84	0.17	9	326.6	69.11	30.9	2858	0.4
DPpaulli	99.57	0.44	1	2809	99.71	0.3	1	2810.5
ChambersBarnes	99.99	0.02	1	3599.5	100	0.01	1	2614.8
Kacem	16.67	83.34	9	0.1	0	100	9	0
Fattahi	99.99	0.02	38	30.4	0.14	99.87	2364	0.1

4.3 Total completion time

As presented in Section 3.5, we adapt all methods to the problem of minimising the total completion time. For this objective function, we illustrate all the considered performance measures in Figures 9–12.

Contrary to the makespan minimisation problem, no method clearly outperforms the others when minimising the total completion time. In general, the observed performances, for instance in Figures 9 and 10, are worse for this objective function, as it makes the problem more difficult. In more detail, we can observe in Figure 9 that very few instances are solved to optimality, only 6% of them, considering together the four methods under study. Among them, the methods that prove the optimality for the largest number of instances are $warm\_CPO$ and CPO, which obtain exactly the same results (with 15 instances over 276) followed by $LBBD_{B\&B}$ (14 instances), and finally $LBBD_{CPO}$ (9 instances). In addition, the optimality gap is very wide, for many instances and all methods, as shown in Figure 10. We also note that the average gap per benchmark is similar for each method, which results in very close overall average gaps: 39% for the $warm\_CPO$ method, 38% for CPO, 37% for $LBBD_{CPO}$, and 36% for $LBBD_{B\&B}$.

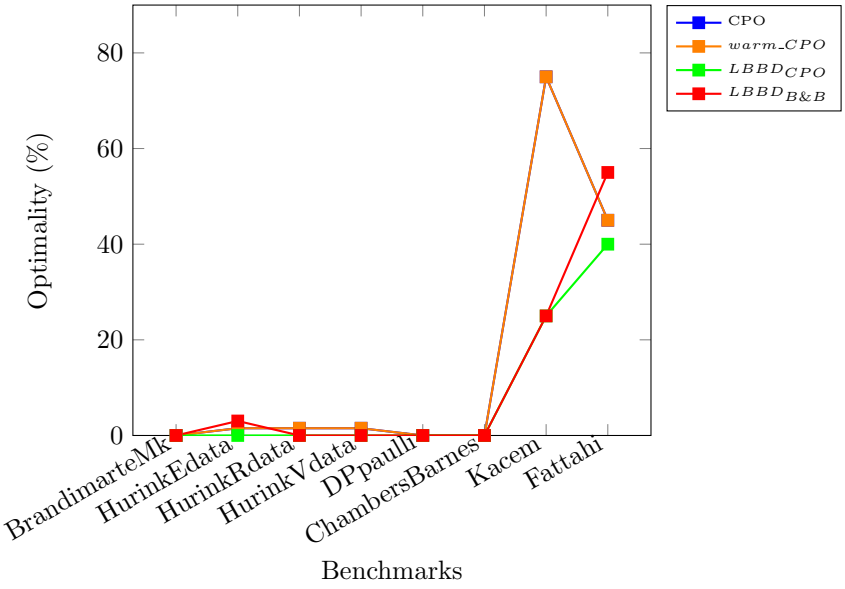

Figure 9: Total completion time minimisation – Proportion of optima found for each method according to benchmarks

Figures 11 and 12 focus on relative performances of the methods concerning the quality of the returned solutions. Again, none of the methods stand out, however, we notice a different behaviour for the CP methods compared to the LBBD methods. For both indicators (number of best solutions and relative deviation) and for most

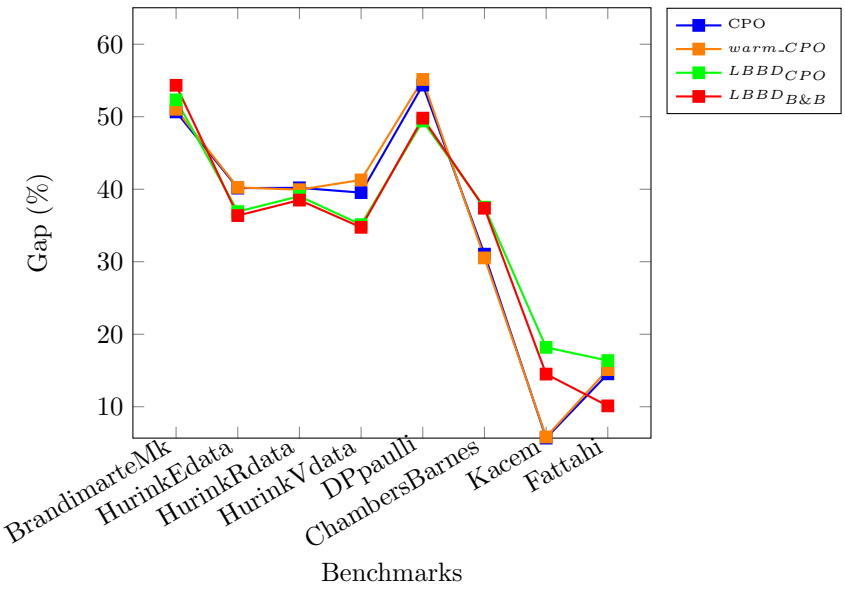

Figure 10: Total completion time minimisation – Average optimality gap for each method according to benchmarks

benchmarks, either the CP methods dominate, or the LBBD methods perform better.

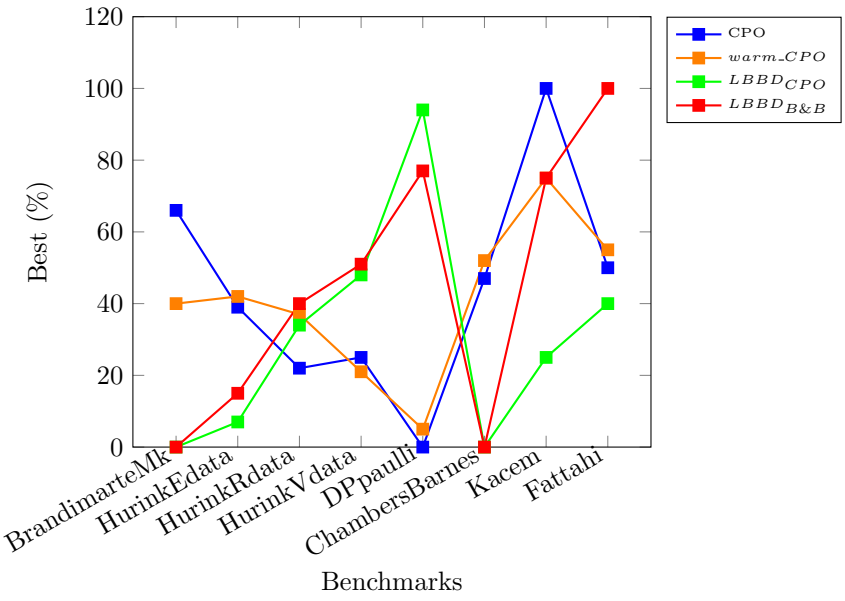

Figure 11: Total completion time minimisation – Proportion of best solutions found for each method according to benchmarks

5 Conclusions

In this paper, we present several exact methods to solve the preemptive flexible job-shop scheduling problem with two different objective functions, the makespan minimisation and the total completion time minimisation. We model the problem with both mixed-integer linear programming and constraint programming. Our main contribution consists in proposing a logic-based Benders decomposition algorithm by splitting the problem into an assignment master problem and a non-flexible scheduling subproblem. Concerning the makespan minimisation, the decomposition makes possible the use of the most powerful procedure in the literature to solve the preemptive job-shop subproblem, namely a branch-and-bound algorithm. With this method, the proposed decomposition outperforms the constraint programming model both in terms of optimality gap and in terms of the quality of the solutions found. We extend these solution methods to the total completion time minimisation problem. Due to

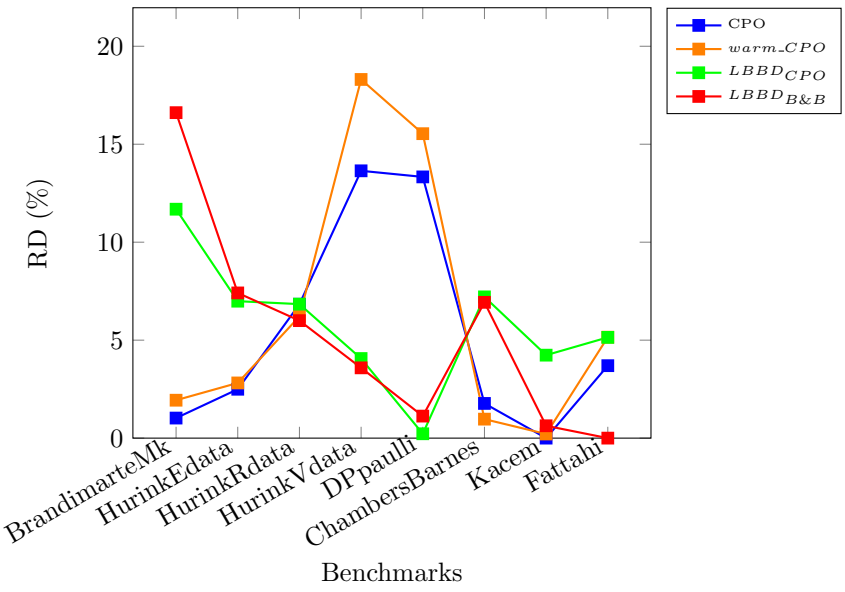

Figure 12: Total completion time minimisation – Average relative deviation to the best solution found for each method according to benchmarks

the greater complexity of this problem, none of the proposed methods can achieve satisfactory results.

Future works could explore a way to take advantage of the proposed decomposition for the total completion time objective. Heuristics are also needed for the purpose of providing a warm-start solution for exact methods, or for simply yielding an approximate solution to compare against.

Acknowledgements

The authors are indebted to Abbas Ebadi and Ghasem Moslehi who very kindly provided their branch-and-bound code.

References

[1] C. Juvin, L. Houssin, P. Lopez, Logic-based benders decomposition for the preemptive flexible job-shop scheduling problem, Computers & Operations Research 152 (2023) 106156.

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

[3] A. Jain, S. Meeran, Deterministic job-shop scheduling: Past, present and future, European Journal of Operational Research 113 (1999) 390–434.

[4] C. Le Pape, P. Baptiste, Resource constraints for preemptive job-shop scheduling, Constraints 3 (1998) 263–287.

[5] Y. S. Yun, Genetic algorithm with fuzzy logic controller for preemptive and non-preemptive job-shop scheduling problems, Computers & Industrial Engineering 43 (2002) 623–644.

[6] A. Ebadi, G. Moslehi, Mathematical models for preemptive shop scheduling problems, Computers & Operations Research 39 (2012) 1605–1614.

[7] A. Ebadi, G. Moslehi, An optimal method for the preemptive job shop scheduling problem, Computers & Operations Research 40 (2013) 1314–1327.

[8] I. A. Chaudhry, A. A. Khan, A research survey: review of flexible job shop scheduling techniques, International Transactions in Operational Research 23 (2016) 551–591.

[9] L. Shen, S. Dauzère-Pérès, J. S. Neufeld, Solving the flexible job shop scheduling problem with sequence-dependent setup times, European Journal of Operational Research 265 (2018) 503–516.

[10] J. Hurink, B. Jurisch, M. Thole, Tabu search for the job-shop scheduling problem with multi-purpose machines, OR Spectrum 15 (1994) 205–215.

[11] F. Pezzella, G. Morganti, G. Ciaschetti, A genetic algorithm for the flexible job-shop scheduling problem, Computers & Operations Research 35 (2008) 3202–3212.

[12] C. Xianzhou, Y. Zhenhe, An improved genetic algorithm for dual-resource constrained flexible job shop scheduling, in: Fourth International Conference on Intelligent Computation Technology and Automation, volume 1, IEEE, 2011, pp. 42–45.

[13] N. Najid, S. Dauzère-Pérès, A. Zaidat, A modified simulated annealing method for flexible job shop scheduling problem, in: IEEE International Conference on Systems, Man and Cybernetics, volume 5, IEEE, 2002, pp. 6–pp.

[14] M. Yazdani, M. Zandieh, R. Tavakkoli-Moghaddam, F. Jolai, Two meta-heuristic algorithms for the dual-resource constrained flexible job-shop scheduling problem, Scientia Iranica 22 (2015) 1242–1257.

[15] C. Özgüven, L. Özbakır, Y. Yavuz, Mathematical models for job-shop scheduling problems with routing and process plan flexibility, Applied Mathematical Modelling 34 (2010) 1539–1548.

[16] M. Ziaee, A mixed integer linear programming model for flexible job shop scheduling problem, International Journal of Mathematical and Computational Sciences 12 (2018) 95–99.

[17] L. Meng, C. Zhang, Y. Ren, B. Zhang, C. Lv, Mixed-integer linear programming and constraint programming formulations for solving distributed flexible job shop scheduling problem, Computers & Industrial Engineering 142 (2020) 106347. doi:10.1016/j.cie.2020.106347.

[18] D. Kress, D. Müller, Mathematical models for a flexible job shop scheduling problem with machine operator constraints, IFAC-PapersOnLine 52 (2019) 94–99.

[19] J. Zhang, J. Yang, Flexible job-shop scheduling with flexible workdays, preemption, overlapping in operations and satisfaction criteria: An industrial application, International Journal of Production Research 54 (2016) 4894–4918.

[20] K. Jansen, M. Mastrolilli, R. Solis-Oba, Approximation algorithms for flexible job shop problems, International Journal of Foundations of Computer Science 16 (2005) 361–379.

[21] J. Hooker, H. Yan, V. Saraswat, P. Van Hentenryck, Verifying logic circuits by Benders decomposition, in: Principles and Practice of Constraint Programming: The Newport Papers, MIT Press, Cambridge, MA, 1995, pp. 267–288.

[22] J. Hooker, Logic based methods for optimization: Combining optimization and constraint satisfaction, John Wiley and Sons, New York, 2000.

[23] J. N. Hooker, G. Ottosson, Logic-based Benders decomposition, Mathematical Programming 96 (2003) 33–60.

[24] J. N. Hooker, Planning and scheduling by logic-based Benders decomposition, Operations Research 55 (2007) 588–602.

[25] Y. Tan, D. Terekhov, Logic-based Benders decomposition for two-stage flexible flow shop scheduling with unrelated parallel machines, in: Canadian Conference on Artificial Intelligence, Springer, 2018, pp. 60–71.

[26] B. Naderi, V. Roshanaei, Critical-path-search logic-based Benders decomposition approaches for flexible job shop scheduling, INFORMS Journal on Optimization 4 (2022) 1–28.

[27] E. H. Bowman, The schedule-sequencing problem, Operations Research 7 (1959) 621–624.

[28] O. Polo-Mejía, C. Artigues, P. Lopez, V. Basini, Mixed-integer/linear and constraint programming approaches for activity scheduling in a nuclear research facility, International Journal of Production Research 58 (2020) 7149–7166.

[29] J. N. Hooker, Integrated methods for optimization, volume 100 of *International Series in Operations Research and Management Science*, Springer, 2007.

[30] J. Carlier, The one-machine sequencing problem, European Journal of Operational Research 11 (1982) 42–47.

[31] P. Brandimarte, Routing and scheduling in a flexible job shop by tabu search, Annals of Operations Research 41 (1993) 157–183.

[32] H. Fisher, G. Thompson, Probabilistic learning combinations of local job-shop scheduling rules, in: J. Muth, G. Thompson (Eds.), Industrial Scheduling, Prentice-Hall, Englewood Cliffs, New Jersey, 1963, pp. 225–251.

[33] S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (Supplement), Graduate School of Industrial Administration, Pittsburgh, Pennsylvania, Carnegie-Mellon University, 1984.

[34] J. Adams, E. Balas, D. Zawack, The shifting bottleneck procedure for job shop scheduling, Management Science 34 (1988) 391–401.

[35] J. Carlier, É. Pinson, An algorithm for solving the job-shop problem, Management Science 35 (1989) 164–176.

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

[37] S. Dauzère-Pérès, J. Paulli, An integrated approach for modeling and solving the general multiprocessor job-shop scheduling problem using tabu search, Annals of Operations Research 70 (1997) 281–306.

[38] J. Barnes, J. Chambers, Flexible job shop scheduling by tabu search, Technical Report ORP96-09, The University of Texas at Austin, 1996.

[39] I. Kacem, S. Hammadi, P. Borne, Pareto-optimality approach for flexible job-shop scheduling problems: Hybridization of evolutionary algorithms and fuzzy logic, Mathematics and Computers in Simulation 60 (2002) 245–276.

[40] P. Fattahi, M. S. Mehrabad, F. Jolai, Mathematical modeling and heuristic approaches to flexible job shop scheduling problems, Journal of Intelligent Manufacturing 18 (2007) 331–342.

[41] D. Behnke, M. J. Geiger, Test instances for the flexible job shop scheduling problem with work centers, Arbeitspapier, Helmut-Schmidt-Universität, Universität der Bundeswehr Hamburg, Institut für betriebliche Logistik und Organisation, 2012. URL: `https://books.google.fr/books?id=RvWFzQEACAAJ`.