Hybrid Methods to Solve the Two-Stage Robust Flexible Job-Shop Scheduling Problem with Budgeted Uncertainty

Carla Juvin, Laurent Houssin, Pierre Lopez

Hybrid Methods to Solve the Two-Stage Robust Flexible Job-Shop Scheduling Problem with Budgeted Uncertainty

Carla Juvin[1], Laurent Houssin[1,2][a] and Pierre Lopez[1][b]
[1]*LAAS-CNRS, Université de Toulouse, CNRS, Toulouse, France*
[2]*ISAE-SUPAERO, Toulouse, France*
{*carla.juvin, laurent.houssin, pierre.lopez*}*@laas.fr*

Keywords: Flexible Job-Shop Scheduling, Robust Optimization, Uncertainty Budget, Mixed Integer Linear Programming, Constraint Programming.

Abstract: This paper addresses the robust flexible job-shop scheduling problem considering uncertain operation processing times associated with an uncertainty budget. Exact solution methods based on mixed integer linear programming and constraint programming are proposed to solve the problem. Such solutions are hybridized in the framework of a two-stage robust optimization, and a column and constraint generation algorithm is used to solve representative instances. The experimental results show the advantages of a two-stage approach where constraint programming and integer programming are mixed to solve a master problem and a subproblem, respectively.

1 INTRODUCTION

The job-shop scheduling problem is a well studied and NP-hard problem where a set of jobs are to be processed on a set of machines (Garey et al., 1976). Each job is composed of a sequence of operations that must be processed on machines with given processing times in a given job-dependent order, and each machine can process only one operation at a time. The flexible job-shop problem (FJSSP) is a generalization of the job-shop scheduling problem: 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. The FJSSP has received considerable attention and both metaheuristics and exact methods have been developed to solve the problem, the majority of them with the assumption that the parameters are deterministically known (Brandimarte, 1993; Dauzère-Pérès and Paulli, 1997). However, in the real world, many sources of uncertainty (processing times variation, machine breakdown, addition of new operations, etc.) can affect the quality and even the feasibility of a schedule.

There exist two major approaches to deal with data uncertainty: stochastic optimization and robust optimization. While stochastic optimization considers probability distribution, robust optimization assumes that uncertain data belong to a given uncertainty set and aims to optimize performance considering the worst-case scenario within that set.

In this paper, we propose exact solution methods to solve the robust flexible job-shop scheduling problem. A two-stage robust optimization is used to deal with processing times uncertainty, where the first stage fixes the assignment and the sequence of operations on machines whilst the second stage sets the operation start times. We provide two robust counterpart models based on mixed-integer linear programming and constraint programming formulations, as well as a column and constraint generation algorithm. A discussion is conducted on the basis of an analysis of experimental results.

2 PROBLEM DEFINITION

2.1 Deterministic Flexible Job-Shop Scheduling Problem

We first introduce the flexible job-shop scheduling problem (FJSSP) with makespan minimization. An instance of the FJSSP implies a set of jobs $\mathcal{J}$ and a set of machines $\mathcal{M}$. Each job $i \in \mathcal{J}$ consists of a sequence

[a] https://orcid.org/0000-0001-5975-7639
[b] https://orcid.org/0000-0003-0413-3188

of n_i operations. The j^{th} operation $O_{i,j} \in O_i$ of a job i must be performed by one of the machines 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 not allowed: once an operation is started, it must be processed without any interruption. The objective is to find an assignment for each operation and a sequence on each machine that minimize the maximum completion time or makespan $C_{\max}$.

Notations and definitions

$\mathcal{J}$	Set of jobs
$\mathcal{M}$	Set of machines
n_i	Number of operations in job i ($i \in \mathcal{J}$)
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 $O_{i,j}$ ($i \in \mathcal{J}, O_{i,j} \in O_i$)
$p_{i,j,m}$	Processing time of operation $O_{i,j}$ on machine m ($i \in \mathcal{J}, O_{i,j} \in O_i, m \in \mathcal{M}_{i,j}$)
I_m	Set of operations that can be processed by machine m ($m \in \mathcal{M}$)
H	Big constant

2.2 Flexible Job-Shop Scheduling Problem with Uncertain Processing Times

We consider that the processing times of operations are uncertain. Each processing time $p_{i,j,m}$ of an operation $O_{i,j} \in O_i, i \in \mathcal{J}$, on machine $m \in \mathcal{M}_{i,j}$, belongs to the interval $[\bar{p}_{i,j,m}, \bar{p}_{i,j,m} + \hat{p}_{i,j,m}]$, where $\bar{p}_{i,j,m}$ is the nominal value and $\hat{p}_{i,j,m}$ the maximum deviation of the processing time from its nominal value.

Example 1. *Consider an FJSSP instance with 3 jobs and 2 machines. The intervals $[\bar{p}_{i,j,m}, \bar{p}_{i,j,m} + \hat{p}_{i,j,m}]$ of processing times $p_{i,j,m}$ of operations $O_{i,j} \in O_i$, $i \in \mathcal{J}$ on each eligible machine $m \in \mathcal{M}_{i,j}$, are given in Table 1.*

Table 1: Numerical example of an instance of the FJSSP: operation processing times.

		$M1$	$M2$
$J1$	$O_{1,1}$	[43; 86]	–
	$O_{1,2}$	[87; 170]	[95; 190]
$J2$	$O_{2,1}$	[63; 142]	[53; 166]
	$O_{2,2}$	–	[73; 131]
$J3$	$O_{3,1}$	[125; 239]	[135; 224]
	$O_{3,2}$	[43; 73]	[61; 174]

2.2.1 Uncertainty Budget

The traditional robust optimization approach (Soyster, 1973) consists in protecting against the case when all parameters can deviate at the same time, which makes the solution overly conservative. Indeed, there is a very low probability that all parameters take their worst value all together. To overcome this limitation, (Bertsimas and Sim, 2004) introduce an uncertainty budget approach that allows a restriction on the number of deviations that can occur simultaneously to a given budget. In order to reach a trade-off between robustness and solution quality, we exploit this approach to define the uncertainty set.

Let Γ be the budget of uncertainty, the maximum number of operations whose processing time deviation can occur simultaneously. The worst-case scenario is always an extremum scenario (Ben-Tal et al., 2009). Thus, for each scenario ξ, the processing time of operation $O_{i,j}$ on machine m is given by:

$$p_{i,j,m}(\xi) = \bar{p}_{i,j,m} + \xi_{i,j} \cdot \hat{p}_{i,j,m}$$

where $\xi_{i,j}$ is equal to 1 if the processing time of the operation deviates, 0 otherwise.

We define the uncertainty set $\mathcal{U}^\Gamma$ as:

$$\mathcal{U}^\Gamma = \{(\xi_{i,j})_{i \in \mathcal{J}, 1 \leq j \leq n_i} \mid \sum_{i \in \mathcal{J}} \sum_{j=1}^{n_i} \xi_{i,j} \leq \Gamma\}$$

2.2.2 Multi-Stage Robust Optimization

Multi-stage robust optimization has been introduced by (Ben-Tal et al., 2004). In some optimization problems, only part of the decision variables have to be fixed before uncertainty is revealed, while the other variables can be chosen after the realization and can thus be adjusted to the scenario. The authors introduce the *adjustable robust counterpart*; the set of decision variables is split into "here and now" decisions and "wait and see" decisions. The objective is to find a solution for the "here and now" decision variables such that constraints involving uncertain parameters remain feasible for all values of the uncertain parameters, and minimizing the objective value.

In our problem, we consider that the purpose is to find the assignment and sequence on the machines (first stage: here and now), allowing to define a start time for each operation and each scenario (second stage: wait and see), minimizing the makespan in the worst-case scenario considering the budget of uncertainty.

Example 2. *Considering Example 1, a feasible solution is the assignment of operations $O_{1,1}$, $O_{3,1}$, $O_{3,2}$ to machine M_1, and of operations $O_{2,1}$, $O_{2,2}$, $O_{1,2}$ to machine M_2, following these sequences. Figure 1 represents this solution when all processing times take their*

nominal value; the makespan is 221. Considering an uncertainty budget $\Gamma = 2$, the worst case, for this solution, is that the processing time of operations $O_{2,1}$ and $O_{1,2}$ deviate and take their greatest value. Figure 2 shows the Gantt chart in this case. The objective function value of this solution for an uncertainty budget $\Gamma = 2$ reaches the makespan equal to 429.

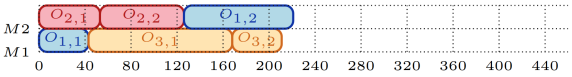

Figure 1: A feasible solution for Example 1, when all processing times take their nominal value.

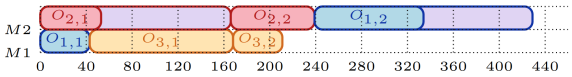

Figure 2: From the solution of Figure 1, the worst-case solution with an uncertainty budget of 2.

3 ROBUST COUNTERPART FOR THE FJSSP

Two robust counterparts of the flexible job-shop scheduling problem are presented. They are respectively based on a Mixed Integer Linear Programming formulation and a Constraint Programming model for the deterministic FJSSP.

3.1 Mixed Integer Linear Programming Model

We define a Mixed Integer Linear Programming (MILP) extended robust model, which is directly inspired by the sequence-based model made by (Shen et al., 2018). The first stage decision variables are defined as follows:

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

$$y_{i,j,i',j'} = \begin{cases} 1 & \text{if operation } O_{i,j} \text{ is processed before } O_{i',j'}; \\ 0 & \text{otherwise} \end{cases}$$

while the start time $t_{i,j}(\xi) \in \mathbb{R}_+$ of each operation $O_{i,j}$ in scenario ξ is part of the second stage variables.

The MILP model is as follows:

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

s.t.

$$\sum_{m \in \mathcal{M}_{i,j}} x_{i,j,m} = 1 \quad \forall i \in \mathcal{J}, \forall j \in O_i \tag{2}$$

$$\sum_{m \in \mathcal{M}_{i,j-1}} x_{i,j-1,m} \cdot p_{i,j-1,m}(\xi) + t_{i,j-1}(\xi) \leq t_{i,j}(\xi)$$
$$\forall i \in \mathcal{J}, \forall j \in O_i, \forall \xi \in \mathcal{U}^\Gamma \tag{3}$$

$$t_{i',j'}(\xi) + p_{i',j',m}(\xi) - (2 - x_{i,j,m} - x_{i',j',m} + y_{i,j,i',j'}) \cdot H \leq t_{i,j}(\xi) \quad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in I_m^2, \forall \xi \in \mathcal{U}^\Gamma \tag{4}$$

$$t_{i,j}(\xi) + p_{i,j,m}(\xi) - (3 - x_{i,j,m} - x_{i',j',m} - y_{i,j,i',j'}) \cdot H \leq t_{i',j'}(\xi) \quad \forall m \in \mathcal{M}, \forall (O_{i,j}, O_{i,j'}) \in I_m^2, \forall \xi \in \mathcal{U}^\Gamma \tag{5}$$

$$t_{i,n_i}(\xi) + \sum_{m \in \mathcal{M}_{i,n_i}} x_{i,n_i,m} \cdot p_{i,n_i,m}(\xi) \leq C_{\max}$$
$$\forall i \in \mathcal{J}, \forall \xi \in \mathcal{U}^\Gamma \tag{6}$$

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

$$y_{i,j,i',j'} \in \{0,1\} \quad \forall (i,i') \in \mathcal{J}^2, \forall j \in O_i, \forall j' \in O_{i'} \tag{8}$$

The objective (1) is to minimize the makespan $C_{\max}$. Constraints (2) are assignment constraints and ensure that each operation is assigned to exactly one machine. Constraints (3) ensure the precedence relation between two consecutive operations of the same job. Constraints (4) and (5) are the disjunctive constraints and avoid overlapping of operations scheduled on the same machine. Note that Constraints (3), (4) and (5) are duplicated for each feasible scenario. Finally, Constraints (6) allow to compute the greatest makespan among all feasible scenarios and therefore to determine the optimal solution in the worst case. Finally, Constraints (7) and (8) define the domain of the variables.

3.2 Constraint Programming Model

While Constraint Programming (CP) obtains very good results in scheduling, to the best of our knowledge it has been never used in robust multi-stage models. The following model is directly inspired from (Kress and Müller, 2019) for the deterministic FJSSP with machine operator constraints. It involves the following variables:

- $task_{i,j,\xi}$: interval variable between the start and the end of the processing of operation $O_{i,j}$ in scenario ξ;
- $mode_{i,j,m,\xi}$: interval variable between the start and the end of the processing of operation $O_{i,j}$ on machine m in scenario ξ (since the operations have multiple eligible machines and must be executed by exactly one of them, this is an optional variable);
- $seqs_{m,\xi}$: sequence variable of tasks scheduled on machine m in scenario ξ.

The CP model is as follows:

$$\min C_{\max} \tag{9}$$

s.t.

$$C_{\max} \geq task_{i,n_i,\xi}.end \;\; \forall i \in \mathcal{J}, \forall \xi \in \mathcal{U}^{\Gamma} \tag{10}$$

$$EndBeforeStart(task_{i,j,\xi}, task_{i,j+1,\xi})$$
$$\forall i \in \mathcal{J}, \forall O_{i,j} \in O_i \setminus \{O_{i,n_i}\}, \forall \xi \in \mathcal{U}^{\Gamma} \tag{11}$$

$$Alternative(task_{i,j,\xi}, mode_{i,j,m,\xi} : \forall m \in \mathcal{M}_{i,j})$$
$$\forall i \in \mathcal{J}, \forall O_{i,j} \in O_i, \forall \xi \in \mathcal{U}^{\Gamma} \tag{12}$$

$$NoOverlap(seqs_{m,\xi}) \;\; \forall m \in \mathcal{M}, \forall \xi \in \mathcal{U}^{\Gamma} \tag{13}$$

$$PresenceOf(mode_{i,j,m,0}) \Rightarrow PresenceOf(mode_{i,j,m,\xi})$$
$$\forall i \in \mathcal{J}, \forall O_{i,j} \in O_i, \forall m \in \mathcal{M}_{i,j}, \forall \xi \in \mathcal{U}^{\Gamma} \setminus \{0\} \tag{14}$$

$$SameSequence(seqs_{m,0}, seqs_{m,\xi})$$
$$\forall m \in \mathcal{M}, \forall \xi \in \mathcal{U}^{\Gamma} \setminus \{0\} \tag{15}$$

Constraints (10) allow the determination of the makespan, which is equal to the end of the last operation. Constraints (11) ensure the precedence relation between two consecutive operations of the same job. Constraints (12) ensure that each operation is assigned to exactly one machine. Constraints (13) ensure that in each scenario each machine performs at most only one operation at the same time. Constraints (10–13) are duplicated for each scenario. Constraints (14) and (15) ensure that the assignment of each operation and the sequence on each machine are the same in each scenario. For these last two constraints, the first scenario $\xi = 0$ is used as reference.

4 COLUMN AND CONSTRAINT GENERATION APPLIED TO THE FJSSP

We present here a column and constraint generation approach to solve the robust flexible job-shop scheduling problem. This method has been introduced by (Zeng and Zhao, 2013) to solve two-stage robust optimization problems. The procedure splits the problem into a master problem and an adversarial subproblem. The idea is to solve the robust counterpart problem (or master problem), for a limited subset of scenarios, and then to identify which scenarios, if any, make the found solution infeasible, using an adversarial subproblem. Then, these scenarios are included in the master problem by generating the corresponding recourse decision variables on the fly. This process repeats until a solution that is feasible for all scenarios is found (Hamaz et al., 2018; Duarte et al., 2020; Silva et al., 2020; Levorato et al., 2022).

4.1 Column and Constraint Algorithm

For the FJSSP, the master problem can be defined as one of the extended models presented in Section 3, for which only a subset of scenarios is considered, i.e. each robust constraint ((3–6) for the MILP model or (10–13) for the CP model) is defined only for a subset of scenarios. An optimal solution for this master problem allows us to obtain a lower bound on the makespan for the global problem. In order to verify the feasibility of this solution, given the assignments and sequences on the machines fixed in the master problem, we check whether one of the possible scenarios leads to a greater makespan than the one found in the master problem. If such a scenario exists, then the variables and constraints associated with it are generated and added to the master problem. In our case, the search for such a scenario can be seen as the search for the scenario that leads to the largest possible makespan.

Figure 3 depicts the scheme of the column and constraint generation algorithm, where LB represents the lower bound and UB the upper bound.

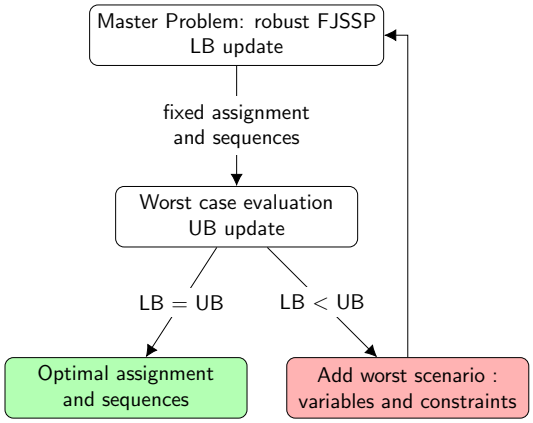

Figure 3: Column and constraint algorithm.

4.2 Worst-Case Evaluation

Having assigned and sequenced operations on machines via the master problem, we know all precedence relations between operations, both the precedences in the jobs induced by the problem, and the precedences on the machines fixed in the master problem. The resulting problem corresponds to the evaluation of the worst-case scenario: Given an uncertainty budget Γ, the aim is to identify the Γ deviating operations that lead to the greatest makespan.

4.2.1 Mixed Integer Linear Programming

We introduce a MILP model with the following decisions variables:

$$\xi_{i,j} = \begin{cases} 1 & \text{if processing time of operation } O_{i,j} \\ & \text{deviates;} \\ 0 & \text{otherwise} \end{cases}$$

$$b_{i,j,i',j'} = \begin{cases} 1 & \text{if operation } O_{i,j} \text{ starts at the end} \\ & \text{of operation } O_{i',j'}; \\ 0 & \text{otherwise} \end{cases}$$

$$d_i = \begin{cases} 1 & \text{if the last operation } O_{i,n_i} \text{ of job } i \text{ is the} \\ & \text{last operation among all to be completed;} \\ 0 & \text{otherwise} \end{cases}$$

Let also $A_{i,j}$ be the set of immediate predecessors of operation $O_{i,j}$. The MILP model developed for the worst-case scenario evaluation is as follows:

$$\max\ C_{\max} \tag{16}$$

s.t.

$$\sum_{i\in\mathcal{J}}\sum_{j\in O_i} \xi_{i,j} = \Gamma \tag{17}$$

$$t_{i,j} - (t_{i',j'} + \bar{p}_{i',j'} + \xi_{i,'j'} \cdot \hat{p}_{i',j'}) \leq H \cdot (1 - b_{i,j,i',j'}) \quad \forall i \in \mathcal{J}, \forall j \in O_i, \forall (i',j') \in A_{i,j} \tag{18}$$

$$\sum_{(i',j')\in A_{i,j}} b_{i,j,i',j'} \geq 1 \quad \forall i \in \mathcal{J}, \forall j \in O_i \mid A_{i,j} \neq \emptyset \tag{19}$$

$$t_{i,1} = 0 \quad \forall i \in \mathcal{J} \mid A_{i,0} = \emptyset \tag{20}$$

$$C_{\max} - (t_{i,n_i} + \bar{p}_{i,n_i} + \xi_{i,n_i} \cdot \hat{p}_{i,n_i}) \leq H \cdot (1 - d_i) \quad \forall i \in \mathcal{J} \tag{21}$$

$$\xi_{i,j} \in \{0,1\} \quad \forall i \in \mathcal{J}, \forall j \in O_i \tag{22}$$

$$b_{i,j,i',j'} \in \{0,1\} \quad \forall i \in \mathcal{J}, \forall j \in O_i, \forall (i',j') \in A_{i,j} \tag{23}$$

$$d_i \in \{0,1\} \quad \forall i \in \mathcal{J} \tag{24}$$

The objective is to find the scenario (the values of $\xi_{i,j}$ for each operation $O_{i,j}$) that maximizes the makespan (16). Constraints (17) ensure that exactly Γ operations have their processing time deviating. Constraints (18) and (19) ensure that each operation, which has at least one predecessor, starts at the maximum time among the completion time of the predecessors. Constraints (20) ensure that operations with no predecessor start at time 0. Constraints (21) ensure that the makespan is equal to the latest completion time among all jobs.

4.2.2 Constraint Programming

We introduce a CP model with optional interval variables $dev_{i,j}$, which are present only if the processing time of the corresponding operation $O_{i,j}$ takes its maximum value (i.e. the operation deviates from its nominal value). Let us define the decision variables used to model the subproblem as follows:

- $task_{i,j}$, interval variable associated to operation $O_{i,j}$ of duration $\bar{p}_{i,j,m}$;
- $dev_{i,j}$, optional interval variable of duration $\hat{p}_{i,j,m}$, present if operation $O_{i,j}$ deviates from its nominal value.

We also use function $HeightAtStart$, which gives the contribution of an interval variable to a cumulative function at its start point, and the cumulative function $StepAtStart$, which defines the contribution of an interval at its start.

$$\max C_{\max} \tag{25}$$

s.t.

$$task_{i,1}.start = 0 \quad \forall i \in \mathcal{J} \mid A_{i,1} = \emptyset \tag{26}$$

$$task_{i,j}.start = \max(\{task_{i',j'}.end \mid O_{i',j'} \in A_{i,j}\}) \quad \forall i \in \mathcal{J}, \forall O_{i,j} \in O_i \tag{27}$$

$$StartAtEnd(dev_{i,j}, task_{i,j}) \quad \forall i \in \mathcal{J}, \forall O_{i,j} \in O_i \tag{28}$$

$$\sum_{i\in\mathcal{J}}\sum_{j\in O_i} HeightAtStart(dev_{i,j}, StepAtStart(dev_{i,j}, 1)) = \Gamma \tag{29}$$

$$C_{\max} = \max(\cup_{\forall i\in\mathcal{J}}(\{task_{i,n_i}.end\} \cup \{dev_{i,n_i}.end\})) \tag{30}$$

The objective is to find the scenario (by determining which optional interval variables $dev_{i,j}$ are present) that maximizes the makespan (25). Constraints (27) and (28) ensure that each operation starts as soon as possible, given the assignment and the sequencing determined by the master problem, and the precedence constraints. Constraints (29) ensure that if the interval variable $dev_{i,j}$ is present, meaning that the processing time of the corresponding operation $O_{i,j}$ deviates from its nominal value, this processing time is equal to its maximum value. Constraints (29) ensure that exactly Γ operations take their maximum values for their processing times. Finally, Constraints (30) ensure that the makespan is equal to the latest completion time among all jobs.

5 EXPERIMENTAL RESULTS

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 are respectively limited to 1 hour and 16 GB.

5.1 Instances and Methods

The instances we used for our tests are small- and medium-size problems instances for the FJSSP taken from (Fattahi et al., 2007) (whose features are described in Table 2), adapted to the robust problem by randomly generating values for the deviations. Let $\bar{p}_{max}$ be the maximum nominal processing time for all operations and all machines. For each operation $O_{i,j}$ and each eligible machine $m \in \mathcal{M}_{i,j}$, we randomly generate a value for $\hat{p}_{i,j,m}$ within a range from 25 % to 100 % of the value of $\bar{p}_{max}$. We repeat this operation three times; we therefore obtain three new instances from one initial deterministic instance. We generate in total 60 instances that we use for our tests. We tested all the models by varying the budget of uncertainty according to four ratios, namely 20 %, 40 %, 60 % and 80 % of the number of operations rounded down to the nearest integer, giving a total of 240 experiments per method.

In addition to the two extended models presented in Section 3, we evaluate four combinations of the column and constraint algorithm presented in Section 4. Table 3 presents the name and configuration of each tested method (*ccg* stands for column and constraint generation).

Table 2: Characteristics of FJSSP instances.

Instance	#Jobs	#Machines	#Operations
SFJS1	2	2	4
SFJS2	2	2	4
SFJS3	3	2	6
SFJS4	3	2	6
SFJS5	3	2	6
SFJS6	3	3	9
SFJS7	3	5	9
SFJS8	3	4	9
SFJS9	3	3	9
SFJS10	4	5	12
MFJS1	5	6	15
MFJS2	5	7	15
MFJS3	6	7	18
MFJS4	7	7	21
MFJS5	7	7	21
MFJS6	8	7	24
MFJS7	8	7	32
MFJS8	9	8	36
MFJS9	11	8	44
MFJS10	12	8	48

Table 3: Methods description.

Method name	Method type	Robust counterpart	Sub-problem
milp	extended	milp	–
cp	extended	cp	–
ccg_milp	ccg	milp	milp
ccg_milp_cp	ccg	milp	cp
ccg_cp	ccg	cp	cp
ccg_cp_milp	ccg	cp	milp

5.2 Results

Table 4 presents general results comparing the performance of the methods for all instances, by reporting the number of instances that each method is able to solve within the time limit. If we focus on the extended methods, we notice that the CP formulation performs slightly better than the MILP method. Secondly, we note that methods based on column and constraint generation allow to solve considerably more instances than extended methods. More specifically, those using a CP formulation for the master problem are the ones with the best results. Finally, we observe that the resolution of the adversarial subproblem with MILP provides a marginally better performance than with the CP model.

Table 5 reports, for each original instance, the number of robust instances (among the 12) solved to optimality by each method and the average time to achieve it. Not surprisingly, we observe that the more the size of the instances increases the more difficult they are to solve, visible by a decreasing number of solved instances and an increasing CPU time. Extended methods manage to solve all instances up to 4 jobs and 5 machines while 'ccg_cp_milp' solves all instances up to 6 jobs and 7 machines.

Table 6 aims to visualize the performance of the methods according to the given uncertainty budget. We notice that for the extended methods, the resolution is more difficult for medium budget (40 % and 60 %); this is due to the fact that these uncertainty budgets generate a greater number of scenarios and, in these methods, the number of decision variables is directly related to the number of scenarios. However, for methods based on the column and constraint generation algorithm, the number of instances solved to optimality increases with the uncertainty budget. It shows that the number of scenarios has no direct ef-

Table 4: Methods performance comparison (number of instances, out of 240, solved to optimality).

Method	#Solv.
milp	130
cp	133
ccg_milp	167
ccg_milp_cp	166
ccg_cp	190
ccg_cp_milp	193

fect on the difficulty of the problem.

6 CONCLUSIONS

In this paper, we consider the two-stage flexible jobshop scheduling problem where the operation processing times are subject to uncertainty. The first stage is devoted to fixing the assignment and sequencing decisions; the second stage determines the start time of the operations.

As a main contribution, we introduce two robust counterpart formulations to solve the robust scheduling problem, including one based on constraint programming. We use a column and constraint generation algorithm mixing both integer linear programming and constraint programming. Experimental results show that the best method, among those tested in this study, is the constraint and column generation method using constraint programming in the master problem and an integer linear programming model for the subproblem.

REFERENCES

Ben-Tal, A., El Ghaoui, L., and Nemirovski, A. (2009). *Robust optimization*, volume 28. Princeton university press.

Ben-Tal, A., Goryashko, A., Guslitzer, E., and Nemirovski, A. (2004). Adjustable robust solutions of uncertain linear programs. *Mathematical Programming*, 99(2):351–376.

Bertsimas, D. and Sim, M. (2004). The price of robustness. *Operations Research*, 52(1):35–53.

Brandimarte, P. (1993). Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41(3):157–183.

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

Duarte, J. L. R., Fan, N., and Jin, T. (2020). Multi-process production scheduling with variable renewable integration and demand response. *European Journal of Operational Research*, 281(1):186–200.

Fattahi, P., Mehrabad, M. S., and Jolai, F. (2007). Mathematical modeling and heuristic approaches to flexible job shop scheduling problems. *Journal of Intelligent Manufacturing*, 18(3):331–342.

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.

Hamaz, I., Houssin, L., and Cafieri, S. (2018). The cyclic job shop problem with uncertain processing times. In *16th International Conference on Project Management and Scheduling (PMS 2018)*, pages 119–122, Rome, Italy.

Kress, D. and Müller, D. (2019). Mathematical models for a flexible job shop scheduling problem with machine operator constraints. *IFAC-PapersOnLine*, 52(13):94–99.

Levorato, M., Figueiredo, R., and Frota, Y. (2022). Exact solutions for the two-machine robust flow shop with budgeted uncertainty. *European Journal of Operational Research*, 300(1):46–57.

Shen, L., Dauzère-Pérès, S., and Neufeld, J. S. (2018). Solving the flexible job shop scheduling problem with sequence-dependent setup times. *European Journal of Operational Research*, 265(2):503–516.

Silva, M., Poss, M., and Maculan, N. (2020). Solution algorithms for minimizing the total tardiness with budgeted processing time uncertainty. *European Journal of Operational Research*, 283(1):70–82.

Soyster, A. L. (1973). Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157.

Zeng, B. and Zhao, L. (2013). Solving two-stage robust optimization problems using a column-and-constraint generation method. *Operations Research Letters*, 41(5):457–461.

Table 5: Methods performance comparison grouping by instances (number of instances, out of 12, solved to optimality).

Instance	milp		cp		ccg_milp		ccg_milp_cp		ccg_cp		ccg_cp_milp	
	#Solv.	t(s)	#Solv.	t(s)	#Solv.	t(s)	#Solv.	t(s)	#Solv.	t(s)	#Solv.	t(s)
SFJS1	12	0.04	12	0.03	12	0.06	12	0.98	12	1.13	12	0.05
SFJS2	12	0.03	12	0.03	12	0.03	12	0.25	12	0.38	12	0.04
SFJS3	12	0.4	12	0.08	12	0.5	12	3.03	12	3.17	12	0.11
SFJS4	12	0.25	12	0.05	12	0.23	12	1.65	12	1.77	12	0.08
SFJS5	12	0.83	12	0.12	12	1.03	12	2.93	12	3.8	12	0.2
SFJS6	12	4.55	12	0.44	12	1.49	12	5.62	12	4.89	12	0.25
SFJS7	12	1.3	12	0.25	12	0.31	12	1.46	12	1.73	12	0.11
SFJS8	12	7.32	12	0.62	12	1.79	12	5.68	12	6.22	12	0.31
SFJS9	12	11.5	12	2.05	12	1.42	12	2.47	12	3.19	12	0.16
SFJS10	12	128	12	3.1	12	2.01	12	8.59	12	11.5	12	0.2
MFJS1	6	1115	6	1445	12	257	12	297	12	73.1	12	19.9
MFJS2	4	1246	6	858	12	161	12	199	12	44.6	12	9.55
MFJS3	0	-	1	1301	11	549	10	414	11	174	12	99.5
MFJS4	0	-	0	-	5	849	5	1715	11	539	11	450
MFJS5	0	-	0	-	5	694	6	1089	10	544	10	328
MFJS6	0	-	0	-	2	1643	1	2522	7	255	9	739
MFJS7	0	-	0	-	0	-	0	-	4	1006	4	815
MFJS8	0	-	0	-	0	-	0	-	3	1144	3	880
MFJS9	0	-	0	-	0	-	0	-	0	-	0	-
MFJS10	0	-	0	-	0	-	0	-	0	-	0	-

Table 6: Methods performance comparison grouping by uncertainty budget (number of instances, out of 60, solved to optimality).

Uncertainty budget	milp	cp	ccg_milp	ccg_milp_cp	ccg_cp	ccg_cp_milp
20 %	35	37	38	38	42	42
40 %	30	31	39	39	45	48
60 %	30	30	42	43	49	49
80 %	35	35	48	46	54	54