Filtering Rules for Flow Time Minimization in a Parallel Machine Scheduling Problem

Margaux Nattaf, Arnaud Malapert

Filtering rules for flow time minimization in a Parallel Machine Scheduling Problem

Margaux Nattaf

Univ. Grenoble Alpes, CNRS, Grenoble INP, G-SCOP, 38000 Grenoble, France
margaux.nattaf@grenoble-inp.fr

Arnaud Malapert

Université Côte d'Azur, CNRS, I3S, France
arnaud.malapert@unice.fr

November 19, 2020

Abstract

This paper studies the scheduling of jobs of different families on parallel machines with qualification constraints. Originating from semi-conductor manufacturing, this constraint imposes a time threshold between the execution of two jobs of the same family. Otherwise, the machine becomes disqualified for this family. The goal is to minimize both the flow time and the number of disqualifications. Recently, an efficient constraint programming model has been proposed. However, when priority is given to the flow time objective, the efficiency of the model can be improved.

This paper uses a polynomial-time algorithm which minimize the flow time for a single machine relaxation where disqualifications are not considered. Using this algorithm one can derived filtering rules on different variables of the model. Experimental results are presented showing the effectiveness of these rules. They improve the competitiveness with the mixed integer linear program of the literature.

keywords : Parallel Machine Scheduling, Job families , Flow time , Machine Disqualification, Filtering Algorithm , Cost-Based Filtering.

1 Introduction

This paper considers the scheduling of job families on parallel machines with time constraints on machine qualifications. In this problem, each job belongs to a family and a family can only be executed on a subset of qualified machines.

In addition, machines can lose their qualifications during the schedule. Indeed, if no job of a family is scheduled on a machine during a given amount of time, the machine lose its qualification for this family. The goal is to minimize the sum of job completion times, i.e. the flow time, while maximizing the number of qualifications at the end of the schedule.

This problem, called scheduling Problem with Time Constraints (PTC), is introduced in [11]. It comes from the semiconductor industries. Its goal is to introduce constraints coming from Advanced Process Control (APC) into a scheduling problem. APC's systems are used to control processes and equipment to reduce variability and increase equipment efficiency. In PTC, qualification constraints and objective come from APC and more precisely from what is called Run to Run control. More details about the industrial problem can be found in [10].

Several solution methods has been defined for PTC [7, 10, 11]. In particular, the authors of [7] present two pre-existing models: a Mixed Integer Linear Program (MILP) and a Constraint Programming (CP) model. Furthermore, they define a new CP model taking advantage of advanced CP features to model machine disqualifications. However, the paper shows that when the priority objective is the flow time, the performance of the CP model can be improved.

The objective of this paper is to improve the performances of the CP model for the flow time objective. To do so, a relaxed version of PTC where qualification constraints are removed is considered. For this relaxation, the results of Mason and Anderson [9] are adapted to define an algorithm to optimally sequence jobs on one machine in polynomial time. This algorithm is then used to define several filtering algorithms for PTC.

Although, the main result of this paper concerns the filtering algorithms for PTC, there is also two more general results incident to this work. First, those algorithms can be directly applied to any problem having a flow time objective and which can be relaxed to a parallel machine scheduling problem with sequence-independent family setup times. Secondly, the approach is related to cost-based domain filtering [3], a general approach to define filtering algorithms for optimization problems.

The paper is organized as follows. Section 2 gives a formal description of the problem the CP model for PTC. Section 3 presents the relaxed problem and the optimal machine flow time computation of the relaxed problem. Section 4 shows how this flow time is used to define filtering rules and algorithms for PTC. Finally, Section 5 shows the performance of the filtering algorithms and compares our results to the literature.

2 Problem description and modeling

In this section, a formal description of PTC is given. Then, a part of the CP model of [7] is presented. The part of the model presented is the part that is useful to present our cost based filtering rules and correspond to the modeling of the relaxation. Indeed, as we are interested in the flow time objective, the machine qualification modeling is not presented in this paper.

2.1 PTC description

Formally, the problem takes as input a set of jobs, $\mathcal{N} = \{1, \dots, N\}$, a set of families $\mathcal{F} = \{1, \dots, F\}$ and a set of machines, $\mathcal{M} = \{1, \dots, M\}$. Each job j belongs to a family and the family associated with j is denoted by $f(j)$. For each family f, only a subset of the machines $\mathcal{M}_f \subseteq \mathcal{M}$, is able to process a job of f. A machine m is said to be qualified to process a family f if $m \in \mathcal{M}_f$. Each family f is associated with the following parameters:

- n_f denotes the number of jobs in the family. Note that $\sum_{f \in \mathcal{F}} n_f = N$.
- p_f corresponds to the processing time of jobs in f.
- s_f is the setup time required to switch the production from a job belonging to a family $f' \neq f$ to the execution of a job of f. Note that this setup time is independent of f', so it is sequence -independent. In addition, no setup time is required neither between the execution of two jobs of the same family nor at the beginning of the schedule, i.e. at time 0.
- γ_f is the threshold value for the time interval between the execution of two jobs of f on the same machine. Note that this time interval is computed on a start-to-start basis, i.e. the threshold is counted from the start of a job of family f to the start of the next job of f on machine m. Then, if there is a time interval $]t, t + \gamma_f]$ without any job of f on a machine, the machine lose its qualification for f.

The objective is to minimize both the sum of job completion times, i.e. the flow time, and the number of qualification looses or disqualifications. An example of PTC together with two feasible solutions is now presented.

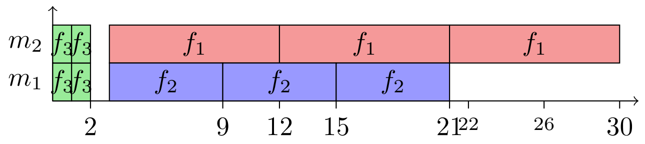
Example 1. *Consider the instance with* $N = 10$, $M = 2$ *and* $F = 3$ *given in Table 1(a). Figure 1 shows two feasible solutions. The first solution, described by Figure 1(b), is optimal in terms of flow time. For this solution, the flow time is equal to* $1 + 2 + 9 + 15 + 21 + 1 + 2 + 12 + 21 + 30 = 114$ *and the number of qualification losses is* 3. *Indeed, machine* 1 (m_1) *loses its qualification for* f_3 *at time* 22 *since there is no job of* f_3 *starting in interval* $]1, 22]$ *which is of size* $\gamma_3 = 21$. *The same goes for* m_2 *and* f_3 *at time* 22 *and for* m_2 *and* f_2 *at time* 26.
The second solution, described by Figure 1(c), is optimal in terms of number of disqualifications. Indeed, in this solution, none of the machines loses their qualifications. However, the flow time is equal to $1 + 2 + 9 + 17 + 19 + 9 + 18 + 20 + 27 + 37 = 159$.

2.2 CP model

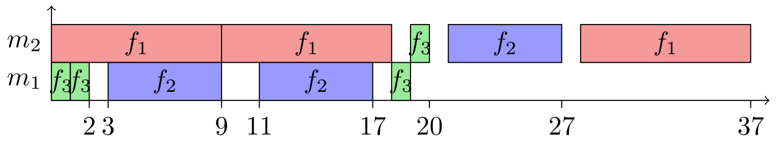
In the following section, the part of the CP model of [7] which is useful for this work is recalled. This part corresponds to a Parallel Machine Scheduling Problem (PMSP) with family setup times. New auxiliary variables used by our cost based filtering rules are also introduced. These variables are written in

f	n_f	p_f	s_f	γ_f	$\mathcal{M}_f$
1	3	9	1	25	$\{2\}$
2	3	6	1	26	$\{1,2\}$
3	4	1	1	21	$\{1,2\}$

(a) Instance with $N = 10$, $M = 2$ and $F = 3$



(b) An optimal solution for the flow time objective



(c) An optimal solution for qualification losses

Figure 1: Two solution examples for PTC.

bold in the variable description. To model the PMSP with family setup times, (optional) interval variables are used [5, 6]. To each interval variable J, a start time $st(J)$, an end time $et(J)$, a duration $d(J)$ and an execution status $x(J)$ is associated. The execution status $x(J)$ is equal to 1 if and only if J is present in the solution and 0 otherwise.

The following set of variables is used:

- $jobs_j$, $\forall j \in \mathcal{N}$: Interval variable modeling the execution of job j;
- $altJ_{j,m}$, $\forall (j,m) \in \mathcal{N} \times \mathcal{M}_{f(j)}$: Optional interval variable modeling the assignment of job j to machine m;
- $\mathbf{flowtime_m}$ and $flowtime$: Integer variables modeling respectively the flow time on machine m and the global flow time;
- $\mathbf{nbJobs_{f,m}}, \forall (f,m) \in \mathcal{F} \times \mathcal{M}_f$: Integer variable modeling the number of jobs of family f scheduled on m;
- $\mathbf{nbJobs_m}, \forall m \in \mathcal{M}$: Integer variable modeling the number of jobs scheduled on m.

To model the PMSP with setup time, the following sets of constraints is used:

$$flowTime = \sum_{j \in \mathcal{N}} et(jobs_j) \tag{1}$$

$$alternative\left(jobs_j, \left\{altJ_{j,m} | m \in \mathcal{M}_{f(j)}\right\}\right) \qquad \forall j \in \mathcal{N} \tag{2}$$

$$noOverlap\left(\left\{altJ_{j,m} | \forall j \text{ s.t. } m \in \mathcal{M}_{f(j)}\right\}, S\right) \qquad \forall m \in \mathcal{M} \tag{3}$$

$$flowtime = \sum_{m \in \mathcal{M}} flowtime_m \tag{4}$$

$$flowtime_m = \sum_{j \in \mathcal{N}} et(altJ_{j,m}) \qquad \forall m \in \mathcal{M} \qquad (5)$$

$$nbJobs_{f,m} = \sum_{j \in \mathcal{N}; f(j)=f} x(altJ_{j,m}) \qquad \forall (f,m) \in \mathcal{F} \times \mathcal{M}_f \qquad (6)$$

$$\sum_{m \in \mathcal{M}} nbJobs_{f,m} = n_f \qquad \forall f \in \mathcal{F} \qquad (7)$$

$$\sum_{f \in \mathcal{F}} nbJobs_{f,m} = nbJobs_m \qquad \forall m \in \mathcal{M} \qquad (8)$$

$$\sum_{m \in \mathcal{M}} nbJobs_m = N \qquad (9)$$

Constraint (1) is used to compute the flow time of the schedule. Constraints (2)–(3) are used to model the PMSP with family setup time. Constraints (2) model the assignment of jobs to machine. Constraints (3) ensure that jobs do not overlap and enforce setup times. Note that S denotes the setup time matrix: $(S_{f,f'})$ is equal to 0 if $f = f'$ and to s_f otherwise. A complete description of *alternative* and *noOverlap* constraints can be found in [5, 6].
In [7], additional constraints are used to make the model stronger, e.g. ordering constraints, cumulative relaxation. They are not presented in this paper.
Constraints (4)–(9) are used to link the new variables to the model. Constraints (4) and (5) ensure machine flow time computation. Constraints (6) compute the number of jobs of family f executed on machine m. Constraints (7) make sure the right number of jobs of family f is executed. Constraints (8) and (9) are equivalent to constraints (6) and (7) but for the total number of jobs scheduled on machine m. The bi-objective optimization is a lexicographical optimization or its linearization [2].

3 Relaxation description and sequencing

3.1 $\mathcal{R}$-PTC description

The relaxation of PTC ($\mathcal{R}$-PTC) is a parallel machine scheduling problem with sequence-independent family setup time without the qualification constraints (parameter γ_f). The objective is then to minimize the flow time. In this section, it is assumed that a total the assignment of jobs to machines is already done and the objective is to sequence jobs so the flow time is minimal. Therefore, since the sequencing of jobs on M machines can be seen as M one machine problems, this section presents how jobs can be sequenced optimally on one machine. In Section 4, the cost-based filtering rules handle partial assignments of jobs to the machines.

3.2 Optimal sequencing for $\mathcal{R}$-PTC

The results presented in this section were first described in [8]. They are adapted from Mason and Anderson [9] who considers an initial setup at the beginning of the schedule. The results are just summarized in this paper.

First, a solution can be represented as a sequence S representing an ordered set of n jobs. Considering job families instead of individual jobs, S can be seen as a series of blocks, where a block is a maximal consecutive sub-sequence of jobs in S from the same family (see Figure 2). Let B_i be the i-th block of the sequence, $S = \{B_1, B_2, \ldots, B_r\}$. Hence, successive blocks contain jobs from different families. Therefore, there will be a setup time before each block (except the first one).

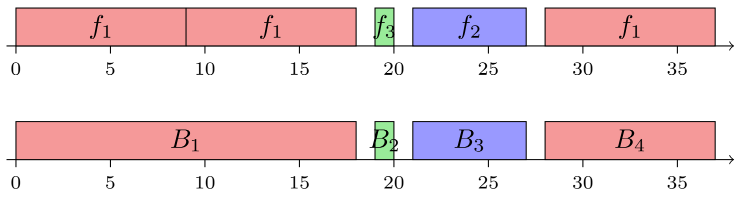


Figure 2: Block representation of a solution.

The idea of the algorithm is to adapt the Shortest Processing Time (SPT) rule [12] for blocks instead of individual jobs. To this end, blocks are considered as individual jobs with processing time $P_i = s_{f_i} + |B_i| \cdot p_{f_i}$ and weight $W_i = |B_i|$ where f_i denotes the family of jobs in B_i (which is the same for all jobs in B_i).

The first theorem of this section states that there always exists an optimal solution S containing exactly $|\mathcal{F}|$ blocks and that each block B_i contains all jobs of the family f_i. That is, all jobs of a family are scheduled consecutively.

Theorem 1. *Let $\mathcal{I}$ be an instance of the problem. There exists an optimal solution $S^* = \{B_1, \ldots, B_{|\mathcal{F}|}\}$ such that $|B_i| = n_{f_i}$ where f_i is the family of jobs in B_i.*

Sketch. For a complete proof of the theorem, see [8].
Consider an optimal solution $S = \{B_1, \ldots, B_u, \ldots, B_v, \ldots, B_r\}$ with two blocks B_u and B_v $(u < v)$, containing jobs of the same family $f_u = f_v = f$. Then, moving the first job of B_v at the end of block B_u can only improve the solution.

Indeed, let us define P and W as: $P = \sum_{i=u+1}^{v-1} P_i + s_f$ and $W = \sum_{i=u+1}^{v-1} |B_i|$. In addition, let S' be the sequence formed by moving the first job of B_v, say job j_v, at the end of block B_u. The difference on the flow time between S and S', is as follows:

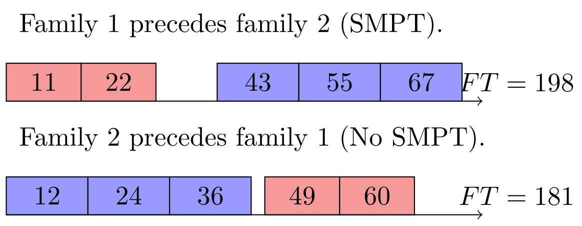
$$FT_{S'} - FT_S = \begin{cases} W \cdot p_f - P & \text{if } |B_v| = 1 \\ W \cdot p_f - P - \sum_{i=v+1}^{r} |B_i| \cdot s_f & \text{if } |B_v| > 1 \end{cases}$$

Using Lemma 1 of [8] stating that $P/W \geq p_f$, then $FT_{S'} - FT_S < 0$ and the flow time is improved in S'. Hence, moving the first job of B_v at the end of block B_u leads to a solution S' at least as good as S.

Therefore, a block B_i contains all jobs of family f_i. Indeed, if not, applying the previous operation leads to a better solution. Hence, $|B_i| = n_{f_i}$ and there are exactly F blocks in the optimal solution, i.e. one block per family. □

f	n_f	p_f	s_f	MPT_f
1	2	11	2	12
2	3	12	9	15

(a) Instance with $N = 5$ and $F = 2$.


Family 1 precedes family 2 (SMPT).
11
22
43
55
67
$FT = 198$
Family 2 precedes family 1 (No SMPT).
12
24
36
49
60
$FT = 181$

(b) Scheduling jobs of a family before those of the other one. Numbers are completion times of the jobs.

Figure 3: Scheduling all blocks according to the SMPT rule is not optimal.

At this point, the number of block and theirs contents are defined. The next step is to order them. To this end, the concept of weighted processing time is also adapted to blocks as follows.

Definition 1. *The Mean Processing Time (MPT) of a block B_i is defined as $MPT(B_i) = P_i/W_i$.*

One may think that, in an optimal solution, jobs are ordered by $SMPT$ (Shortest Mean Processing Time). However, this is not always true since no setup time is required at time 0. Indeed, the definition of block processing time always considers a setup time before the block. In our case, this is not true for the first block. Example 2 gives a counter-example showing that scheduling all blocks according to the SMPT rule is not optimal.

Example 2 (Counter-example – Figure 3)**.** *Consider the instance given by Table 3(a) that also gives the MPT of each family ($MPT_f = p_f + (s_f \div n_f)$). Figure 3(b) shows the SMPT rule may lead to sub-optimal solutions when no setup time is required at time 0. Indeed, following the SMPT rule, jobs of family 1 have to be scheduled before jobs of family 2. This leads to a flow time of 198. However, schedule jobs of family 2 before jobs of family 1 leads to a better flow time, i.e. 181.*

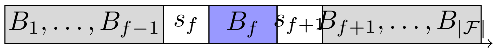
Actually, the only reason why the $SMPT$ rule does not lead to an optimal solution is that no setup time is required at time 0. Therefore, in an optimal solution, all blocks except the first one are scheduled according to the $SMPT$ rule. That is the statement of Theorem 2 for which a proof is given in [8].

Theorem 2. *In an optimal sequence of the problem, the blocks 2 to $|\mathcal{F}|$ are ordered by $SMPT$ (Shortest Mean Processing Time). That is, if $1 < i < j$ then $MPT(B_i) \leq MPT(B_j)$.*

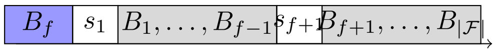
The remaining of this section explains how these results are used to define a polynomial time algorithm for sequencing jobs on a machine so the flow time is minimized. This algorithm is called `sequencing` in the remaining of the paper.

Theorem 1 states that there exists an optimal solution S containing exactly $|\mathcal{F}|$ blocks and that each block B_i contains all jobs of family f_i. Theorem 2 states that the blocks B_2 to $B_{|\mathcal{F}|}$ are ordered by $SMPT$. Finally, one only needs to determine which family is processed in the very first block.

Algorithm `sequencing` takes as input the set of jobs and starts by grouping them in blocks and sorting them in SMPT order. The algorithm then computes the flow time of this schedule. Each block is then successively moved to the first position (see Figure 4) and the new flow time is computed. The solution returned by the algorithm is therefore the one achieving the best flow time.



(a) SMPT Sequence



(b) B_f is moved in the first position.

Figure 4: SMPT Sequence and Move Operation.

The complexity for ordering the families in $SMPT$ order is $O(F \log F)$. The complexity of moving each block to the first position and computing the corresponding flow time is $O(F)$. Indeed, there is no need to re-compute the entire flow time. The difference of flow time coming from the *Move operation* can be computed in $O(1)$. Hence, the complexity of `sequencing` is $O(F \log F)$.

4 Filtering rules and algorithms

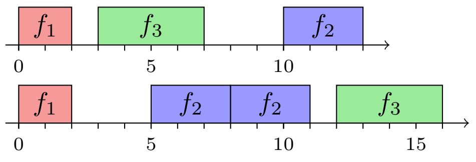
This section is dedicated to the cost-based filtering rules and algorithms derived from the results of Section 3. They are separated into three parts, each one corresponding to the filtering of one variable: $flowtime_m$ (Section 4.1); $nbJobs_{f,m}$ (Section 4.2); $nbJobs_m$ (Section 4.3). Note that the two last variables constrained by the sum constraint 8.

During the solving of the problem, jobs are divided for each machine m into three categories based on the interval variables $altJ_{j,m}$: each job either is, or can be, or is not, assigned to the machine. Note that the time windows of the interval variables are not considered in the relaxation. For a machine m, let $\mathcal{N}_m^A$ be the set of jobs for which the assignment to machine m is decided. In the following of this section, an instance $\mathcal{I}$ is always considered with the set $\mathcal{N}^A = \cup_{m \in \mathcal{M}} \mathcal{N}_m^A$ of jobs already assigned to a machine. Thus, an instance is denoted by $(\mathcal{I}, \mathcal{N}^A)$.

Some notations are now introduced. For a variable x, $\underline{x}$ (resp. $\overline{x}$) represents the lower (resp. the upper) bound on the variable x. Furthermore, let $FT^*(\mathcal{X})$ be the flow time of the solution returned by the algorithm `sequencing` applied on the set of jobs $\mathcal{X}$.

f	n_f	p_f	s_f
1	3	2	5
2	3	3	3
3	4	4	1

(a) Instance data.



(b) Illustration of Rule 1 and Rule 3.

Figure 5: Illustration of $flowtime_m$ filtering (Rule 1).

4.1 Increasing the machine flow time

The first rule updates the lower bound on the $flowtime_m$ variable and follows directly from Section 3. The complexity of this rule is $O(M \cdot F \cdot \log F)$.

Rule 1. $\forall m \in \mathcal{M},\ flowtime_m \geq FT^*(\mathcal{N}_m^A)$

Proof. It is sufficient to notice that, for a machine m, `sequencing` gives a lower bound on the flow time. In particular, $FT^*(\mathcal{N}_m^A)$ is a lower bound on the flow time of m for the instance $(\mathcal{I}, \mathcal{N}^A)$. □

Example 3. *Consider an instance with* 3 *families. Their parameters are given by Table 5(a). A specific machine m is considered and set $\mathcal{N}_m^A$ is composed of one job of each family. This instance and $\mathcal{N}_m^A$ is used in all the example of this section.*
Suppose $flowtime_m \in [0, 35]$. The output of `sequencing` *is given on the top of Figure 5(b). Thus, the lower bound on the flow time can be updated to* $2+7+13 = 22$*, i.e. $flowtime_m \in [22, 35]$.*
Suppose now that an extra job of family f_2 is added to $\mathcal{N}_m^A$ (on the bottom of Figure 5(b)). Thus, $FT^*(\mathcal{N}_m^A) = 2+8+11+16 = 37$ and $37 > \overline{flowtime}_m = 35$. Thus, the assignment defined by $\mathcal{N}_m^A$ is infeasible.

Another rule can be defined to filter $flowtime_m$. This rule is stronger than Rule 1 and is based on $\mathcal{N}_m^A$ and $\underline{nbJobs}_m$. Indeed, $\underline{nbJobs}_m$ denotes the minimum number of jobs that has to be assigned to m. Thus, if one can find the $\underline{nbJobs}_m$ jobs (including the one of $\mathcal{N}_m^A$) that leads to the minimum flow time, it will give a lower bound on the flow time of machine m.
Actually, it may be difficult to know exactly which jobs will contribute the least to the flow time. However, considering jobs in SPT order and with 0 setup time gives a valid lower bound on $flowtime_m$. First, an example illustrating the filtering rule is presented and then, the rule is formally given.

Example 4. *Consider the instance described in Example 3. Suppose $\mathcal{N}_m^A$ is composed of one job of each family and $flowtime_m \in [22, 60]$. Suppose also $\underline{nbJobs}_m = 6$. Thus,* 3 *extra jobs need to be assigned to m.*
Families in SPT order are $\{f_1, f_2, f_3\}$ and the remaining number of jobs in each family is $2, 2, 3$*. Hence, the* 3 *extra jobs are:* 2 *jobs of f_1 and* 1 *job of f_2.*

To make sure the lower bound on the flow time is valid, those jobs are sequenced on m with no setup time. In Figure 6, f_j denotes "classical" jobs of family f_j while f'_j denotes jobs of family f_j with no setup time.
Figure 6 shows the results of `sequencing` *on the set of jobs composed of $\mathcal{N}_m^A$ plus the 3 extra jobs, i.e. $\underline{nbJobs}_m = 6$. Here, $FT^* = 2{+}4{+}6{+}9{+}14{+}20 = 55$. Thus, the lower bound on $flowtime_m$ can be updated and $flowtime_m \in [55, 60]$.*

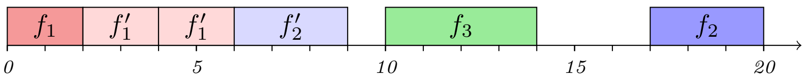


Figure 6: Illustration of $flowtime_m$ filtering (Rule 2).

Note that, because Rule 1 does not take $\underline{nbJobs}_m$ into account, it gives a lower bound of 22 in this case.

Let $\mathcal{N}_m^O$ denotes the set composed of the first $\underline{nbJobs}_m - |\mathcal{N}_m^A|$ remaining jobs in SPT order with setup time equal to 0.

Rule 2. $\forall m \in \mathcal{M},\ flowtime_m \geq FT^*(\mathcal{N}_m^A \cup \mathcal{N}_m^O)$

Proof. First note that if $\mathcal{N}_m^O = \emptyset$, Rule 1 gives the result. Thus, suppose that $|\mathcal{N}_m^O| \geq 1$. By contradiction, suppose $\exists m \in \mathcal{M}$ such that $flowtime_m < FT^*(\mathcal{N}_m^A \cup \mathcal{N}_m^O)$. Thus, there exists another set of jobs $\overline{N_m^O}$ with:

$$FT^*(\mathcal{N}_m^A \cup \overline{N_m^O}) < FT^*(\mathcal{N}_m^A \cup \mathcal{N}_m^O) \tag{10}$$

First, note that w.l.o.g. $|\overline{N_m^O}| = |\mathcal{N}_m^O|$. Indeed, if $|\overline{N_m^O}| > |\mathcal{N}_m^O|$, we can remove jobs from $|\overline{N_m^O}|$ without increasing the flow time. Furthermore, w.l.o.g. we can consider that $\forall j \in \overline{N_m^O},\ s_{f_j} = 0$. Indeed, since setup times can only increase the flow time, thus inequality (10) is still verified.
Let $\overline{S} = \{\overline{j}_1, \cdots, \overline{j}_{nbJobs_m}\}$ be the sequence returned by the `sequencing` algorithm on $\mathcal{N}_m^A \cup \overline{N_m^O}$. Let also $\overline{j}_i$ be the job in $\overline{N_m^O} \setminus \mathcal{N}_m^O$ with the SPT. Finally, let j_k be the job with the SPT in $\in \mathcal{N}_m^O \setminus \overline{N_m^O}$. Thus, since $p_{f_{\overline{j}_i}} > p_{f_{j_k}}$, sequence $\overline{S}' = \{\overline{j}_1, \cdots, \overline{j}_{i-1}, j_k, \overline{j}_{i+1}, \cdots \overline{j}_{nbJobs_m}\}$ has a smaller flow time than $\overline{S}$.
Repeated applications of this operation yield to a contradiction with equation (10). □

The complexity of Rule 2 is $O(M \cdot F \cdot \log F)$. Indeed, sorting families in SPT order can be done in $O(F \cdot logF)$. Creating the set $\mathcal{N}_m^O$ is done in $O(F)$ and `sequencing` is applied in $O(F \cdot logF)$ which gives a total complexity of $O(F \cdot logF + M \cdot (F + F \cdot \log F))$.

4.2 Reducing the maximum number of jobs of a family

The idea of the filtering rule presented in this section is as follows. For a family f, $\overline{nbJobs}_{f,m}$ define the maximum number of jobs of family f that can be scheduled on m. Thus, if adding those $\overline{nbJobs}_{f,m}$ to $\mathcal{N}_m^A$ leads to an infeasibility, $\overline{nbJobs}_{f,m}$ can be decreased by at least 1. Let denote by $\mathcal{N}_m^f$ the set composed of $\overline{nbJobs}_{f,m}$ jobs of family f minus those already present in $\mathcal{N}_m^A$.

Rule 3. *If $\exists(f,m) \in \mathcal{F} \times \mathcal{M}$ such that $FT^*(\mathcal{N}_m^A \cup \mathcal{N}_m^f) > \overline{flowtime}_m$, then $nbJobs_{f,m} \leq \overline{nbJobs}_{f,m} - 1$*

Proof. Suppose that for a family f and a machine m, we have a valid assignment such that $FT^*(\mathcal{N}_m^A \cup \mathcal{N}_m^f) > \overline{flowtime}_m$ and $nbJobs_{f,m} = \overline{nbJobs}_{f,m}$. By Rule 1, the assignment is infeasible which is a contradiction. □

Example 5. *Let us consider the instance defined by example 3. In the first part of this example, $\mathcal{N}_m^A$ is composed of one job of each family and $flowtime_m \in [22, 35]$. Suppose that $\overline{nbJobs}_{f_2,m} = 2$. Thus, $\mathcal{N}_m^A \cup \mathcal{N}_m^f$ contains one job of family f_1 and f_3 and two jobs of family f_2. The bottom part of figure 5(b) shows that $FT^*(\mathcal{N}_m^A \cup \mathcal{N}_m^f) = 37 > \overline{flowtime}_m = 35$. Thus, $\overline{nbJobs}_{f_2,m} < 2$.*

The complexity of Rule 3 is $O(M \cdot F \cdot \log F + M \cdot F^2 \cdot \log |\mathcal{D}_{nbJobs_{f,m}}^{MAX}|)$ with $|\mathcal{D}_{nbJobs_{f,m}}^{MAX}|$ the maximum size of the domain of variables $nbJobs_{f,m}$. Indeed, the first part corresponds to the complexity for applying the `sequencing` algorithm on all machine. This algorithm only need to be applied once since each time we remove jobs from $\mathcal{N}_m^f$, FT^* can be updated in $O(F)$. Indeed, only the position of the family in the sequence has to be updated. Thus, the second part corresponds to the updating of FT^* for each family and each machine. By proceeding by dichotomy, this update has to be done at most $\log |\mathcal{D}_{nbJobs_{f,m}}^{MAX}|$ times. Thus, the complexity of Rule 3 is $O(M \cdot F \cdot (\log F + F \cdot \log |\mathcal{D}_{nbJobs_{f,m}}^{MAX}|))$.

4.3 Reducing the maximum number of jobs on a machine

The idea behind Rule 2 can be used to reduce the maximum number of jobs on machine m. Indeed, for a machine m, $\overline{nbJobs}_m$ is the maximum number of jobs that can be scheduled to m. Thus, if it is not possible to schedule $\overline{nbJobs}_m$ on m without exceeding the flow time, then $\overline{nbJobs}_m$ can be decreased.

The extra jobs that will be assigned on m must be decided. Note that those jobs must give a lower bound on the flow time for $\overline{nbJobs}_m$ jobs with the pre-assignment defined by $\mathcal{N}_m^A$. Thus, jobs can be considered in SPT order with no setup time. Before giving the exact filtering rule, an example is described.

Example 6. *Consider the instance described in Example 3. In the first part of this example, $\mathcal{N}_m^A$ is composed of one job of each family and $flowtime_m \in [22, 60]$. Suppose $\overline{nbJobs}_m = 7$. Thus, the 4 extra jobs assigned to m are: 2 jobs of f_1 and 2 jobs of f_2. Figure 7 shows the results of `sequencing` on the set of jobs composed of $\mathcal{N}_m^A$ plus the 4 extra jobs. Here, $FT^* = 2+4+6+11+14+17+23 =$*

77 which is greater than $\overline{flowtime}_m = 60$*. Thus,* $\overline{nbJobs}_m$ *cannot be equal to* 7 *and can be filtered.*

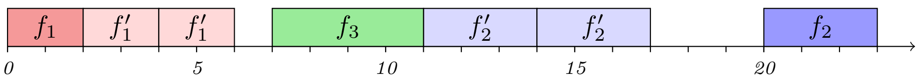


Figure 7: Illustration of Rule 4.

Let $\mathcal{N}_m^{NS}$ denotes the set composed of the first $\overline{nbJobs}_m - |\mathcal{N}_m^A|$ jobs in SPT order with setup time equal to 0.

Rule 4. *If* $\exists m \in \mathcal{M}$ *such that* $FT^*(\mathcal{N}_m^A \cup \mathcal{N}_m^{NS}) > \overline{flowtime}_m$*, then* $nbJobs_m \leq \overline{nbJobs}_m - 1$

Proof. The arguments are similar to those for Rule 2. □

The complexity of Rule 4 is $O(M \cdot F \cdot \log F + M \cdot F^2 \cdot |\mathcal{D}_{nbJobs_m}^{MAX}|)$ with $|\mathcal{D}_{nbJobs_m}^{MAX}|$ the maximum size of the domain of variables $nbJobs_m$. Indeed, as for Rule 3, the algorithm `sequencing` only needs to be applied once and then can be updated in $O(F)$. For each machine and each family, this update has to be done at most $|\mathcal{D}_{nbJobs_m}^{MAX}|$ times. Indeed, proceeding by dichotomy here implies that FT^* cannot be updated but has to be re-computed each time. Thus, the complexity of Rule 4 is $O(M \cdot F \cdot (\log F + F \cdot |\mathcal{D}_{nbJobs_m}^{MAX}|))$.

5 Experimental Results

This section starts with the presentation of the general framework of the experiments in 5.1. Following the framework, the filtering rules are evaluated in Section 5.2. Then, the model is compared to those of the literature in Section 5.3. Last, a brief sensitivity analysis is given in Section 5.4.

5.1 Framework

The experiment framework is defined so the following questions are addressed.
Q1. Which filtering rule is efficient? Are filtering rules complementary?
Q2. Which model of the literature is the most efficient?
Q3. What is the impact of the heuristics? Of the bi-objective aggregation?
To address these questions, the following metrics are analyzed: number of feasible solutions, number of proven optimums, upper bound quality; solving times; number of fails (for CP only).

The benchmark instances used to perform our experiments are extracted from [10]. In this paper, 19 instance sets are generated with different number of jobs (N), machines (M), family (F) and qualification schemes. Each of the instance sets is a group of 30 instances. There is a total of 570 feasible instances with $N = 20$ (180), $N = 30$ (180), $N = 40$ (30), $N = 50$ (30), $N = 60$ (60), $N = 70$ (90).

The naming scheme for the different solving algorithms is described in Table 1. The first letter represents the model where ILP model, CP_O model and CP_N model denotes respectively the ILP model and CP model of [7], and the CP model detailed in Section 2.2. The models are implemented using IBM ILOG CPLEX Optimization Studio 12.10 [4]. That is CPLEX for the ILP model and CP Optimizer for CP models. The second letter indicates whether two heuristics are executed to find solutions which are used as a basis for the models. These heuristics are called *Scheduling Centric Heuristic* and *Qualification Centric Heuristic* [10]. The goal of the first heuristic is to minimize the flow time while the second one tries to minimize the number of disqualifications. The third letter indicates the filtering rules that are activated for the CP_N model. Rule 2 is used for the letter `L` because it has been shown more efficient than Rule 1 in preliminary experiments. The fourth letter indicates the bi-objective aggregation method: lexicographic optimization; linearization of lexicographic optimization. The last letter indicates the objective priority. Here, the priority is given to the flow time in all experiments because the cost-based filtering rules concern the flow time objective.

Model		Heuristic		Filtering rule				Bi-objective		Priority	
`I`	ILP model	`_`	None	`L`	Rule 2	`_`	None	`S`	Weighted sum	`F`	Flow time
`O`	CP_O model	`H`	All	`F`	Rule 3	`A`	All	`L`	Lexicographic	`Q`	Disqualifications
`N`	CP_N model			`M`	Rule 4						

Table 1: Algorithms Encoding.

All the experiments were led on a Dell computer with 256 GB of RAM and 4 Intel E7-4870 2.40 GHz processors running CentOS Linux release 7 (each processor has 10 cores). The time limit for each run is 300 seconds.

5.2 Evaluation of the filtering rules

In this section, the efficiency and complementary nature of the proposed filtering rules are investigated. In other words, the algorithms `N_*LF` are studied. To this end, the heuristics are not used to initialize the solvers. The lexicographic optimization is used since it has been shown more efficient in preliminary experiments.

First, all algorithms find feasible solutions for more than 99% of the instances. Then, for each algorithm, the number of instances solved optimally is drawn as a function of the time in Figure 8(a). The leftmost curve is the fastest whereas the topmost curve proves the more optima. Clearly, compared to `N__LF`, the filtering rules accelerates the proof and allow the optimal solving of around eighty more instances. One can notice that the advanced filtering rules (`N_FLF`, `N_MLF`, `N_ALF`), also slightly improves the proof compared to the simple