RAIRO-Oper. Res. 53 (2019) 593–608
https://doi.org/10.1051/ro/2017073

RAIRO Operations Research
www.rairo-ro.org

EVALUATION OF A NEW DECISION-AID PARAMETER FOR JOB SHOP SCHEDULING UNDER UNCERTAINTIES

Zakaria Yahouni[1,*], Nasser Mebarki[1] and Zaki Sari[2]

Abstract. This paper addresses the groups of permutable operations method. This method is a flexible scheduling approach to hedge against uncertainties and is composed of a proactive/reactive phase. The proactive phase consists of computing a set of solutions (schedule) to a scheduling problem, leaving the choice of executing one of these solutions during the reactive phase according to the current state of the shop floor. During the reactive phase, the remaining decisions have to be made in real-time. The *worst-case* evaluation of the remaining solutions is a decision-aid parameter used during the reactive phase in order to control the final schedule from exceeding a *worst-case* performance. While the existing literature only tackles the *worst-case* evaluation of the groups of permutable operations, this paper deals with its *best-case* evaluation. For solving this problem, a new lower bound calculating this parameter in polynomial time is proposed. The computational efficiency of this parameter in a reactive algorithm exhibits very good performance. Moreover, the experiments show the robustness of this evaluation allowing this parameter to be used in an unstable job shop environment.

Mathematics Subject Classification. 90B35, 90B36, 68M20.

Received July 20, 2016. Accepted October 5, 2017.

1. Introduction

Most of the classical scheduling studies assume that all data and parameters of the problem are fully known. However, in practice, manufacturing systems are not so deterministic. During the execution of the offline generated schedule, many disturbances may occur, like arrival of new jobs, uncertain task duration, machine breakdown, *etc.*, [9,20,31]. These disturbances will in most cases, deteriorate the expected performances.

This issue has received considerable critical attention over the past years, [5,10,26,27] are among who first considered this problem. To address this issue, different approaches have been proposed in the literature. These approaches have been classified by [20,30,43] into three main classes: proactive, reactive and proactive-reactive approaches.

The proactive approaches rely generally on estimating and anticipating the perturbations before executing a schedule [23]. One of the most known proactive methods in project scheduling is the classical critical chain method used in several project management pieces of software [28]. This method introduces slacks on the operations of the critical path to absorb the delays caused by the perturbations. Another proactive method

Keywords. Scheduling, decision-aid system, lower bounds, Job shop, makespan.

[1] LS2N, Laboratoire des Sciences du Numérique de Nantes, UMR CNRS 6004, Nantes, France.

[2] MELT, Manufacturing Engineering Laboratory of Tlemcen, Algeria.

*Corresponding author: zakaria.yahouni@gmail.com

is *just-in-case* scheduling [45]. This method identifies the activities most likely to fail based on the available uncertainty information. The weakness of this kind of approaches is that it deals with a prediction of the system instead of the current state of the shop.

The second class encompasses the reactive approaches. Over the past years, much effort has been devoted to developing reactive methods that use generally a real-time control heuristics [19, 35, 46, 47]. These real-time control methods build dynamically the schedule based on the current state of the system [18]. Although, these reactive methods consider the current state of the shop floor contrarily to the proactive methods, their performance is good on a very high disturbed environment and generally poor on a low disturbed environment [13].

The third class combines the proactive and reactive approaches. The work of [43] addresses the relative importance of such combination to hedge against uncertainties. The basic idea of these approaches is to develop during the offline phase of the scheduling process, a flexible schedule with some degree of freedom, allowing during the online reactive phase (execution phase) to revise and adapt the schedule according to the new state of the shop.

The right choice of the scheduling approach depends on the scheduling environment and some assumptions that are taken into account when analyzing manufacturing systems [49]. The proactive-reactive approaches seem to be the most effective approaches to absorb production uncertainties while maintaining a good performance [48]. These approaches focus on the proper combination between a proactive flexible method and a reactive algorithm capable of revising the flexible schedule without deteriorating too much the expected performances. Some of these methods can be found in [8, 25, 32, 37, 40, 48]. *Groups of Permutable Operations* (GoPO) is one of the most studied proactive-reactive methods [2, 3, 9, 13, 22]. This method was created by LAAS-CNRS laboratory of Toulouse, France [21] and is implemented in an industrial software [41]. It is composed of two phases:

- A proactive phase which aims at computing a flexible solution offline. This solution is represented by groups of permutable operations such that each permutation between two operations gives a feasible schedule.
- A reactive phase in which the proactive schedule is processed online on the shop floor. This phase consists of choosing during the execution of a GoPO solution, the order of operations to be executed in each group of permutable operations. The decision of ordering the operations in each group can be chosen either using a reactive algorithm or by a human, named the operator. The schedule executed during this phase is called *realized schedule*.

This flexible method has an interesting property; once the groups are generated during the proactive phase, the worst schedule called the *worst-case* (which represents the worst permutation in each group), can be computed in polynomial time [1, 2, 22]. This allows to guarantee a minimal quality of the *realized schedule* (after taking all the decisions). Due to this property, the *worst-case* can be used during the reactive phase of GoPO in a decision-aid algorithm/system in order to schedule the operations inside each group according to the current state of the shop. The literature review has primarily focused on this parameter. However, making a decision during the reactive phase of GoPO using only the *worst-case* parameter can lead prioritizing lower quality solutions.

In this paper, we raised this issue by proposing a new parameter that can be used for the reactive phase of GoPO. This parameter consists of evaluating the *best-case* performance which corresponds to the best possible permutation in each group, such that the final *realized schedule* is an optimal solution of GoPO. The literature research has not focused on this parameter (*best-case*) as its computation is NP-hard and can be too time-consuming to be used in real-time in a decision-aid system. But a lower bound of this parameter should be of good interest for different reasons:

- Associated with the *worst-case* value, it permits to better represent the quality of GoPO as denoted by [22]. This quality can be measured by a range of all possible performances [*worst-case* ... *best-case*].
- It also permits to know, once the groups are generated during the proactive phase, the best achievable performance in case of no disruptions. For example, if the scheduling objective is to minimize the tardiness,

it permits to know if there is no schedule that has no late job which can be of great interest for the decision maker.
- Most importantly, it permits to know, during the execution of a GoPO solution (reactive phase), the next optimal (or near-optimal) operation to be chosen in each group of permutable operations.

Two main contributions are presented in this work: the first one concerns the computation of the *best-case* of GoPO. As this problem is challenging, an efficient polynomial lower bound for the *best-case makespan* performance has been proposed. Three steps are used for the computation of this new lower bound: first, we adapt the literature findings to GoPO. Then, this lower bound is tightened using the groups precedence property and the last step is the improvement of this lower bound using a precedence constraint property between the operations. The second contribution of the work presented in this article concerns the study of the usefulness of the *best-case* parameter in a decision-aid system. For this, the *best-case* parameter has been implemented in a decision-aid reactive algorithm and experimented in both deterministic and non-deterministic environments. This experimental implementation of the *best-case* parameter exhibits good performance.

The rest of the article is structured as follows: in Section 2 and 3, GoPO is described and the problematic of the presented work is discussed. In Section 4, the lower bound for the *best-case* of GoPO is presented. Section 5 is devoted to the reactive phase of GoPO where a new reactive algorithm using the *best-case* parameter has been evaluated on well-known instances of the job shop problem. In the same section, the reactive decision-aid algorithm which uses the *best-case* parameter is studied on both deterministic and non-deterministic environments. Finally, main conclusions are summarized in the last section.

2. Groups of permutable operations

We consider the job shop problem with release dates and precedence constraints (J/r_i,Pred/f according to the classification of [29]) where we have n jobs $J_1, J_2, \ldots, J_n$ to be processed on m machines $M_1, M_2, \ldots, M_m$, each machine can treat only one operation at a time. Job i consists of n_i operations. Associated with every operation O_i: a machine allocation μ_i, a release date r_i, a processing time p_i and a completion time C_i. Γ_i^- and Γ_i^+ denote respectively the predecessor and the successor of a given operation. Generally, the job shop problem uses a regular objective function f that is a non-decreasing function of the C_i. In this work, we focus on the *makespan* objective $C_{\max} = \max(C_i)$.

GoPO is defined as a sequence of groups (of permutable operations) on each machine to be performed in a particular given order. The group containing operation O_i is denoted G_i. Every group contains one or many operations that can be executed in an arbitrary order. On a given machine, the group after (resp. before) the group containing the operation O_i is denoted by G_i^+ (resp. G_i^-). A GoPO schedule is feasible if for each group, all the permutations among all the operations of the same group give a feasible schedule (*i.e.* a schedule which satisfies all the constraints of the problem).

To illustrate this problem, let us study a job shop example of three jobs and three machines as described in Table 1. Figure 1 presents a feasible GoPO solving this problem. This GoPO is made of seven groups: two groups of two operations and five groups of one operation. The first machine is composed of two groups, the group G_6 containing O_6 and its predecessor group G_6^- containing the permutable operations O_1 and O_7. To generate these groups of permutable operations, a greedy algorithm (described in [22]) may be used for the construction of the groups. This algorithm starts with an *initial schedule* with one operation per each group (this initial schedule is usually generated using simple heuristics). Then, the algorithm tries to merge each two successive groups according to different criteria[3] until no group merging is possible.

In this paper, we focus on the execution phase of GoPO; we suppose that the groups are already given. The execution of a GoPO schedule can be viewed as a sequence of decisions: each decision consists in choosing

[3]Such that the groups are not related with a precedence constraint and the worst-case permutation in this group does not exceed a given threshold.

TABLE 1. Example of a job shop problem.

J_i	J_1			J_2			J_3		
O_i	O_1	O_2	O_3	O_4	O_5	O_6	O_7	O_8	O_9
μ_i	M_1	M_2	M_3	M_2	M_3	M_1	M_1	M_3	M_2
Γ_i^-	/	O_1	O_2	/	O_4	O_5	/	O_7	O_8
p_i	1	4	1	2	3	1	4	2	3

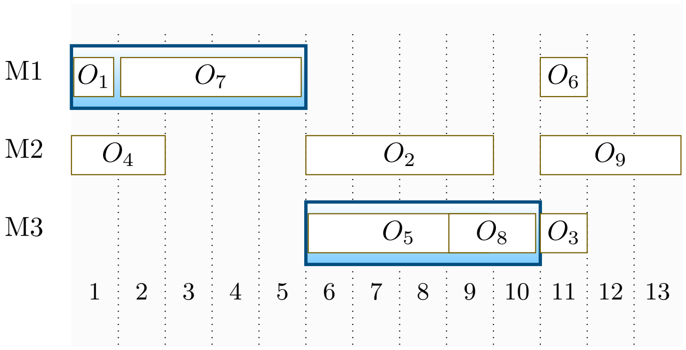


FIGURE 1. Groups of Permutable Operations solution.

an operation to execute in a group when this group is composed of more than one operation. The number of decisions that need to be taken is expressed as follows:

$$\sum_{\forall i}(|G_i|-1)$$

For instance, for the solution described on Figure 1 and Table 1, there are two decisions to be taken: on M_1, at the beginning of the scheduling, either operation O_1 or O_7 has to be executed. Let us suppose that the decision taken is to schedule O_1 before O_7. There is another decision to be taken on M_3: scheduling operation O_5 or O_8 first. Figure 2 represents all the different semi-active[4] schedules that can be obtained after taking these two decisions. Note that these schedules do not always have the same quality (performance); the best schedule has a $C_{\max} = 10$ as shown in Figure 2a and the worst schedule has a $C_{\max} = 12$ as shown in Figures 2c and 2d.

3. PROBLEMATIC

The main problematic of this paper is to find the best schedule (*best-case*) to be used as a decision-aid parameter during the decision phase of GoPO. This problem is computationally challenging as it is a combinatorial problem and the number of possible schedules that can be described by a GoPO solution is $\prod_{\forall i}((|G_i|)!)$.

The most effective and commonly operational research tool for solving optimally this problem is to use an exact method like the branch and bound algorithm. However, an exact method can be very costly and time-consuming in a decision-aid system, especially for solving large scale NP-hard combinatorial optimization problems. Instead, an effective lower bound can be computed in real-time to produce a near optimal (or optimal) anticipation of the best schedule that can be achieved.

To compute this lower bound and to avoid the exhaustive enumeration of all possible schedules, we can use a symmetrical calculation of the *worst-case* evaluation [2,3]. This new evaluation can be measured by computing the *best-case-earliest-starting-time* of operations and groups.

[4]A feasible schedule is called semi-active if no operation can be finishing earlier without changing the order of the operations on the machines.

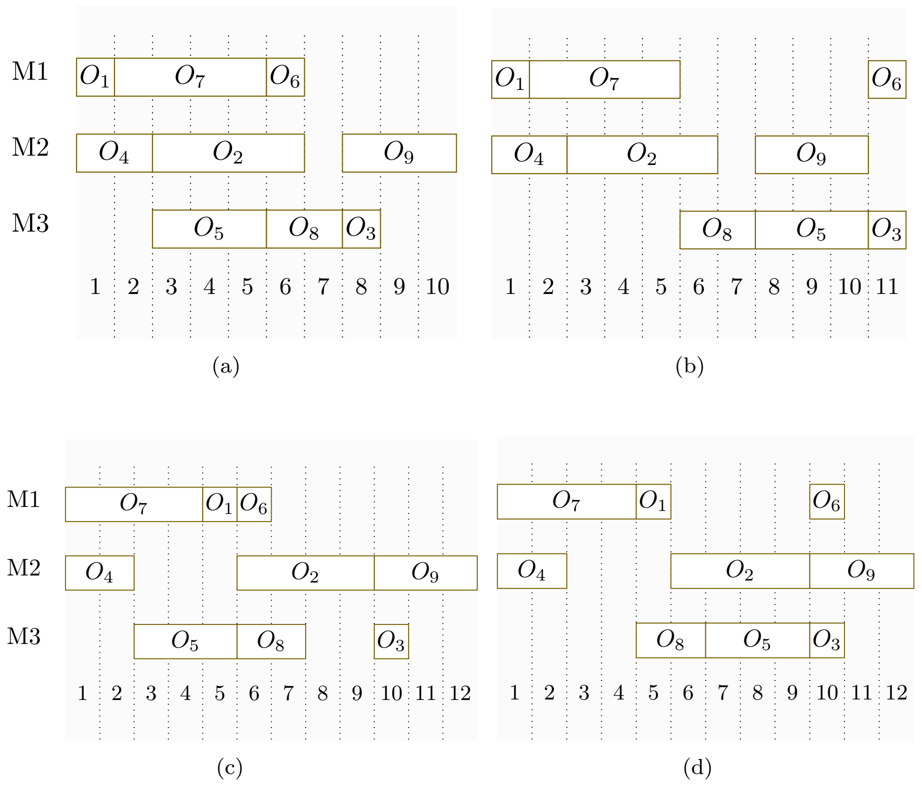


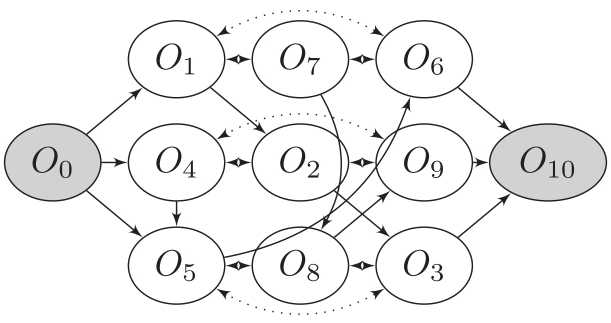
Figure 2. Set of semi-active schedules.

The computation of this *best-case-earliest-starting-time* problem is similar to the computation of the longest path in a disjunctive graph [11, 15]. A disjunctive graph is defined as $G = (N, A, B)$ [42], N denotes the set of nodes (operations) including two fictitious operations; a source operation O_0 and sink operation $O_{(n\times m)+1}$ ($n \times m$ being the number of operations), A is the set of *conjunctive* arcs representing the precedence constraints between operations and denoted by $O_i \rightarrow O_j$ (*i.e.*, O_i precedes O_j). B represents the set of *disjunctive* arcs for each pair of operations that must be performed in the same group and denoted by $O_i \dashleftarrow\dashrightarrow O_j$ which means that either $O_i \rightarrow O_j$ or $O_j \rightarrow O_i$. A feasible semi-active schedule can be obtained as soon as all the disjunctive arcs are selected.

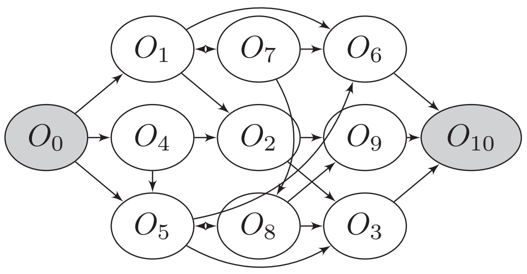
Figures 3a and 3b present respectively the disjunctive graph representation of the initial job shop problem described in Table 1 and the groups of permutable operations shown in Figure 1. These two figures show that the GoPO example is a partial schedule of the initial job shop problem, where some disjunctive arcs are selected. In this GoPO representation, there are two remaining disjunctive arcs $O_1 \dashleftarrow\dashrightarrow O_7$ and $O_5 \dashleftarrow\dashrightarrow O_8$. The selection of these arcs leads to one of the schedules presented in Figure 2.

The literature review on the job shop lower bound problem are based on two main techniques identified by [44]: the adjustment conjunctive constraints based techniques and the classical one-machine relaxation-based techniques [4, 16, 17, 38]. These techniques can be the starting point of the GoPO lower bound computation.

However, computing a lower bound for the *best-case* in a GoPO problem is quite different from computing a lower bound for the optimal schedule of the general job shop problem, because of the conjunctive constraints between groups on the same machines as shown in Figure 3. Moreover, due to these conjunctive constraints, the *best-case* is not necessarily the optimal schedule of the initial general job shop problem. In the next section, this



(a) Initial job shop problem described in table 1



(b) GoPO problem described in fig. 1

FIGURE 3. Disjunctive graph representation.

lower bound calculation is presented in three steps: a job shop lower bound adaptation, the groups' conjunctive constraints improvement and a precedence constraint improvement.

4. Lower bound of the *best-case* schedule

4.1. The *best-case-earliest-starting/completion-time* of an operation

The lower bound of the *best-case-earliest-completion-time* of an operation (χ_i) corresponds to the smallest value of C_i in every semi-active schedule described by GoPO. To compute χ_i we need first to calculate the lower bound of the $best-case-earliest-starting-time$ of this operation (θ_i).

In the first step, we can compute such a lower bound using a relaxation on the resources by making the assumption that each resource has an infinite capacity. In the GoPO problem, θ_i is computed as the maximum of the $best-case-earliest-completion-time$ (lower bound) (χ_j) of all its predecessors: for an operation O_i, its predecessors include the predecessors given by the problem (Γ_i^-) and also the operations of the previous group on the same machine (each operations in G_i^-). In the job shop example described in Figure 1, the predecessors of operation O_6 (executed on M_1) are: operation O_5 (executed on M_3) because of the precedence constraints, and the operations O_1 and O_7 executed on the same machine M_1 because they are in the previous group G_6^-. So, we have:

$$\begin{cases} \theta_i = \max(r_i, \underbrace{\max}_{j\in G_i^-} \chi_j, \underbrace{\max}_{j\in \Gamma_i^-} \chi_j) \\ \chi_i = \theta_i + p_i \end{cases} \tag{4.1}$$

TABLE 2. Computation of the Lower bounds using equation (4.2).

O_i	O_1	O_7	O_4	O_2	O_5	O_8	O_3	O_6	O_9
θ_i	0	0	0	2	2	4	7	5	6
χ_i	1	4	2	6	5	6	8	6	9
$\gamma(G_i)$	5		2	6	7		8	6	9

Calculating θ_i using equation (4.1) is equivalent to the head computation of operation O_i as explained in [15]. [44] used a similar idea based on a one-machine relaxation using the precedence constraints property between operations.

In a second step, we use the precedence constraints between groups to improve the computation of the lower bound; in this case, an operation in a given group G_i cannot be executed before all the operations of its previous group G_i^- have been executed. As a consequence, an operation can only begin after the optimal $\max(C_i)$ of the previous group.

Thus, it needs the computation of the optimal completion time of a group (named as $\gamma(G_i)$). We have previously computed θ_i as a release date so we can generate a $1|r_i|C_{\max}$ problem instance that corresponds to our problem, with $r_i = \theta_i$. This problem is polynomially solvable by ordering the operations in ascending release dates [12,33].

Thus, the improved lower bound for O_i is computed as follows:

$$\begin{cases} \theta_i = \max(r_i, \gamma_{G_i^-}, \underbrace{\max}_{j\in\Gamma^-(i)} \chi_j) \\ \chi_i = \theta_i + p_i \\ \gamma(G_j) = C_{\max} of 1|r_j|C_{\max}, \forall O_j \in G_j, r_j = \theta_j \end{cases} \tag{4.2}$$

Table 2 illustrates the computation of the lower bound of all the groups and operations *best−case−earliest−completion − times* in our job shop example. The computation shows that $\theta_9 = 9$ which is not the optimal completion time of O_9 (= 10) as shown in Figures 2a and 2b.

4.2. Precedence constraint improvement

Using equation (4.2) for the computation of the *best − case − earliest − completion − time* of operation nine (θ_9) has given an error gap of one. It is noticeable that for the calculation of this lower bound, we put O_2 and O_8 on their both *best − starting − time*, as well as O_7 and O_1 transitively, which is inconsistent. We know for a fact that O_7 and O_1 cannot start together at their *best − starting − time* at the same time. In this section, we present a precedence constraint rule to improve θ_i using an adjustment technique based on the precedence consistency of operations belonging to the same group.

This improvement is based on a property about any two operations (O_i and O_j) that belongs to two successive groups (G_i and G_j / $G_j = G_i^+$) on the same machine and have their predecessors (direct or indirect) in the same group (Γ_i^- and $\Gamma_j^- \in G_k$). In this case, one of the operations O_i^- or O_j^- has to start after the other one. Executing one of these operations at first and delaying the other one will have an influence on the completion time of G_j. Figure 4 shows this precedence constraint property for our job shop example.

From the example shown in Figure 4, we have on Machine M_2, operation O_9 that has to start after O_8 because of the precedence constraint, and after operation O_2 because of the group precedence on the same machine. Both predecessors of these operations, O_1 and O_7 are in the same group. Executing one of them first will have an influence on the starting time of operation O_9.

To take into account this situation, we propose an adjustment technique for χ_i that requires two times the calculation of χ_i using equation (4.2); in our job shop example χ_9 will be computed one time for the case when

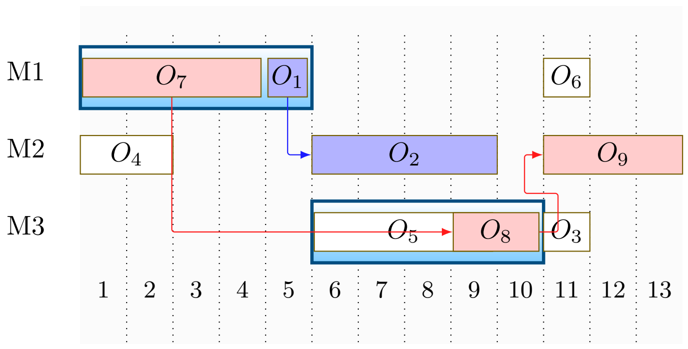


FIGURE 4. Precedence consistency between O_2 and O_9.

TABLE 3. Improved Lower bounds results.

	$r_1 < r_7$			$r_7 < r_1$			
O_i	θ_i^1	χ_i^1	$\gamma_{G_i}^1$	θ_i^2	χ_i^2	$\gamma_{G_i}^2$	$\min(\gamma_{G_i}^1, \gamma_{G_i}^2)$
O_1	0	1	5	4	5	5	5
O_7	1	5		0	4		
O_4	0	2	2	0	2	2	2
O_2	2	6	6	5	9	9	6
O_5	2	5	7	2	5	7	7
O_8	5	7		4	6		
O_3	7	8	8	9	10	10	8
O_6	5	6	6	5	6	6	6
O_9	7	10	10	9	12	12	10

O_1 is executed before O_7 and the other one when O_7 is executed before O_1. The new χ_9 will be the minimum value between these two evaluations. The results of this improvement are shown on Table 3. In this case, the lower bound's improvement gives the optimal value for the $C_{\max}$.

Computing χ_i using equation (4.2) is polynomial and has a complexity of $O(n^2)$. In a GoPo, this precedence constraints property may be found at most $n \times m$ times. So we have a complexity of $O(n^3)$.

4.3. Heads and tails for the best-case schedule

The classical computation of a job shop lower bound is generally based on the calculation of:

- The *best-case-earliest-starting-time*, called *Head*, which represents the longest path from O_* and the operation O_i in a disjunctive graph.
- The *best-case-latest-completion-time* of an operation, called *Tail*, which represents the distance between the latest completion time of an operation and the end of the schedule.

So any improvement of these two values will generally improve the lower bound for the objective function. In a job shop problem, the classical concept of the *one-machine* relaxation is used; each machine is considered alone and heads and tails have to be found for each *one-machine-problem*. The maximum value of these evaluations represents the lower bound of the job shop problem.

In a GoPO problem, we can use this relaxation more efficiently relying on the concept of *one-group-one-machine-problems* relaxation. In this relaxation each group is considered as a *one-machine* and efficient heads and tails are computed for each group.

Using equation (4.2), θ_i is a valid head and must be quite effective. Because of the symmetry of heads and tails, tails can be computed as θ_i using a reversed version of equation (4.2): rather than starting the computation at the beginning of the scheduling problem, the computation begins at the end. So, replacing predecessor by successor, the new formulation is:

$$\begin{cases} \theta_i^{'} = \max(\gamma_{G_i^+}^{'}, \underbrace{\max}_{j \in \Gamma_i^+} \chi_j^{'}) \\ \chi_i^{'} = \theta_i^{'} + p_i \\ \gamma_{G_j}^{'} = C_{\max} of 1|r_j|C_{\max}, \forall O_j \in G_j, r_j = \theta_j^{'} \end{cases} \tag{4.3}$$

With θ_i being a valid head and $\theta_i^{'}$ being a valid tail for the group relaxation. The *one-group-one-machine-problems* are then generated. Then, as for the classical lower bound for the job shop problem, we optimally evaluate the *one-group-one-machine-problems* using the branch and bound method of [14]. The maximal evaluation is a lower bound of the *best-case* schedule of GoPO. In the rest of the article, *LB_best-case* and *LB_best-case(O_i)* denote respectively the lower bound of the *best-case* schedule (for all semi-active schedules generated by the GoPo solution), and the lower bound of the *best-case* schedule when O_i is executed first in its group.

5. Reactive phase of GoPO

During the reactive phase, each time a group containing more than one operation has to be executed, a lower bound of the *best-case* of each operation of this group is calculated. This lower bound (*LB_best-case(O_i)*) can be either used in a decision reactive algorithm or can be proposed in a decision-aid system.

Similarly to the *best-case*, the *worst-case* of an operation (*worst-case(O_i)*) represents the worst *realized schedule* if operation O_i is sequenced first in its group. These two parameters (*LB_best-case(O_i)* and *worst-case(O_i)*) are available to help the decision maker to look for an alternative schedule in case of a disturbance, without the need of rescheduling again. In this context, *LB_best-case(O_i)* should be of good interest for selecting only the schedules with the highest performances.

The algorithm *RA_best-case* described bellow is a reactive algorithm which simulates the decision process based only on *LB_best-case(O_i)* parameter.

Algorithm 1: *RA_best-case* for the reactive phase of GoPO.

```
$L(G) = \{G_1, G_2, \dots G_n\}$;
($G_i$ is a group containing more than one operation) $LB\_best\text{-}case{:=}\ maximum\ value$;
for every group $G_i$ in $L(G)$ do
    while $Card(G_i) > 1$ do
        for every operation $O_i$ in $G_i$ do
            - Put $O_i$ first ;
            - Calculate $LB\_best\text{-}case(O_i)$ ;
        end
        - $LB\_best\text{-}case{:=} \min_{O_i \in G_i}(LB\_best\text{-}case,\ LB\_best - case(O_i))$;
        (Ties are broken using $\theta_i$) - Remove $O_i$ from $G_i$;
    end
    - remove $G_i$ from $L(G)$.;
end
```

As an illustration of how *RA_best-case* works, let us enumerate all the groups of our job shop GoPO example.

$$G_1 : \{O_1, O_7\}, G_2 : \{O_5, O_8\}$$
$$L := \{G_1, G_2\}$$

The reactive procedure is executed on the first group of the list $G_1 : \{O_1, O_7\}$:

- *LB_best-case(O_1)* = *10* and *LB_best-case(O_7)* = *12* (using the result in Tab. 3)
- Remove O_1 from the group and update *LB_best-case* $= \min(10, 12) = 10$
- The current group contains only O_7 and is removed from the list.

Then the reactive procedure is executed on the second group of the list $G_2 : \{O_5, O_8\}$:

- *LB_best-case(O_5)* = *10* and $LB\_best - case(O_8) = 11$
- Remove O_5 from the group
- The current group contains only O_8 and is removed from the list.

At the end of the reactive phase, the *realized schedule* generated by *RA_best-case* has an optimal value of 10. Of course, in this example no disturbances occurred and therefore the final schedule generated by *RA_best-case* (based on the lower bound) is the optimal schedule of GoPO.

Similarly to *RA_best-case*, we create two other reactive algorithms, both based on the *worst-case(O_i)* parameter; the first one called *RA_worst-case-1* is similar to the algorithm described above where the variable *LB_best-case(O_i)* is replaced by *worst-case(O_i)*. The second reactive algorithm is called *RA_worst-case-2* and is similar to *RA_worst-case-1* with an additional condition; ties between operations having the same value of *worst-case(O_i)* are broken using *LB_best-case(O_i)*. The following steps illustrates *RA_worst-case-2* on our job shop example for the two groups, G_1 at first then G_2:

- *worst-case*(O_1) = *11* and *worst-case*(O_7) = *12* (using the polynomial algorithm of [2] described also in [3]), therefore O_1 is sequenced before O_7 using *RA_worst-case-2*.
- Remove O_1 from the group and update *worst-case* $= \min(11, 12) = 11$ and go to the second group.
- *worst-case*(O_5) = *10* and *worst-case*(O_8) = *11* (if for example *worst-case*(O_5) = *worst-case*(O_8) then *LB_best-case*(O_5) and *LB_best-case*(O_8) will be calculated to determine which operation should be sequenced first.
- In this case, O_5 is sequenced before O_8 and the *realized schedule* will have a *makespan* of 10.

In the rest of this section, we assess the relative importance of the three algorithms to their overall performances on a well-known job shop instances.

5.1. Protocol of the experiment

We took a set of benchmark instances called La01 to La40 from [34] with well-known optimal solutions for the *makespan* objective [11, 22, 36]. These forty instances (*i.e.*, from La01 to La40) are widely used in the job shop literature [7, 39]. These are classical job shop instances of different sizes (5 instances for each size) with a range from 10 x 5 to 30×10 where $n \times m$ represents n jobs and m machines. These instances can be downloaded from [6] and each job of these instances can start at time zero.

For each instance, we generate a GoPO schedule with a maximum possible flexibility and such that the *best-case* is not only the optimal schedule of GoPO but also the optimal schedule of the instance[5]. To do so, we use an adaptation of the algorithm of [22] mentioned in Section 2. This algorithm has the following steps:

- **Step 0**. Construct a schedule solving the job shop problem. This schedule is called an *initial schedule* and can be calculated using a branch and bound algorithm or a simple heuristic. In our case, for each of the forty Lawrence instances we took the *optimal schedule* found in [11, 36, 39]
- **Step 1.** Initialize the groups in the machines such that each operation O_i is contained by a group G_i.
- **Step 2.** Calculate the $worst - case$ for each two successive groups that can be merged. If there is no possible merging between the groups in all the machines (due to precedence constraints), finish the algorithm. Otherwise, go to Step 3.
- **Step 3.** Merge the two groups having the minimum $worst - case$ and go back to Step 2.

[5]In this way, we can measure the efficiency of the *best-case* lower bound without using a branch and bound algorithm.

For each GoPO problem type, we compare the the *initial schedule* (optimal) with the *realized schedule* obtained by the three algorithms *RA_worst-case-1*, *RA_worst-case-2* and *RA_best-case*. The algorithms were coded using the JAVA language and the experiments are made on an Intel(R) Core(TM) i5 CPU.

5.2. Results and discussion

The results of these experiments are exposed in Tables 4 and 5. Table 4 presents for each instance, the number of groups generated, the number of decisions for the reactive phase, the *best-case*, Brucker's lower bound [11], the lower bound of the *best-case* (*LB_best-case*) using equation 4.2 and equation 4.3 and the gap error between *LB_best-case* and the *best-case*.

Table 5 presents for each reactive algorithm (*RA_worst-case-1*, *RA_worst-case-2*, *RA_best-case*) the gap error between the *makespan* of the *realized schedule* and the *initial schedule*. For the three algorithms and for all instances, the *realized schedule* were found in almost one second.

Based on the results shown on Table 4, the lower bound computation provided a near-optimal solution with an average gap error less than 1%. Whereas the average gap error of Brucker's lower bound is 2.74%. Moreover, the proposed lower bound gave the optimal solution for twenty-three instances out of forty (57%). Most of these instances are the same as the optimal instances found by Brucker and are problems of 10(jobs) X 5(machines), 15X5, 20X10, 30X10. This is not that surprising, because the proposed lower bound is based on an adaptation of the classical job shop lower bound. Moreover, the proposed lower bound has been computed for less than one second on each instance.

The results presented in Table 5 showed that the reactive algorithm using our improved lower bound (*RA_best-case*) dominates all the instances with a performance about fifteen times (11 615/745) better than *RA_worst-case-2* and twenty times (14667/745) better than that of *RA_worst-case-1*.

RA_best-case gave optimal results for all problems with five machines and all problems with thirty jobs and ten machines except La35; nineteen instances were solved optimally compared to only one instance (La14) by *RA_worst-case-1* and *RA_worst-case-2*. This can be explained by the accuracy of the lower bound calculated for these instances as shown in Table 4.

Compared to *RA_worst-case-1*, *RA_worst-case-2* that uses the *best-case* parameter for breaking ties (*RA_worst-case-2*) performs better; Its average gap error on the forty instances was 26.45% compared to 32.44% for the reactive algorithm that uses only the *worst-case* parameter (*RA_worst-case-1*). But these two algorithms have a poor performance compared to *RA_best-case* which provided a *realized schedule* with an average gap error of only 1.71%. These results suggest that a good decision in a group of permutable operations cannot be relied primarily on the *worst-case* parameter.

Overall, this suggestion is not that surprising in regards of what is expected from the *worst-case* parameter; This parameter is more effective in controlling the performance of the final schedule from not exceeding a performance threshold than guaranteeing an optimal or near-optimal final schedule.

5.2.1. *Performance of the reactive algorithm RA_best-case on a non deterministic environment*

In this section, we investigate the robustness of the reactive algorithm *RA_best-case* in a disturbed environment.

For this, we try to quantify the performance of the *realized schedule* in case of *bad decisions*. A bad decision is called so when the operation chosen first from a group of permutable operations is not the one giving the best-case. For example, in the job shop presented in Section 2, choosing operation O_7 (*LB-best-case(O_7) = 12*) to be executed first in G_1 instead of choosing O_1 (*LB-best-case(O_1)=10*) is considered as a bad decision. This situation may appear whenever a perturbation has occurred on the shop, or simply due to a lack of concentration by the operator. We focus on determining the number of *bad decisions* required to deteriorate the *makespan* of the *realized schedule* with a factor ρ from the optimal schedule (the percentage of performance lost because of a non-optimal schedule). The degraded performance of the *best-case* can be computed as follows: *best-case = optimal-solution * (1 + ρ%).*

TABLE 4. Lower bound results for the best-case parameter (*LB_best-case*).

n X m	Instances	*Number of groups*	*Number of decisions*	*Optimal solution*	*Brucker's Lower bound*	*Lower bound (LB_best-case) (%)*	*Gap error*
10 X 5	La01	19	31	666	666	666	0%
	La02	18	32	655	655	655	0%
	La03	15	35	597	588	588	1.51%
	La04	14	36	590	567	588	0.34%
	La05	15	35	593	593	593	0%
15 X 5	La06	19	56	926	926	926	0%
	La07	17	58	890	890	890	0%
	La08	16	59	863	863	863	0%
	La09	16	59	951	951	951	0%
	La10	19	56	958	958	958	0%
20 X 5	La11	20	80	1222	1222	1222	0%
	La12	21	79	1039	1039	1039	0%
	La13	18	82	1150	1150	1150	0%
	La14	19	81	1292	1292	1292	0%
	La15	22	78	1207	1207	1207	0%
10 X 10	La16	43	57	945	875	931	1.48%
	La17	36	64	784	739	751	4.21%
	La18	40	60	848	770	815	3.89%
	La19	39	61	842	709	796	5.46%
	La20	39	61	902	807	850	5.76%
15 X 10	La21	54	96	1046	995	1045	0.10%
	La22	49	101	927	913	927	0%
	La23	54	96	1032	1032	1032	0%
	La24	50	100	935	881	914	2.25%
	La25	50	100	977	894	954	2.35%
20 X 10	La26	53	147	1218	1218	1218	0%
	La27	56	144	1252	1235	1231	1.68%
	La28	63	137	1273	1216	1273	0%
	La29	60	140	1202	1114	1189	1.08%
	La30	58	142	1355	1355	1355	0%
30 X 10	La31	69	231	1784	1784	1784	0%
	La32	64	236	1850	1850	1850	0%
	La33	64	236	1719	1719	1719	0%
	La34	72	228	1721	1721	1721	0%
	La35	73	227	1888	1888	1888	0%
15 X 15	La36	88	137	1268	1224	1244	1.89%
	La37	84	141	1397	1355	1392	0.36%
	La38	88	137	1196	1077	1147	4.10%
	La39	86	139	1233	1221	1230	0.24%
	La40	86	139	1222	1170	1207	1.23%
Average							0.95%

TABLE 5. Comparison between the three reactive algorithms.

	RA_worst-case-1		*RA_worst-case-2*		*RA_best-case*	
	Gap error	*Gap error(%)*	*Gap error*	*Gap error(%)*	*Gap error*	*Gap error(%)*
La01	155	23.27%	155	23.27%	0	0%
La02	187	28.55%	187	28.55%	0	0%
La03	200	33.50%	282	47.24%	0	0%
La04	184	31.19%	157	26.61%	0	0%
La05	55	9.27%	55	9.27%	0	0%
La06	63	6.80%	63	6.80%	0	0%
La07	132	14.83%	183	20.56%	0	0%
La08	153	17.73%	93	10.78%	0	0%
La09	225	23.66%	97	10.20%	0	0%
La10	71	7.41%	30	3.13%	0	0%
La11	58	4.75%	58	4.75%	0	0%
La12	252	24.25%	119	11.45%	0	0%
La13	351	30.52%	351	30.52%	0	0%
La14	0	0%	0	0%	0	0%
La15	272	22.54%	272	22.54%	0	0%
La16	239	25.29%	301	31.85%	74	7.83%
La17	129	16.45%	119	15.18%	15	1.91%
La18	432	50.94%	302	35.61%	43	5.07%
La19	437	51.90%	354	42.04%	5	0.59%
La20	373	41.35%	451	50.00%	22	2.44%
La21	317	30.31%	317	30.31%	21	2.01%
La22	824	88.89%	481	51.89%	29	3.13%
La23	522	50.58%	417	40.41%	62	6.01%
La24	181	19.36%	220	23.53%	71	7.59%
La25	206	21.08%	271	27.74%	30	3.07%
La26	747	61.33%	715	58.70%	38	3.12%
La27	495	39.54%	402	32.11%	26	2.08%
La28	326	25.61%	284	22.31%	1	0.08%
La29	645	53.66%	410	34.11%	29	2.41%
La30	527	38.89%	266	19.63%	25	1.85%
La31	416	23.32%	324	18.16%	0	0%
La32	681	36.81%	370	20.00%	0	0%
La33	774	45.03%	677	39.38%	0	0%
La34	352	20.45%	336	19.52%	0	0%
La35	562	29.77%	328	17.37%	31	1.64%
La36	647	51.03%	530	41.80%	41	3.23%
La37	579	41.45%	400	28.63%	40	2.86%
La38	746	62.37%	457	38.21%	45	3.76%
La39	461	37.39%	321	26.03%	38	3.08%
La40	691	56.55%	460	37.64%	59	4.83%
$\sum$	14 667		11 615		745	
Average		32.44%		26.45%		1.71%

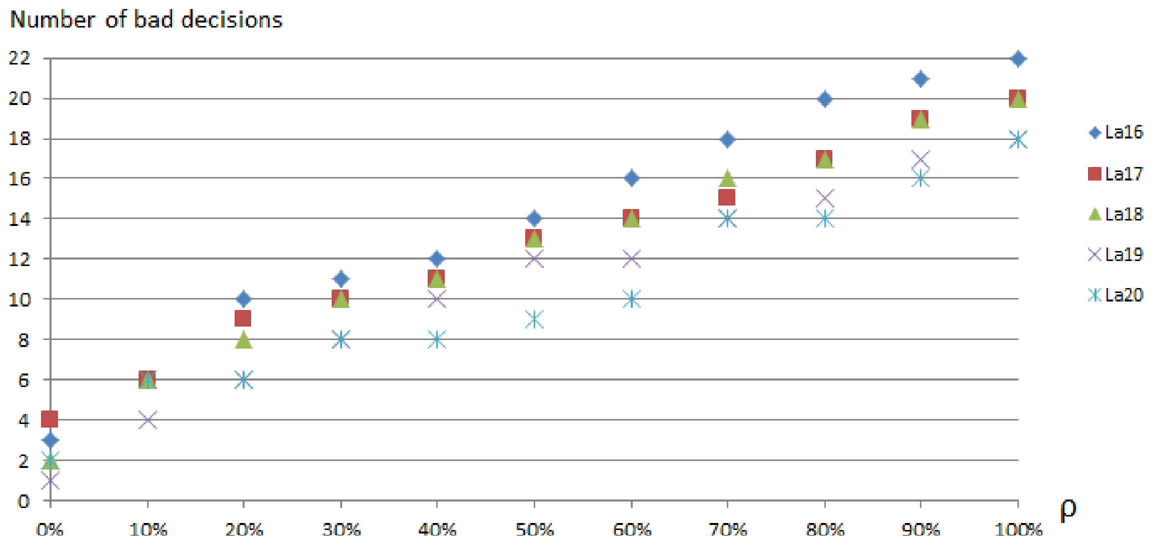


FIGURE 5. Number of bad decisions to deteriorate the *makespan* with a distance factor ρ from the optimal schedule.

To determine the number of *bad decisions* required to deteriorate the *makespan*, we incorporate these bad decisions randomly in the reactive algorithm *RA_best-case* during the decision-aid process. This algorithm was executed fifty times on each square instance (La16 to La20); as these instances have been found to be the most difficult instances in practice for job shop problems [24]. Then, the number of bad decisions has been measured according to $\rho\%$ as shown in Figure 5.

The results obtained showed that for an optimal *realized schedule* ($best-case = optimal-solution * (1+0\%)$), four bad decisions out of 57 decisions (4/57) does not have any impact on the optimal *realized schedule* obtained for La17. For the other instances: La16, La18, La20 and La19, the number of bad decisions are respectively 3/57, 2/60, 2/61 and 1/60. It means that, for La19, two bad decisions are enough to deteriorate the *makespan*.

For all other variations of ρ, La16 gave the maximum number of possible bad decisions. For example, when $\rho = 100\%$, the reactive algorithm was allowed to take 22 bad decisions, which represents 38% of the total decisions. It can be noticed that the average number of bad decisions for all the five instances almost varies linearly with regards to ρ with a coefficient of 3% (the percentage number of bad decisions with regards to the total number of decisions). Therefore, the percentage number of bad decisions may be expressed by the following equation:

$$\approx 3\% + (\rho/10\% \times 3\%)$$

It can be suggested that this computation can be used as a post-evaluation performance deviation check in order to trigger the operator concentration before taking further decisions.

6. Conclusion

In this article we have proposed an evaluation of the *base-case* parameter in a GoPO schedule. This evaluation relies on a computation of an efficient lower bound, which is calculated in three steps. The first step is an application of the general job shop lower bound to the GoPO problem. The second step uses the precedence property between groups in GoPO to improve this lower bound. This step needs the calculation of a one-machine problem adapated to GoPO. For this, we proposed the one-machine-one-group relaxation such that the lower bound of each group is considered separately. Then, in the final step, we introduced a new precedence constraint property to tighten this lower bound.

This lower bound can be used as a decision-aid parameter during the reactive phase of GoPO. To evaluate the efficiency of this parameter, three reactive algorithms for GoPO are proposed: the first two algorithms relies primarily on the *worst-case* parameter, whereas the third algorithm relies only on the *best-case* parameter. These three algorithms have been implemented in a job shop benchmark problem using the *makespan* objective. The comparative results show the benefits of the proposed parameter (*best-case*) which exhibits clearly better performance than the *worst-case* parameter.

The experimentation of the $best - case$ reactive algorithm has been extended to a disturbed environment. The results obtained has confirmed that the proposed parameter could be of good use during the reactive phase of GoPO in order to maintain a good performance in presence of disruptions.

However, there is still room for improvement:

- The proposed lower bound should also be implemented in a branch and bound method in order to evaluate the exact best possible performance of a flexible schedule.
- The calculation of this parameter may be generalized on other regular objectives or multi-objective approaches.

For further research, the proposed parameter should be implemented in a real decision-aid system involving the human temper factor in order to confirm and extend the experimental theoretical results.

References

[1] M.A. Alloulou and C. Artigues, Worst-case evaluation of flexible solutions in disjunctive scheduling problems. *ICCSA'07 Proceedings of the 2007 international conference on Computational science and its applications - Volume Part III.* In Vol. 1205 of *Lecture Notes in Computer Science* (2007) 1027–1036.

[2] C. Artigues, J.C. Billaut and C. Esswein, Maximization of solution flexibility for robust shop scheduling. *Eur. J. Operat. Res.* **165** (2005) 314–328.

[3] Ch. Artigues, J.-Ch. Billaut, A. Cherif, N. Mebarki and Z. Yahouni, Robust machine scheduling based on group of permutable jobs. In: Robustness Analysis in Decision Aiding, Optimization, and Analytics, edited by M. Doumpos, C. Zopounidis and E. Grigoroudis International Series in Operations Research & Management Science, vol 241. Springer, Cham (2016).

[4] P. Baptiste and C. Le Pape, Edge-finding constraint propagation algorithms for disjunctive and cumulative scheduling. In *Proceedings 15th Workshop of the U.K. Planning Special Interest Group* (1996).

[5] J.C. Bean, J.R. Birge, J. Mittenthal and C.E. Noon, Match-up scheduling with multiple resources, release dates and disruptions. *Option Res.* **39** (1991) 470–483.

[6] J. Beasley, Job shop benchmark instances (2004).

[7] D. Behnke and M.J. Geiger, Test instances for the flexible job shop scheduling problem with work centers. Technical report, Universitat Der Bundewehr Hamburg (2012).

[8] J. Bidot, Th. Vidal, Ph. Laborie and J. Ch. Beck, A theoretic and practical framework for scheduling in a stochastic environment. *J. Scheduling* **12** (2008) 315–344.

[9] J.C. Billaut, A. Moukrim and E. Sanlaville, Flexibility and Robustness in Scheduling. Wiley-ISTE (2008).

[10] J.R. Birge, M.A. Dempster, H.I. Gassman, E.A. Gunn, A.J. King and S.W. Wallace, A standard input format for multi-period stochastic linear programs. *Math. Program. Soc.* **17** (1987) 1–21.

[11] P. Brucker, B. Jurisch and B. Sievers, A branch and bound algorithm for the job-shop scheduling problem. *Discrete Appl. Math.* **49** (1994) 107–127.

[12] P. Brucker and S. Knust, Complexity results for scheduling problems (2007).

[13] O. Cardin, N. Mebarki and G. Pinot, A study of the robustness of the group scheduling method using an emulation of a complex fms. *Inter. J. Production Econ.* **146** (2013) 199–207.

[14] J. Carlier, The one machine problem. *Eur. J. Oper. Res.* (1982) 42–47.

[15] J. Carlier and E. Pinson, An algorithm for solving the job-shop problem. *Manag. Sci.* **35** (1989) 164–176.

[16] J. Carlier and E. Pinson, A practical use of jackson's preemptive schedule for solving the job shop problem. *Ann. Oper. Res.* **26** (1991) 269–287.

[17] J. Carlier and E. Pinson, Adjustment of heads and tails for the job-shop problem. *Eur. J. Oper. Res.* **78** (1994) 146–161.

[18] F.T.S. Chan, H.K. Chan and H.C.W. Lau, The state of the art in simulation study on fms scheduling: a comprehensive survey. *Inter. J. Adv. Manuf. Techn.* **19** (2002) 830–849.

[19] E.B. Da Silva, M.G. Costa, M.F. da Silva and F.H.I. Pereira, Simulation study of dispatching rules in stochastic job shop dynamic scheduling. *World J. Model. Simul.* **10** (2014) 231–240.

[20] A.J. Davenport and J.C. Beck, A survey of techniques for scheduling with uncertainty (2000).

[21] Erschler and Roubellat, An approach for real time scheduling for activities with time and resource constraints Advances in project scheduling. Edited by R. Slowinski and J. Weglarz. Elsevier (1989).

[22] C. Esswein, *Un apport de flexibilité séquentielle pour l'ordonnancement robuste.* Ph.D. thesis, Université François Rabelais Tours, France (2003).

[23] C. Fang, R. Kolisch, L. Wang and C. Mu, An estimation of distribution algorithm and new computational results for the stochastic resource-constrained project scheduling problem. *Flexible Services and Manufacturing J.* **27** (2015) 585–605.

[24] H. Fisher and G.L. Thomson, Probabilistic learning combinations of local job-shop scheduling rules. Industrial Scheduling (1963) 225–251.

[25] N. Fu, P. Varakantham and H.Ch. Lau, Towards finding robust execution strategies for rcpsp/max with durational uncertainty. In *Proc. 20th Inter. Confer. Automated Planning and Scheduling, ICAPS 2010, Toronto, Ontario, Canada. 12–16* 2010 (2010) 73–80.

[26] G. Gallego, Linear control policies for scheduling a single facility after an initial disruption. Technical report, School of operations research and industrial engineering. College of Engineering, Cornell univeresity, ITHACA, New york 14853 (1988).

[27] G. Gallego, Produce-up-to policies for scheduling a single facility after an initial disruption. Technical report. School of operations research and industrial engineering, College of Engineering, Cornell univeresity, ITHACA, New york 14853 (1988).

[28] E.M Goldratt, *Critical Chain.* North River Press (1997).

[29] Graham, Lawler, Lenstra and K. Rinnooy, Optimization and approximation in deterministic sequencing and scheduling: a survey. *Ann. Discrete Math.* **5** (1979) 287–326.

[30] W. Herroelen and R. Leus, Project scheduling under uncertainty: Survey and research potentials. *Europ. J. Oper. Res.* **165** (2005) 289–306.

[31] P. Kouvelis and G. Yu, Robust Discrete Optimization and Its Applications. Kluwer Academic Publishers (1997).

[32] O. Lambrechts, E. Demeulemeester and W. Herroelen, Proactive and reactive strategies for resource-constrained project scheduling with uncertain resource availabilities. *J. Scheduling* **11** (2008) 121–136.

[33] E.L. Lawler, Optimal sequencing of a single machine subject to precedence constraints. *Manag. Sci.* **19** (1973) 544–546.

[34] S. Lawrence, Resource constrained project scheduling: an experimental investigation of heuristic scheduling techniques (supplement) (1984).

[35] N. Mebarki and A. Shahzad, Correlation among tardiness-based measures for scheduling using priority dispatching rules. *Inter. J. Produc. Res.* **51** (2013) 3688–3697.

[36] K. Morikawa and K. Takahashi, A flexible branch and bound method for the job shop scheduling problem. *Industrial Eng. Manag. Syst.* **8** (2009) 239–246.

[37] Ch. Muise, J. Ch. Beck and Sh.A. McIlraith, Flexible execution of partial order plans with temporal constraints. In *Inter. Joint Confer. Artificial Intell.* (2013) 2328–2335.

[38] W. Nuijten and C. Le Pape, Constraint-based job shop scheduling with ilog sheduler. *J. Heuristics* **3** (1998) 271–286.

[39] G. Pinot and N. Mebarki, Best-case lower bounds in a group sequence for the job shop problem. In *17th IFAC World Congress* (2008) 14876–14881.

[40] N. Policella, A. Oddi, S.F. Smith and A. Cesta, Generating Robust Partial Order Schedules. Springer Berlin Heidelberg, Berlin, Heidelberg (2004) 496–511.

[41] F. Roubellat, J.-C. Billaut and M. Villaumié, Ordonnancement d'atelier en temps réel: d'orabaid à ordo. *Revue Automat. Productique Appl.* **8** (1995) 683–713.

[42] B. Roy and B. Sussmann, Les problèmes d'ordonnancement avec contraintes disjonctives. Rapport de recherches n° 9. SEMA (1964).

[43] I. Sabuncuoglu and S. Goren, Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research. *Inter. J. Comput. Integrated Manufacturing* **22** (2009) 138–157.

[44] F. Sourd and W. Nuijten, Multiple-machine lower bounds for shop-scheduling problems. *INFORMS J. comput.* **12** (2000) 341–352.

[45] K. Swanson, J. Bresina and M. Drummond, Just-in-case scheduling. In *AAAI conference*, Seattle (1994).

[46] K. Szczesny and M. Knig, Reactive scheduling based on actual logistics data by applying simulation-based optimization. *Visualiz. Eng.* **3** (2015).

[47] R. Tadei, F. Della Croce and G. Menga, Advanced search techniques for the job shop problem: a comparison. *RAIRO: OR* **29** (1995) 179–194.

[48] S. Van de vonder, E. Demeulemeester and W. Herroelen, A classication of predictive-reactive project scheduling procedures. *J. Scheduling* **10** (2007) 195–207

[49] P. Wojakowski and D Waroek, The classification of scheduling problems under production uncertainty. *Res. Logistics Prod.* **4** (2014) 245–255.