Using real-time information to reschedule jobs in a flowshop with variable processing times*

Jose M. Framinan† Victor Fernandez-Viagas Paz Perez-Gonzalez

Industrial Management, School of Engineering,
University of Seville. Camino de los Descubrimientos s/n. 41092 Seville, Spain

Abstract

In a time where detailed, instantaneous and accurate information on shop-floor status is becoming available in many manufacturing companies due to Information Technologies initiatives such as Smart Factory or Industry 4.0, a question arises regarding when and how this data can be used to improve scheduling decisions. While it is acknowledged that a continuous rescheduling based on the updated information may be beneficial as it serves to adapt the schedule to unplanned events, this rather general intuition has not been supported by a thorough experimentation, particularly for multi-stage manufacturing systems where such continuous rescheduling may introduce a high degree of nervousness in the system and deteriorates its performance. In order to study this research problem, in this paper we investigate how real-time information on the completion times of the jobs in a flowshop with variable processing times can be used to reschedule the jobs. In an exhaustive computational experience, we show that rescheduling policies pay off as long as the variability of the processing times is not very high, and only if the initially generated schedule is of good quality. Furthermore, we propose several rescheduling policies to improve the performance of continuous rescheduling while greatly reducing the frequency of rescheduling. One of these policies, based on the concept of critical path of a flowshop, outperforms the rest of policies for a wide range of scenarios.

Keywords: Rescheduling, Flowshop, Industry 4.0, Critical Path

*Preprint submitted to the journal Computers & Industrial Engineering. DOI: https://doi.org/10.1016/j.cie.2019.01.036

†Corresponding author. Tel. +3495 448 7327; fax: +3495 448 7329. E-mail address: framinan@us.es

1 Introduction

Recent developments in Industrial Informatics and Information and Communications Technology –such as Industry 4.0– seem to anticipate a situation in which the shop floor status in many manufacturing companies is instantly available (Chen et al., 2016). The potential challenges and advantages of using this real-time data in production management have been discussed in different contributions, such as e.g. Waschneck et al. (2017). In this paper, we intend to focus on the potential of using such real-time information to improve scheduling decisions in a manufacturing scenario where the processing times of the jobs are subject to variability. More specifically, we wish to assess the advantages of using information about the jobs already processed in some machines to reschedule the subsequent jobs yet in a permutation flowshop where the processing times are not deterministic. The choice of the flowshop environment is motivated by the enormous attention that this layout has attracted from both academia and practice due to its challenging theoretical complexities and wide applications in assembly/manufacturing industries, as well as in medical operations (see e.g. Liu et al., 2017a for a recent recollection of flowshop scheduling applications, and Fernandez-Viagas et al., 2017 for a recent review of the scientific literature). We consider the makespan as an indicator of the performance of system, as it is undoubtedly the most common objective for flowshop scheduling in general and specifically for the study of this problem under uncertainty (González-Neira et al., 2017).

Despite this interest, the literature on rescheduling in the flowshop setting is rather scarce as compared to rescheduling in other layouts. Even the analysis of rescheduling literature in other layouts does not shed too much light on the subject of our research: In general, it is acknowledged that the performance of the system is related to the frequency of rescheduling, which seems intuitive as the schedule is continuously modified to take into account events that could not have been foreseen at the time the initial schedule was developed. However, these rather general results are obtained for the case where an infinite arrival of jobs is considered and it has to be noted that, in a permutation flowshop setting, all jobs must follow the same route across all machines, so once a job enters into the shop floor (i.e. in the first machine), then it should be processed according to the same sequence in the rest of the machines. Since the sequence of the jobs already in the shop cannot change, the only

point in time where the shop floor information can be used is when a job has completed its processing on the first machine, leaving this machine free for the subsequent jobs. In these circumstances, it is not clear under which conditions rescheduling pays off: First, the permutation constraint limits the number of choices (and potential advantages) for rescheduling. Second, although more data are known, the rescheduling procedure has to rely on estimates of the completion times of the jobs already in the shop, therefore some procedures to estimate such completion times and to perform an efficient rescheduling –usually time consuming– have to be provided.

More specifically, in this paper we address the following two (interrelated) research questions:

1. Are there scenarios where it is advantageous to use real-time information on the (actual) completion times of the jobs already in the shop to reschedule the remaining jobs so the performance of the system is improved? What are the main features of these scenarios?

2. In those scenarios, how rescheduling should be performed? I.e. Are sophisticated methods required for schedule generation and/or rescheduling, or relatively simple approaches might suffice?

Note that the answers to these research questions have important implications for schedulers, as they greatly influence the approach to schedule jobs in flowshops subject to processing times variability. In the subsequent experimentation carried out in the paper, we will show, with respect to 1), that there are advantages in resequencing using the available information if the variability of processing times is low or medium. These advantages blur for scenarios where the variability of the processing times is high. Furthermore, we will show that the quality of the initial schedule is critical to achieve such superior performance, so resequencing an initial schedule of poor quality does not lead to a good system's performance. Regarding 2), we propose several event-driven rescheduling policies that compare favourably to a continuous rescheduling policy while being much less time consuming. Most notably, triggering the rescheduling procedure based on the disruption of the critical path of the base schedule provides excellent results in terms of performance and of rescheduling requirements.

The remainder of the paper is as follows: Section 2 presents the problem background and discusses the related literature, while in Section 3 we formalise our problem and introduce the

required notation. In Section 4 we present different strategies that can be adopted to carry out the resequencing process. The computational experience is described in Section 5 together with the main results, while in Section 6 we present the conclusions of our work and point out future research lines.

2 Background

In actual manufacturing systems, production is a highly dynamic process subject to different unexpected events and requirements (Hao and Lin, 2010). Therefore, shop floor managers must consider different strategies, policies and methods to cope with this changing environment. The classical paper by Vieira et al. (2003) describes a general framework to classify such strategies, policies and methods and, in the following discussion of the related literature, we basically adopt their framework, although we further differentiate some sub-categories to make explicit some the options. More specifically, we consider that there are two main approaches for shop floor scheduling in dynamic environments (Church and Uzsoy, 1992), i.e. *dispatching rules*, or *schedule generation*.

Under the approach of dispatching rules, whenever a machine becomes free, a priority rule is employed to decide what job to process next. This approach is also known as a completely reactive approach (Aytug et al., 2005). Dispatching rules are usually fast and can be implemented easily, which constitutes a great advantage. However, there is empirical evidence that for complex systems with high competition for resources, the schedule generation approach has the potential to significantly outperform dispatching rules (see e.g. Ovacik and Uzsoy, 1994). Furthermore, in our specific case, the constraint regarding the permutation of the jobs across the machines greatly limits the potential of using dispatching rules.

Under the schedule generation approach, a schedule is developed so the starting times of each job on each machine is determined in advance. Since different unforeseen events may take place, two different strategies can be adopted:

- To maintain the initial schedule —also labelled *base schedule*–, other than a time shifting in the Gantt chart (i.e. right-shifting or left-shifting the starting times of the jobs to delay or advance the schedule but without changing the relative order of the operations). This option

is also labelled as *fixed sequencing* (Sabuncuoglu and Kizilisik, 2003). Clearly, the suitability of this option depends on how the initial schedule has been generated and the nature of the unforeseen events. If the schedule has been generated assuming a deterministic scheduling setting, it is likely that the performance of the initial schedule is substantially worsened with respect to the initial results expected. Another possibility is to develop a schedule which explicitly considers the probability of unforeseen events when generating the initial schedule, usually by minimizing the expected value of the objective function(s), i.e. to use stochastic scheduling techniques to develop the base schedule. Finally, a third alternative would be to employ (deterministic) robust scheduling in order to produce an initial schedule which is rather insensitive to the unexpected events, i.e. the potential performance of the initial schedule is sacrificed so a schedule is generated in order that its expected behaviour is not too much degraded in front of unforeseen events.

- To reschedule, i.e. to modify the scheduling in view of the new data available. This approach is labelled as scheduling/rescheduling (Sabuncuoglu and Kizilisik, 2003) or proactive-reactive (Aytug et al., 2005), and it is the most commonly used method under uncertainty (Liu et al., 2017b). The main decision in this case is to decide when the rescheduling process is triggered (i.e. the so-called *rescheduling policy*). Different types of rescheduling policies may be adopted (Church and Uzsoy, 1992):

  - Continuous rescheduling (CR). Rescheduling is performed every time an event that is recognised by the system (such as the arrival of new jobs, machine breakdown, or the completion time of a job), occurs.
  - Periodic rescheduling (PR). Rescheduling is performed periodically in given time intervals –named rescheduling points–, which have been fixed in advance. The events occurring between rescheduling points are ignored until the following rescheduling point.
  - Event-driven rescheduling (EDR). Rescheduling is triggered upon the fulfilment of certain conditions related to the status of the shop floor. Note that both CR and PR can be seen as a particular case of EDR (indeed this is the case in the framework by Vieira et al., 2003), nevertheless we wish to explicitly differentiate these options as it is also

customary in the related literature (see e.g. Aytug et al., 2005).

In addition, it is commonplace to classify the manufacturing environment where the above scheduling decisions take place either as *static* (the number of jobs to be scheduled is finite), or *dynamic* (the number of jobs to be scheduled is assumed infinite), see Vieira et al. (2003).

Equipped with the above framework, we can classify the related literature on the flowshop layout. Apart from several work dealing with dispatching rules (see e.g. El-Bouri, 2012, El Bouri and Amin, 2015 and Heger et al., 2016), several contributions can be noted with respect to rescheduling approaches. Akturk and Gorgulu (1999) suggest a strategy by which, after machine failure, part of the initial schedule is rescheduled to match up with the original schedule at some point in time. Since rescheduling takes place after each machine failure, their approach can be considered a CR policy. Swaminathan et al. (2007) address the problem of flowshop scheduling where new jobs arrive over time using a CR policy. They study the differences between enforcing the use of permutation schedules or not. Their results conclude that there may be significant gains in performance when the permutation requirement is relaxed. The paper by Katragjini et al. (2013) addresses rescheduling in a dynamic permutation flowshop with different types of disruptions –random job arrivals, machine breakdowns, and job ready time variations–, and presents a metaheuristic to continuously reschedule the jobs with the two objectives of system performance and schedule stability. Rahman et al. (2013) presents the order acceptance problem in a dynamic permutation flowshop subject to random job arrivals. If the newly arrived job is accepted, then it can be scheduled using a right-shifting strategy, or by conducting a reschedule using a memetic algorithm. The results conclude that rescheduling using a memetic algorithm produces a lower makespan. Their approach also uses a CR policy. Rahmani and Heydari (2014) address the problem of schedule generation and rescheduling in a dynamic flowshop with variability of the processing times and random job arrivals. The schedule generation is aimed to obtain a robust solution whereas a multi-objective linear programming model (with makespan, schedule stability and robustness as objectives) is used to perform a continuous rescheduling. The procedure is favourably compared against other heuristics. Liu et al. (2016) study a permutation flowshop with sequence dependent setup times and different types of disruption events (including processing time variation). They use a multi-objective metaheuristic to conduct a CR policy in order to obtain a Pareto set of indicators of system's performance and deviation

from initial schedule, which is obtained using a deterministic hybrid Genetic Algorithm. Liu et al. (2017a) address the problem of scheduling a flowshop with machine breakdowns and unexpected job arrivals under a predictive-reactive approach using CR. The rescheduling procedure aims at considering simultaneously the robustness of the schedule, and customer and users satisfaction. Finally, Liu et al. (2017b) address the problem of rescheduling a new order (consisting of many jobs) arriving to a dynamic permutation flowshop where the objective is to minimise the makespan of the new jobs while not violating the due date of the set of old jobs. Different CR policies mixing new and old jobs while satisfying the problem constraints are tested.

As it can be seen from the review of the related literature, although rescheduling in permutation flowshops has been addressed in several contributions, none of the references focuses on the research questions posed in Section 1. More specifically, it can be seen that there is no study assessing the potential advantages of rescheduling as compared with not to perform it. Despite its intuitive advantages, there is also a substantial body on contradictory research that discourages frequent replanning and scheduling in many shop floor layouts (see e.g. the review by Hozak and Hill, 2009), therefore this issue should be properly assessed. Furthermore, most of the studies assume a dynamic flowshop. Finally, regarding the different rescheduling procedures and policies, it can be seen that the most common is to perform a CR policy and to reschedule using sophisticated approximate methods. However, this renders the rescheduling process extremely time consuming so it may not be feasible from a practical point of view (Pfeiffer et al., 2007). Therefore, it is of interest to investigate whether there are other, less CPU-intensive, efficient policies and procedures. To address these issues, first we formalise the problem under consideration in order to develop the computational experience, while in Section 4 we discuss the different rescheduling policies and procedures that can be considered.

3 Problem statement

In the deterministic permutation flowshop scheduling problem, there are n jobs to be processed sequentially on a set of m machines. The permutation constraint indicates that the sequence of the jobs is the same for all machines. Additionally, it is usually assumed that machines are always

available, and that there are no setup times. For a complete list of the assumptions in the classical permutation flowshop problem, see e.g. Framinan et al. (2014).

Due to the permutation constraint, a sequence of jobs Π specifies a complete solution to the problem. Let us denote $C_{ij}(\Pi)$ the completion time on machine i of job in order j in sequence Π. Then, the maximum completion time or makespan of the solution Π can be computed using the following recursive expression:

$$C_{ij}(\Pi) = \max\{C_{i-1,j}; C_{i,j-1}\} + p_{ij} \quad \forall i = 1, \ldots, m, j = 1, \ldots, n \tag{1}$$

where p_{ij} denotes the processing time of job in position j ($j = 1, \ldots, n$) on machine i ($i = 1, \ldots, m$). Clearly, $C_{0,j} = 0$ and $C_{i,0} = 0$. Then, $C_{max} = C_{mn}$. Among all possible sequences, the goal is to find the one yielding the minimum makespan. This decision problem is well-known to be NP-hard for $m > 2$.

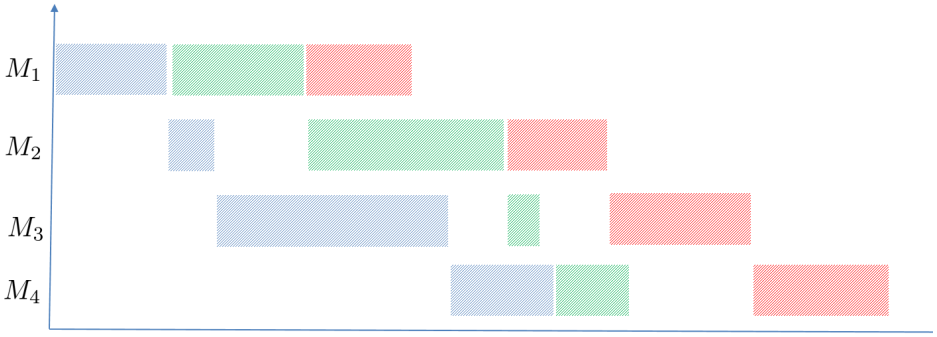
In our research we assume that the processing times are random variables with expected value $\tilde{p}_{ij}$. The objective is here to minimise the expected makespan. Clearly, since p_{ij} are random variables, so are C_{ij}, see (1). An estimate of the values of C_{ij} – denoted as $\tilde{C}_{ij}$ – can be computed using the expected value of the processing times, i.e.:

$$\tilde{C}_{ij}(\Pi) = \max\{\tilde{C}_{i-1,j}; \tilde{C}_{i,j-1}\} + \tilde{p}_{ij} \quad \forall i = 1, \ldots, m, j = 1, \ldots, n \tag{2}$$

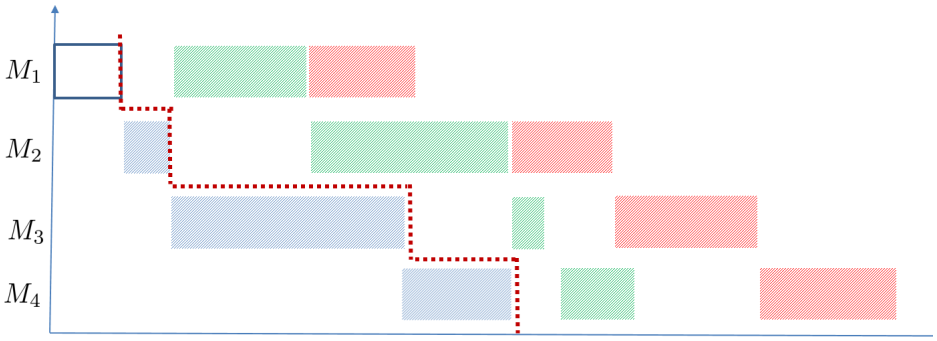
also $\tilde{C}_{0,j} = 0$ and $\tilde{C}_{i,0} = 0$.

The stochastic behaviour of the flowshop motivates that the completion times of the jobs in the initial schedule (see Figure 1a) do not have to be the same as the actual completion times when the schedule is executed (see Figure 1b). More specifically, whenever job in position j in sequence Π has completed its processing on the first machine of the flowshop, this machine is available for the next job. In this situation, two options are possible:

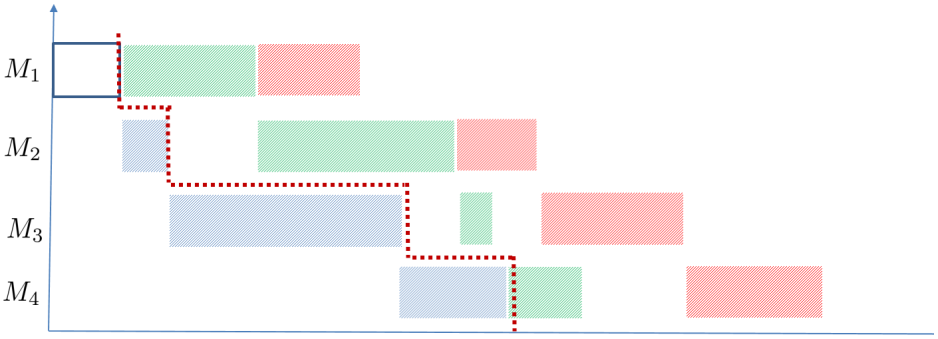
- To proceed with the next job in Π, that is, to enter job in position $j + 1$ in the first machine. We will refer to this option as to *no rescheduling*, as it results in sequencing the jobs according to the initial sequence Π (except perhaps some right/left shifting of the jobs). This situation is depicted in Figure 1c.



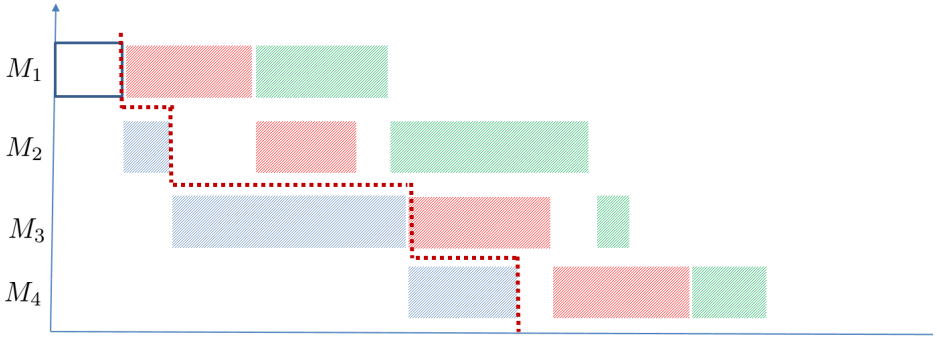
(a) Initial schedule



(b) Actual completion time of the job in the first machine and new (expected) machine availability



(c) No rescheduling (except left shifting of the scheduled jobs)



(d) Rescheduling in view of the expected machine availability

Figure 1: Options when the actual processing times on the first machine do not match the expected processing times.

- To resequence the jobs in positions $j+1$ to n in Π to take into account the possible early/tardy completion of the preceding jobs with respect to the initial sequence, see Figure 1d. Clearly, resequencing the remaining jobs is equivalent to solving a decision problem with machine availability constraints (see Perez-Gonzalez and Framinan, 2009 for a detailed analysis of this problem). In the first machine, these availability is caused by the actual completion time of job in position j, whereas for the remaining machines $2, \ldots, m$ is an *estimated* availability based on the expected completion time of job in position j in these machines, even though it incorporates some information regarding the actual completion times of the jobs.

As discussed in Section 2, there are potential merits in both options: On the one hand, re-sequencing uses the available (real-time) data to cope with the uncertainty of the shop floor. On the other hand, deterministic scheduling based on average processing times performs extremely well for the stochastic flowshop, as shown by Framinan and Perez-Gonzalez (2015). Furthermore, resequencing may introduce excessive nervousness in the scheduling process with perhaps little practical advantage, given the fact that, although using some additional data, the computation of the completion times of the remaining jobs is still based on estimates. Finally, if a rescheduling strategy is to be carried out, its performance would depend on the rescheduling strategy adopted, i.e. when to schedule (rescheduling frequency) and how to schedule (rescheduling procedure) (Sabuncuoglu and Kizilisik, 2003). Different rescheduling strategies that can be adopted will be discussed in Section 4.

A final remark is that, clearly, re-sequencing does not come at zero computational costs, as in practical terms means that a new sequence has to be found in near real-time. However, although extremely important in practice, in this research we will set aside this issue and assume that the time can be 'frozen' so a true re-optimisation of the partial sequence can be conducted. By doing so, we aim to establish the best case regarding the resequencing option, as it may turn out that, under certain conditions, it is not profitable to re-sequence even if we could afford very long decision intervals. This is also the usual strategy adopted in the rescheduling papers reviewed in Section 2.

4 Rescheduling strategies in permutation flowshops

In this section, we discuss different rescheduling strategies that may be adopted. In the flowshop setting, the following issues have to be dealt to fully describe a rescheduling strategy:

1. The generation of the initial schedule, i.e. which procedure is to be used to generate a base schedule.

2. The estimation of the machine availability for rescheduling: Since rescheduling with the permutation flowshop constraint implies not to alter the sequence of the job already on the shop, the completion times of these jobs have to be estimated to determine the (estimated) availability of the machines in order to reschedule the remaining jobs.

3. The rescheduling procedure adopted, i.e. the algorithm to be employed to solve the resulting scheduling problem with machine availability constraints (using the estimated machine availability in #2).

4. The rescheduling policy that is adopted, i.e. when the rescheduling procedure identified in #3 is triggered.

The generation of the initial schedule (#1) and the rescheduling procedure (#3) are discussed in Section 4.2, while #2 is described in Section 4.1. Finally, different rescheduling policies (#4) are presented in Section 4.3.

4.1 Procedure for the estimation of machine availability

Upon the completion of the processing of job in position j in the first machine of the flowshop, we can estimate the machine availability that would experience the jobs that have not yet entered in the shop. More specifically, let $a_i(j)$ denote the availability of machine i after the processing of job in position j in the first machine. The computation of this machine availability is illustrated in Figure 2 and entails the following steps:

- For the first machine ($i = 1$), $a_1(j)$ is given by the real completion time of job in position j on the first machine, i.e. :

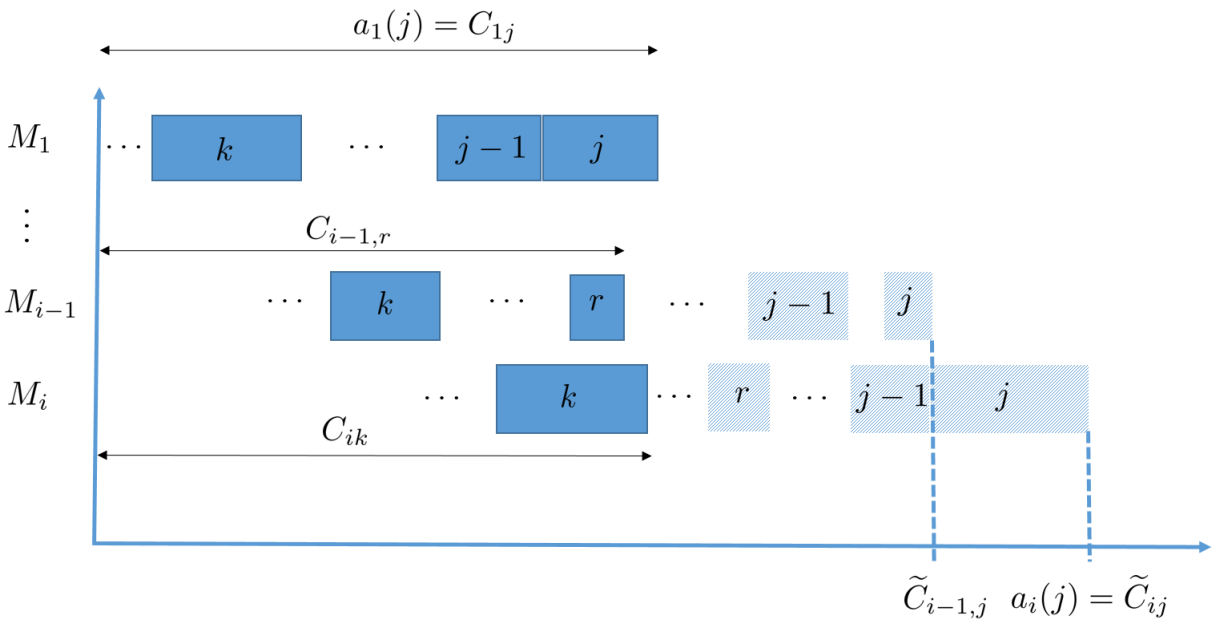


Figure 2: Illustration of the estimation of machine availability

$$a_1(j) = C_{1j}.$$

- For the rest of the machines ($i = 2, \ldots, m$), two sub-cases can be distinguished:

  - If, by the time the job in position j is finished on machine 1, machine i has completed the processing of job in position $j-1$ (i.e. if $a_1(j) > C_{i,j-1}$), then the expected availability is given by:

$$a_i(j) = a_{i-1}(j) + \tilde{p}_{ij} \quad i = 2, \ldots, m. \tag{3}$$

  - In contrast, if machine i has not completed the processing of job in position $j-1$, we must estimate the completion time of this job. To compute this estimate, let us denote by k the position of the last scheduled job that is completed in machine i at time given by $a_1(j)$, i.e. $C_{i,k} \leq a_1(j)$. This is the last scheduled job for which we have all realized completion times for machines $1, 2, \ldots, m$. Let us define $\tilde{C}_{l,k} = C_{l,k}$ for $l = 1, \ldots, m$. For the subsequent jobs in position r ($r = k+1, \ldots, j$), their estimated completion times are:

* For the first machine, note that all jobs in positions $r = k+1, \ldots, j-1$ have completed their processing (otherwise job j cannot have completed its own processing), therefore:

$$\tilde{C}_{1,r} = C_{1,r} \quad \forall \; r = k+1, \ldots, j. \tag{4}$$

* For the remaining machines ($l = 2, \ldots, m$), the completion times of job in position r have to be estimated if their real completion times fall behind $a_1(j)$ (see Figure 2). Two cases can be distinguished: if $C_{l,r} \leq a_1(j)$ then job in position r has completed its processing time in machine l, therefore $\tilde{C}_{l,r} = C_{l,r}$. In contrast, if $C_{l,r} > a_1(j)$ then the job in position r has not completed its processing on machine l, but its completion time can be estimated using previous estimates of the completion times, i.e. $\max\{\tilde{C}_{l,r-1}, \tilde{C}_{l-1,r}\}+\tilde{p}_{lr}$. Note, however, that this latter expression may provide a value lower than $a_1(j)$ which does not not reflect the real situation, as we know that the processing time of this job has not completed at time $a_1(j)$. Therefore, we can assume that at least this value should be greater or equal than $a_1(j)$. Equation (5) summarises these options:

$$\tilde{C}_{l,r} = \begin{cases} C_{l,r} & \text{if} \quad C_{l,r} \leq a_1(j) \\ \max\{a_1(j); \max\{\tilde{C}_{l,r-1}; \tilde{C}_{l-1,r}\} + \tilde{p}_{lr}\} & \text{otherwise.} \end{cases} \tag{5}$$

$$l = 2, \ldots, i \qquad r = k+1, \ldots, j$$

Therefore,

$$a_i(j) = \tilde{C}_{i,j}.$$

In this manner, the $a_i(j)$ values can be computed upon the completion of a job in the first machine of the flowshop in order to run the rescheduling procedure described in the next section.

Procedure *IG(d, t_{limit}, T)*

Π_0 := Initial solution (random)

Perform a local search in the neighborhood of Π_0. Let Π be the best so-found solution

repeat

d jobs at random are extracted in Π and reinserted one by one in the position yielding the best (partial) makespan. Let us denote Π' the so-obtained sequence.

Perform a local search in the neighborhood of Π'. Let Π^* be the best so-found solution.

if $C_{max}(\Pi^*) < C_{max}(\Pi)$ **or** $random \leq e^{-\frac{C_{max}(\Pi^*)-C_{max}(\Pi)}{T}}$ **then**

Set $\Pi := \Pi^*$

until t_{limit} *is not exceeded*;

Return the best-so-found solution.

end

Figure 3: Main pseudocode of the Iterated Greedy algorithm

4.2 Procedures for Scheduling and Rescheduling

Different algorithms can be employed both to obtain the initial schedule as well as the successive (re)schedules. In the problem under consideration, this is equivalent to obtaining an initial sequence of jobs, and a modified sequence for the jobs not already in the shop subject to machine availability constraints, respectively.

Regarding the generation of the initial job sequence, it is obtained assuming a deterministic behaviour of the flowshop, i.e. $\tilde{p}_{ij}$ the mean processing times of the jobs on each machine are employed to solve the corresponding deterministic permutation flowshop scheduling problem.

With the generation of the initial sequence, we intend to analyse two different scenarios: In the first one, the initial sequence should be of excellent quality no matter the costs required to obtain it. In the second scenario, a random sequence would be used as initial sequence. In this manner, we can investigate the effect of rescheduling if the starting sequence is of very high quality, and also the influence of the initial sequence in the subsequent rescheduling, i.e. whether it pays off to develop an initial sequence, or similar results could be obtained without it.

For these scenarios with initial solutions of very high quality, we use the IG (Iterated Greedy) algorithm of Ruiz and Stützle (2007). The IG algorithm (see main pseudocode in Figure 3) iteratively performs a local search in the neighbourhood of the incumbent solution followed by a diversification mechanism consisting on removing a number of positions from the incumbent solution and reinserting them in random positions. This algorithm is known to be the best method for

the problem under consideration (see Fernandez-Viagas et al., 2017 for a computational evaluation of this method), so – particularly for the small instances in the testbed – we can be quite confident that the makespan is optimal or close to the optimal (deterministic) value. In order to ensure a good quality of the solution, we allow the algorithm a generous amount of computation time (30 seconds). The values of the parameters employed in the IG are the same as in Ruiz and Stützle (2007), but in our case the seed solution for the IG is random, so in this way the random solution could be interpreted as not performing any iteration of the IG algorithm. Note that our intention is to assess the merits of different strategies for rescheduling (including not to reschedule), therefore the IG algorithm is used as a proxy of the best initial solution that could be obtained, but we are not claiming that using different parameters and/or starting solutions could not provide better (deterministic) results. Furthermore, the processing times variability induces a degree of stochasticity that would blur these potential difference, in case they exist.

Regarding the development of a (possibly) modified sequence for the jobs not yet in the shop when any of the rescheduling policies described in Section 4.3 is triggered, the following steps are adopted:

Let us denote the initial (base) sequence of jobs by $\Pi := (\pi_1, \ldots, \pi_n)$. Every time job π_j completes its processing on the first machine:

1. An estimation of the machine availability is conducted according to the procedure described in Section 4.1. At this point, jobs $(\pi_1, \ldots, \pi_j)$ are already in the shop, so $a_i(j)$ denotes such estimation of the availability of the machines.

2. The problem of scheduling the remaining jobs $(\pi_{j+1}, \ldots, \pi_n)$ subject to the initial machine availability $a_i(j)$ has to be solved. Since this problem is also NP-hard (as it is a general case of the problem without the availability constraint), there are several options:

    - To use a very high-quality procedure to solve this problem, leaving aside the time to obtain the solution.
    - To use a (nearly) real-time procedure with a good –but not necessary of very high quality– performance.

When doing both choices we can take into account the results by Perez-Gonzalez and Framinan (2009), which indicate that the procedures for the $Fm||C_{max}$ problem also perform well for the availability-constrained version unless a_i are extremely different (i.e. several times the magnitude of the processing times of the jobs). Clearly, this is not our case as the increases of these unavailability periods should not exceed, on average, the mean processing times of the jobs. Therefore, we can adapt the IG algorithm to the problem under consideration, with the same parameters as in the generation of the initial sequence, and be quite confident on obtaining high-quality results for the problem with machine availability constraints. For the second option, we adapt the NEH heuristic (Nawaz et al., 1983), which is known to be the most efficient constructive heuristic for the unconstrained problem. In this manner, we can model the two options described above.

3. The first job of the new sequence obtained by some of the methods described in the above step enters in the first machine of the flowshop.

4.3 Rescheduling policies

The rescheduling procedure described in the previous section can be triggered every time a job is completed in the first machine of the shop, or upon the fulfilment of additional requirements. Perhaps the simplest way –and certainly the most popular in the literature reviewed in Section 2– is to perform a continuous rescheduling (CR) policy, i.e. to trigger the rescheduling procedure every time that a job finishes its processing in the first machine of the shop. However, on the one hand continuous rescheduling is very time consuming, as it has been already discussed. On the other hand, such frequent rescheduling may introduce some instability in the system so its performance may deteriorate. Therefore, we also propose two Event-Driven Rescheduling policies that are described in the next subsections.

4.3.1 Discrepancy-Based Policy

Clearly, if the variability of the processing times is minimal, then there would be no need to reschedule, as the actual completion times would be very similar to those expected from the base schedule.

In this regard, it seems intuitive to argue that the need for rescheduling increases as the discrepancy between the expected completion times and the actual processing times becomes higher. Therefore, our first proposal is a policy labelled Discrepancy-Based Rescheduling or DBR, which, for each job j which has just finished its processing on the first machine, it measures the relative discrepancy between the expected completion time of this job in the first machine and its actual completion time on this machine. If this relative discrepancy exceeds a certain value δ (a parameter of the policy), then the rescheduling procedure is triggered. More specifically, the procedure is triggered if the following condition holds:

$$\left| \frac{\tilde{C}_{1,j} - C_{1,j}}{\tilde{C}_{1,j}} \right| > \delta \tag{6}$$

It is clear that δ greatly influences the behaviour of this policy: for small δ values, this policy is expected to be similar to a CR policy, whereas for large δ values, it is expected to be similar to not to reschedule.

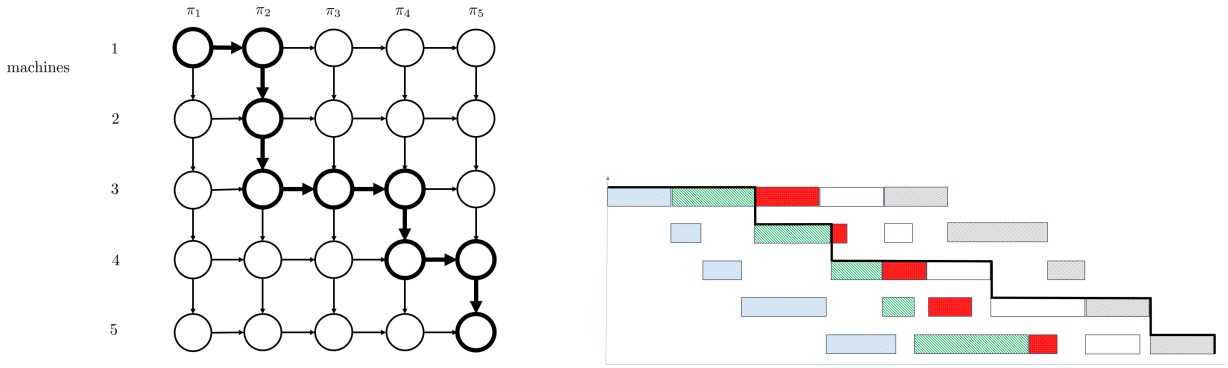
4.3.2 Critical Path Rescheduling Policy

The second rescheduling policy proposed in this paper is based on the concept of *critical path* of a flowshop (Nowicki and Smutnicki, 1996). Given a sequence $\Pi = (\pi_1, \ldots, \pi_n)$, a graph $G(\Pi) = (N, E)$ can be obtained by connecting $N = n \cdot m$ nodes (representing each operation of the n jobs on the m machines of the flowshop), each one of weight $\tilde{p}_{\pi_i,j}$. A set of arcs E connects each node (i, j) with nodes $(i + 1, j)$ and $(i, j + 1)$. The resulting graph is depicted in Figure 4a. In this graph, the critical path is the longest path to go from node $(1, 1)$ to node (n, m). Clearly, the length of the critical path is the (deterministic) makespan of the sequence Π, i.e.:

$$\tilde{C}_{max}(\Pi) = \sum_{\forall (i,j) \in CP(\Pi)} \tilde{p}_{i,j}$$

where $CP(\Pi)$ is the set of nodes in the critical path of sequence Π. The interpretation of the critical path in the Gantt chart is given in Figure 4b.

In a scenario with variable processing times it is clear that, if the realization of the processing times of a given operation changes the previous critical path, then the makespan of this sequence



(a) Graph representation and critical path (b) Gantt chart with critical path

Figure 4: Example of the critical path concept.

might be greatly altered with respect to that initially expected. In contrast, if the actual processing times of the current sequence do not change the initial critical path, then possibly there is no need to reschedule the remaining jobs, as possibly some left/right shifting will suffice. Therefore, our proposal is to use this alteration in the critical path as a mechanism to trigger the rescheduling.

More specifically, after the completion time of job j in the first machine, we check whether the actual completion times have modified the critical path with respect to that of the current sequence. If the resulting critical path is not the same, then the rescheduling procedure is triggered. This policy is labelled CPR (Critical Path Rescheduling) policy, and does not require any parameter.

5 Computational experience

In this section we describe the experiments carried out to respond to the research questions posed in Section 1. In Section 5.1 we discuss the design of the experiments, while in Section 5.2 we present the results of the computational experience.

5.1 Design of the Experiments

The different procedures described in Section 4 are applied to the testbed of Taillard (1993), that contains 120 flowshop scheduling instances with different numbers of jobs and machines. This testbed has become an standard for the evaluation of the effectiveness of flowshop scheduling techniques, and it is known to contain instances for which it is difficult to find their optimal values, as the

processing times in the instances have been chosen so that a lengthy tabu search procedure yields solutions which are relatively far from their lower bounds. Out of these 120 instances, the first 100 ones have been selected for our experimentation. Aside from the enormous computation times required for the simulation of the re-sequencing procedure in the biggest instances (see below), we have opted for this subset of instances in order to ensure that the IG procedure is able to find very good solutions within the allocated CPU time. Therefore, the following combinations have been tested: $(n, m) \in \{(20, 5), (20, 10), (20, 20), (50, 5), (50, 10), (50, 20), (100, 5), (100, 10), (100, 20), (200, 10)\}$, with 10 instances of each problem size.

Note that other testbeds for permutation flowshop problems exist, such as the VRF testbed by Vallada et al. (2015). However, this testbed is relatively new and it seems reasonable to think that the best-known values of these instances are not so close to their optima as those in Taillard's testbed, which, in many cases, are known to be in fact optimal. Since we will use these deterministic best-known values to compute the quality of the approaches (see Section 5.2), we believe that using these, in principle, tighter upper bounds would produce more consistent results.

The processing times in the instances are assumed to follow a log normal distribution, which has two parameters: mean μ and standard deviation σ. This distribution has been widely employed to model stochastic processing times (see e.g. Baker and Trietsch, 2011, Baker and Altheimer, 2012, or Framinan and Perez-Gonzalez, 2015). Furthermore, we can easily control the variability of the processing times by setting different levels of coefficient of variation $cv = \frac{\sigma}{\mu}$. More specifically, we set $cv \in \{0.5, 1, 1.5\}$, which are assumed to represent a low, medium and large variability respectively in manufacturing shop floors (see Hopp and Spearman, 2008). In this manner, the processing times given in Taillard's testbed are interpreted as the mean value of these processing times ($\mu = \tilde{p}_{ij}$) and σ_{ij} the corresponding standard deviation is obtained once cv is given, i.e. $\sigma_{ij} = \tilde{p}_{ij} \cdot cv$.

For each instance in the testbed, different rescheduling strategies have been tested. These are labelled using the format $(B, \{RP, RA\})$, where B indicates how the base schedule is obtained, RP indicates the rescheduling policy –i.e. when the rescheduling algorithm is triggered–, and RA indicates the rescheduling algorithm employed. Using this notation, the following options have been tested:

- $(IG, -)$: The IG procedure described in Section 4.2 has been used to develop an initial

schedule, which has not been subsequently modified.

- $(IG, \{CR, IG\})$: The IG procedure has been used to develop an initial schedule, and the CR rescheduling procedure described in Section 4.3 using the IG algorithm is triggered every time a job has finished its processing in the first machine.

- $(Rand, \{CR, IG\})$: A random sequence has been used as initial schedule, and the CR rescheduling procedure using the IG algorithm is triggered every time a job has finished its processing in the first machine.

- $(IG, \{CR, NEH\})$: The IG procedure has been used to develop an initial schedule, and the CR rescheduling procedure using the NEH heuristic is triggered every time a job has finished its processing in the first machine.

- $(IG, \{DBR(\delta), IG\})$: The IG procedure has been used to develop an initial schedule, and the DBR rescheduling procedure described in Section 4.3.1 using the IG algorithm is triggered if the conditions in Equation (6) have been fulfilled when a job has finished its processing in the first machine, i.e. every time that the discrepancy has been greater than $\delta \in \{0.25, 0.5\}$. For $\delta > 0.5$, preliminary experiments have detected that the rescheduling procedure was triggered for all jobs, therefore it was equivalent to a CR policy.

- $(IG, \{CPR, IG\})$: The IG procedure has been used to develop an initial schedule, and the CPR rescheduling procedure described in Section 4.3.2 using the IG algorithm is triggered if the new information implies a change on the critical path of the current schedule.

The computational effort carried out involves solving 30 replications of 100 problem sizes for each of the 7 options described above. Three out of these 7 options involve invoking of the IG algorithm $n-1$ times with a time limit of 30 seconds for each coefficient of variation, which makes a total equivalent computation time of 22.18 days only for the three continuous rescheduling options. In total, the experimentation has taken the equivalent of several months of computations.

n	m	$(IG, -)$	$(IG, \{CR, IG\})$	$(Rand, \{CR, IG\})$	$(IG, \{CR, NEH\})$	$(IG, \{DBR(0,25), IG\})$	$(IG, \{DBR(0,5), IG\})$	$(IG, \{CPR, IG\})$
20	5	21.379	19.671	23.263	20.475	20.880	22.651	19.368
20	10	27.296	26.669	30.791	26.396	26.153	26.467	25.899
20	20	25.144	25.130	27.250	26.399	26.033	24.973	24.489
50	5	17.066	15.218	16.477	15.496	15.195	15.527	13.794
50	10	25.566	24.433	27.024	26.382	24.550	24.117	23.810
50	20	29.551	28.315	29.804	31.387	27.792	28.042	28.069
100	5	13.352	12.084	12.668	12.234	12.210	12.146	11.893
100	10	20.307	18.468	20.449	19.900	18.546	18.088	17.381
100	20	26.907	25.531	27.027	28.026	25.449	24.175	23.915
200	10	16.209	14.892	15.246	15.596	14.862	14.319	13.101
Average		**22.952**	**21.724**	**23.862**	**22.966**	**21.868**	**21.798**	**20.958**

Table 1: Average RPD for $cv = 0.5$

n	m	$(IG, -)$	$(IG, \{CR, IG\})$	$(Rand, \{CR, IG\})$	$(IG, \{CR, NEH\})$	$(IG, \{DBR(0,25), IG\})$	$(IG, \{DBR(0,5), IG\})$	$(IG, \{CPR, IG\})$
20	5	43.532	41.847	44.694	44.723	41.579	41.216	41.178
20	10	56.218	54.068	61.854	56.257	56.230	57.741	55.778
20	20	58.581	58.697	61.698	57.995	58.031	59.487	57.994
50	5	34.430	33.818	34.346	33.160	33.549	33.088	32.623
50	10	52.806	52.346	52.290	53.082	51.766	52.501	52.105
50	20	62.462	60.403	64.844	65.048	62.077	62.144	61.432
100	5	26.828	25.543	26.266	26.733	26.311	27.708	25.211
100	10	41.359	39.779	42.045	41.171	41.239	40.747	38.105
100	20	57.289	56.668	56.783	58.786	56.088	54.396	54.649
200	10	33.607	31.766	32.636	31.631	32.613	31.571	30.053
Average		**48.167**	**47.019**	**49.424**	**48.551**	**47.430**	**47.670**	**46.564**

Table 2: Average RPD for $cv = 1.0$

n	m	$(IG, -)$	$(IG, \{CR, IG\})$	$(Rand, \{CR, IG\})$	$(IG, \{CR, NEH\})$	$(IG, \{DBR(0,25), IG\})$	$(IG, \{DBR(0,5), IG\})$	$(IG, \{CPR, IG\})$
20	5	65.460	65.073	65.835	62.049	58.928	67.254	59.371
20	10	77.959	77.775	82.043	83.921	83.919	78.731	81.644
20	20	86.766	87.920	93.570	87.619	91.258	90.188	87.475
50	5	47.192	46.228	45.964	49.411	47.741	49.529	45.885
50	10	80.147	76.678	81.362	74.654	76.875	75.806	76.062
50	20	94.308	95.021	99.883	98.629	100.453	95.496	95.954
100	5	40.061	38.407	40.332	40.101	40.019	41.588	38.461
100	10	60.820	60.981	62.597	62.077	61.562	62.493	60.013
100	20	88.549	86.243	85.850	87.753	85.264	85.600	84.904
200	10	49.350	48.524	48.445	49.541	48.973	50.122	45.705
Average		**71.251**	**70.481**	**73.048**	**71.802**	**71.780**	**71.854**	**69.974**

Table 3: Average RPD for $cv = 1.5$

5.2 Results

The results for each value of cv are summarised in Tables 1, 2 and 3 in terms of the Average Relative Percentage Deviation (RPD), where the RPD of instance i for a procedure p is computed as:

$$RPD(i,p) = \frac{C_{max}(i,p) - C^*_{max}(i)}{C^*_{max}(i)} \cdot 100 \tag{7}$$

where $C_{max}(i,p)$ is the makespan obtained by applying procedure p on instance i, and C^*_{max} is the best-known value for instance i assuming deterministic processing times.

In order to compute the savings in triggering the rescheduling procedures achieved by the different event-driven rescheduling policies –i.e. $(IG, \{DBR(0,25), IG\})$, $(IG, \{DBR(0,5), IG\})$ and $(IG, \{CPR, IG\})$–, we also measure the relative reduction in the number of times (RRN) that the rescheduling processes is triggered for a given rescheduling policy with respect to that of the continuous rescheduling. More specifically, for each instance i and a rescheduling policy p we compute:

$$RRN(i,p) = \frac{NCR_i - NR_{i,p}}{NCR_i} \tag{8}$$

where NCR_i is the number of times that a continuous rescheduling policy would trigger the rescheduling process (i.e. the number of jobs in instance i minus one), and $NR(i,p)$ is the number of times that rescheduling policy p triggers the rescheduling process. The average RNN values for each problem size are shown in Table 4.

In order to check the statistical significance of the results, the 95% confidence intervals for the different coefficient of variations are shown in Figures 5, 6, and 7 respectively. Finally, the results of Tukey HSD are shown in Tables 5, 6, and 7.

An ANOVA conducted on all the results establishes that the most significant factor is the coefficient of variation, therefore it makes sense to divide the discussion of the results along the different values of cv. For scenarios with low variability ($cv = 0.5$), it can be seen that there are advantages on rescheduling as compared with no rescheduling as long as the base schedule is a good one and we use an excellent rescheduling algorithm (in our case, IG). This can be clearly seen in Table 1 and in Figure 5 by checking that there are statistically significant differences be-

Problem size		$cv = 0.5$			$cv = 1.0$			$cv = 1.5$		
n	m	$\delta = 0.25$	$\delta = 0.5$	CPR	$\delta = 0.25$	$\delta = 0.5$	CPR	$\delta = 0.25$	$\delta = 0.5$	CPR
20	5	0.487	0.937	0.905	0.379	0.898	0.814	0.376	0.860	0.737
20	10	0.560	0.993	0.761	0.463	0.958	0.657	0.394	0.908	0.595
20	20	0.895	1.000	0.794	0.745	0.993	0.696	0.670	0.975	0.599
50	5	0.429	0.901	0.948	0.305	0.844	0.882	0.254	0.752	0.826
50	10	0.274	0.803	0.908	0.123	0.701	0.854	0.101	0.604	0.821
50	20	0.384	0.987	0.847	0.205	0.965	0.801	0.117	0.883	0.772
100	5	0.417	0.823	0.969	0.283	0.773	0.939	0.220	0.704	0.909
100	10	0.280	0.757	0.953	0.086	0.679	0.918	0.057	0.571	0.895
100	20	0.296	0.804	0.926	0.084	0.752	0.899	0.028	0.649	0.876
200	10	0.283	0.676	0.982	0.119	0.623	0.957	0.046	0.525	0.947
Average		**0.447**	**0.889**	**0.890**	**0.297**	**0.840**	**0.829**	**0.246**	**0.767**	**0.781**

Table 4: Average RRN for the event-driven policies and the different coefficients of variations.

tween $(IG, -)$ and any of the rescheduling policies that use IG for both schedule generation and rescheduling. $(IG, -)$ obtains sequences that are, on average, around 10% worse than those obtained by the $(IG, \{CPR, IG\})$ policy (the best-performing rescheduling policy) and 3.6% worse than those obtained by the $(IG, \{CR, IG\})$ policy (the worst-performing rescheduling policy). If these differences are measured using the best of the tested policies as $C^*_{max}(i)$, this figure is about 27% for $(IG, \{CPR, IG\})$ and about 10% for $(IG, \{CR, IG\})$. This speaks for substantial performance differences between these cases. The results also show that these potential rescheduling advantages blur if the base schedule is not excellent (see $(Rand, \{CR, IG\})$ in Figure 5) or if the rescheduling algorithm is not excellent, see $(IG, \{CR, NEH\})$.

Among the rescheduling policies that can be implemented, the results also show that it is better to implement event-driven rescheduling policies as compared to continuous rescheduling. Particularly, the proposed Critical-Path Rescheduling policy outperforms the rest of the policies, and this difference is statistically significant, as it can be seen in the Tukey HSD data in Table 5. Furthermore, it can be seen in Table 4 that the CPR policy requires a very small number of reschedules (a saving of around 90% with respect to CR) and does not need setting any parameter, which represents another advantage as compared to DBR policies.

When analysing the trade-off between a good base schedule followed by no rescheduling, and a bad base schedule followed by a good rescheduling algorithm (i.e. $(IG, -)$ vs. $(Rand, \{CR, IG\})$ in Figure 5), the second option is better, not only on average but also in statistical terms. This fact

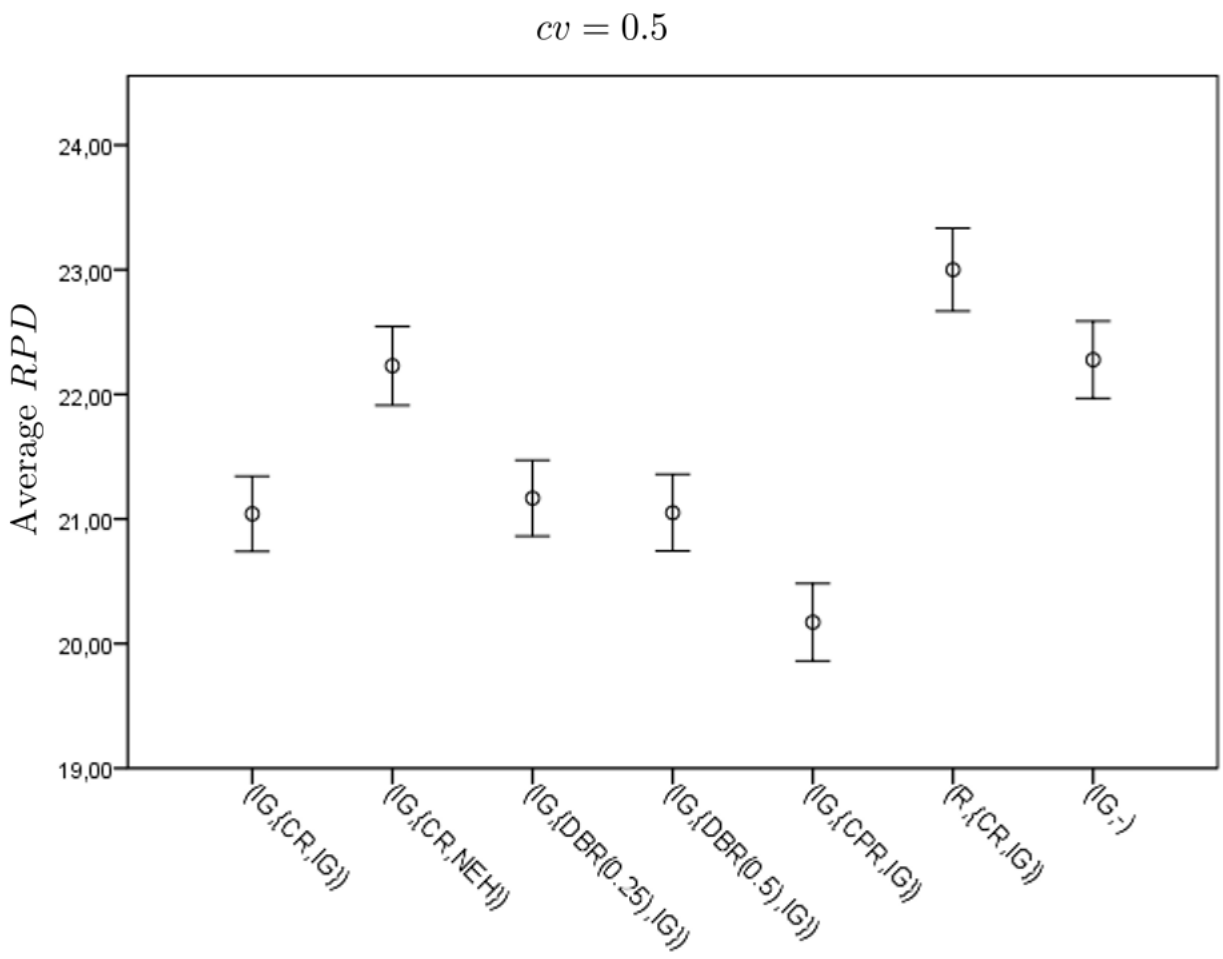


Figure 5: Mean and 95% CI for $cv = 0.5$

speaks for the relative robustness of deterministic scheduling in front of low variability of processing times.

For an scenario with medium variability ($cv = 1.0$), Figure 6 shows that there are advantages on rescheduling as compared with no rescheduling as long as the base schedule is a good one and we use a combination of an excellent rescheduling algorithm (in our case, IG) and a suitable rescheduling policy (Critical Path). Otherwise, there may be advantages in rescheduling with respect to the average values (see Table 2), but they are not statistically significant. The difference between $(IG, \{CPR, IG\})$ and $(IG, -)$ for this case are around 5% with respect to deterministic scheduling, and around 8% with respect to the best tested option. Therefore, although the differences between the strategies shrink, there are still substantial differences between them.

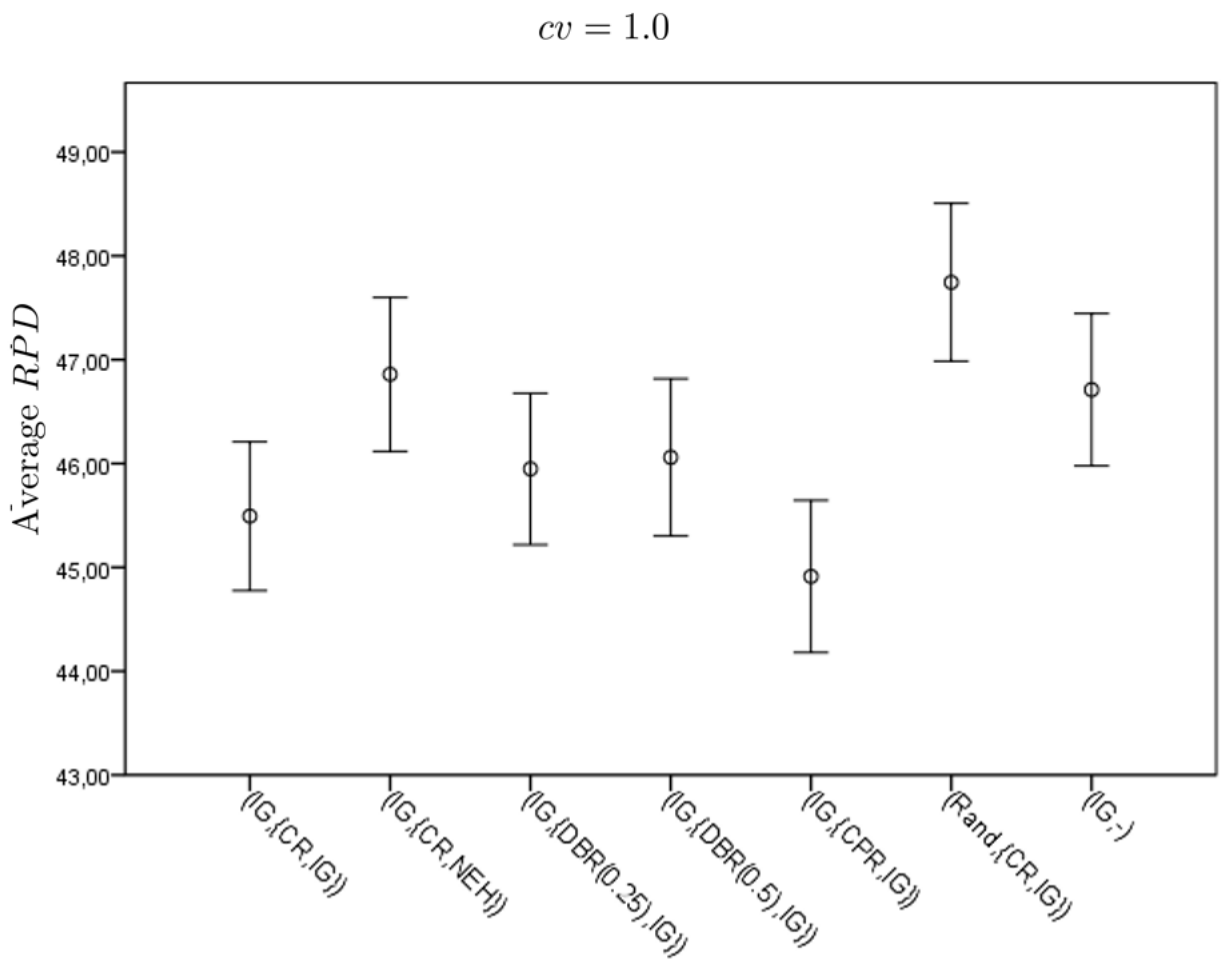


Figure 6: Mean and 95% CI for $cv = 1.0$

Among the rescheduling policies, $(IG, \{CPR, IG\})$ is still the best one regarding the average values, and also taking into account the smaller number of times that the rescheduling procedure is triggered, see Table 4. Finally, when analysing the trade-off between a good base schedule followed by no rescheduling, and a bad base schedule followed combined with a good rescheduling algorithm (i.e. $(IG, -)$ vs. $(Rand, \{CR, IG\}_)$ in Figure 6), the second option is still better with respect to the average quality of the solutions, although there are not statistically significant differences between them, see Table 6.

Finally, for an scenario with high variability ($cv = 1.5$), there are no statistically significant differences between the different options of rescheduling and not rescheduling when all problem sizes are aggregated. However, the analysis of the larger instances ($n = 200$) reveals that there are statistical differences between the methods (see Figure 8). This is indicating that, even for the

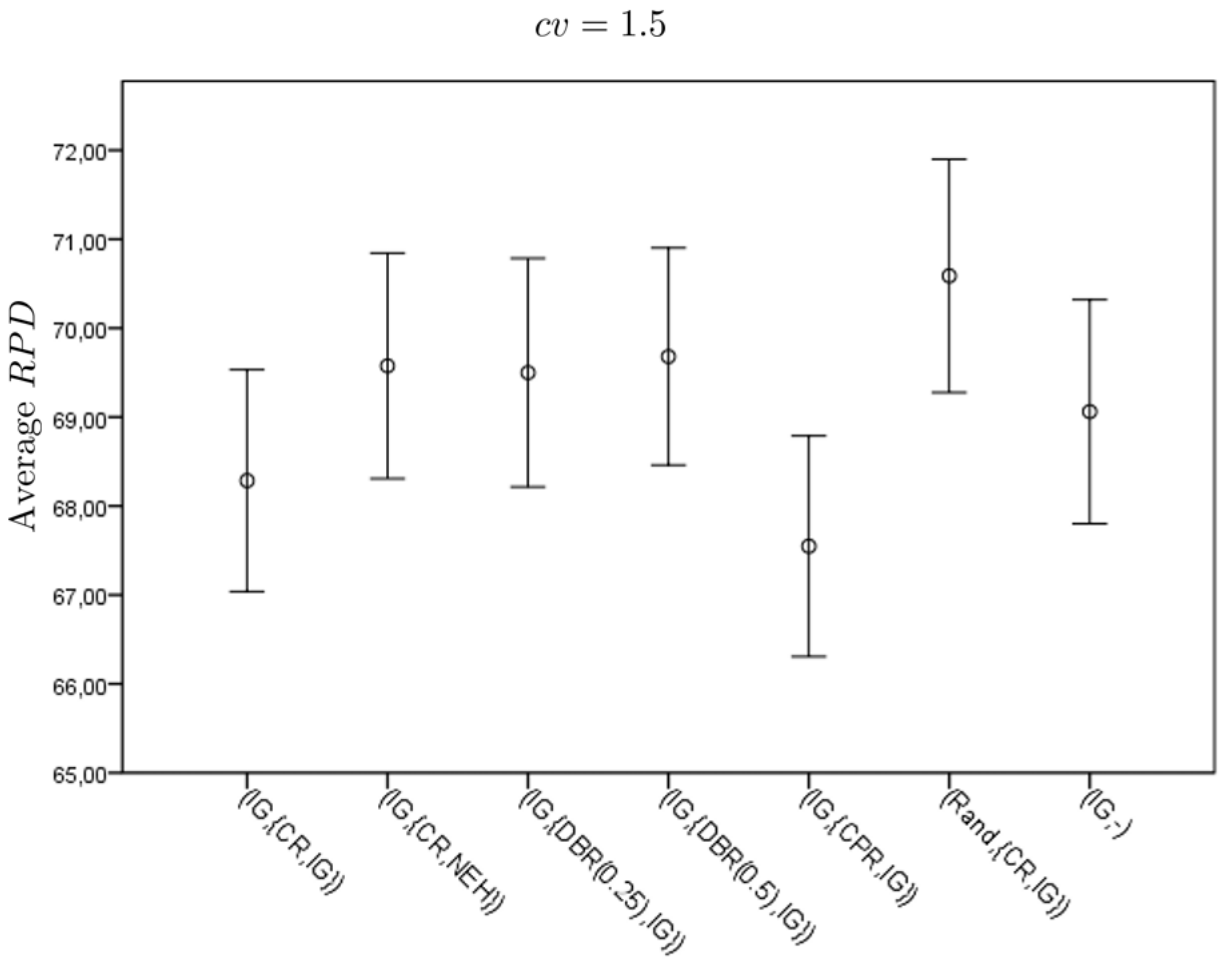


Figure 7: Mean and 95% CI for $cv = 1.5$

scenarios with high variability, rescheduling –particularly if triggered using the Critical Path Policy proposed in this paper– also pays off.

6 Conclusions

This paper explores the value of using real-time data for conducting rescheduling decisions in a permutation flowshop where the processing times are assumed to be stochastic. To do so, we compare the expected performance of well-known rescheduling strategies (including not rescheduling, or performing a continuous rescheduling policy), and propose two novel event-driven rescheduling policies, with triggering mechanisms based on the concepts of discrepancy with respect to the expected com-

		Subsets			
Method	Sample size	1	2	3	4
$(IG, \{CPR, IG\})$	3000	20.1719			
$(IG, \{CR, IG\})$	3000		21.041139		
$(IG, \{DBR(0.5), IG\})$	3000		21.050505		
$(IG, \{DBR(0.25), IG\})$	3000		21.167193		
$(IG, \{CR, NEH\})$	3000			22.22911	
$(IG, -)$	3000			22.277568	
$(Rand.\{CR, IG\})$	3000				23.000002
Sig.		1	0.991	1	1
Mean squared error	45.162				
Alpha	0.05				

Table 5: Tukey HSD for $cv = 0.5$.

		Subsets			
Method	Sample size	1	2	3	4
$(IG, \{CPR, IG\})$	3000	44.912831			
$(IG, \{CR, IG\})$	3000	45.493531	45.493531		
$(IG, \{DBR(0.25), IG\})$	3000	45.948307	45.948307	45.948307	
$(IG, \{DBR(0.5), IG\})$	3000	46.059851	46.059851	46.059851	
$(IG, -)$	3000		46.711231	46.711231	46.711231
$(IG, \{CR, NEH\})$	3000			46.858676	46.858676
$(Rand, \{CR, IG\})$	3000				47.745502
Sig.		0.106	0.069	0.342	0.196
Mean squared error	277.184				
Alpha					

Table 6: Tukey HSD for $cv = 1.0$.

		Subsets	
Method	Sample size	1	2
$(IG, \{CPR, IG\})$	3000	67.547393	
$(IG, \{CR, IG\})$	3000	68.284935	
$(IG, -)$	3000	69.061143	69.061143
$(IG, \{DBR(0.25), IG\})$	3000	69.499301	69.499301
$(IG, \{CR, NEH\})$	3000	69.575527	69.575527
$(IG, \{DBR(0.5), IG\})$	3000	69.68064	69.68064
$(Rand, \{CR, IG\})$	3000		70.588082
Sig.		0.082	0.426
Mean squared error	890.009		
Alpha			

Table 7: Tukey HSD for $cv = 1.5$.

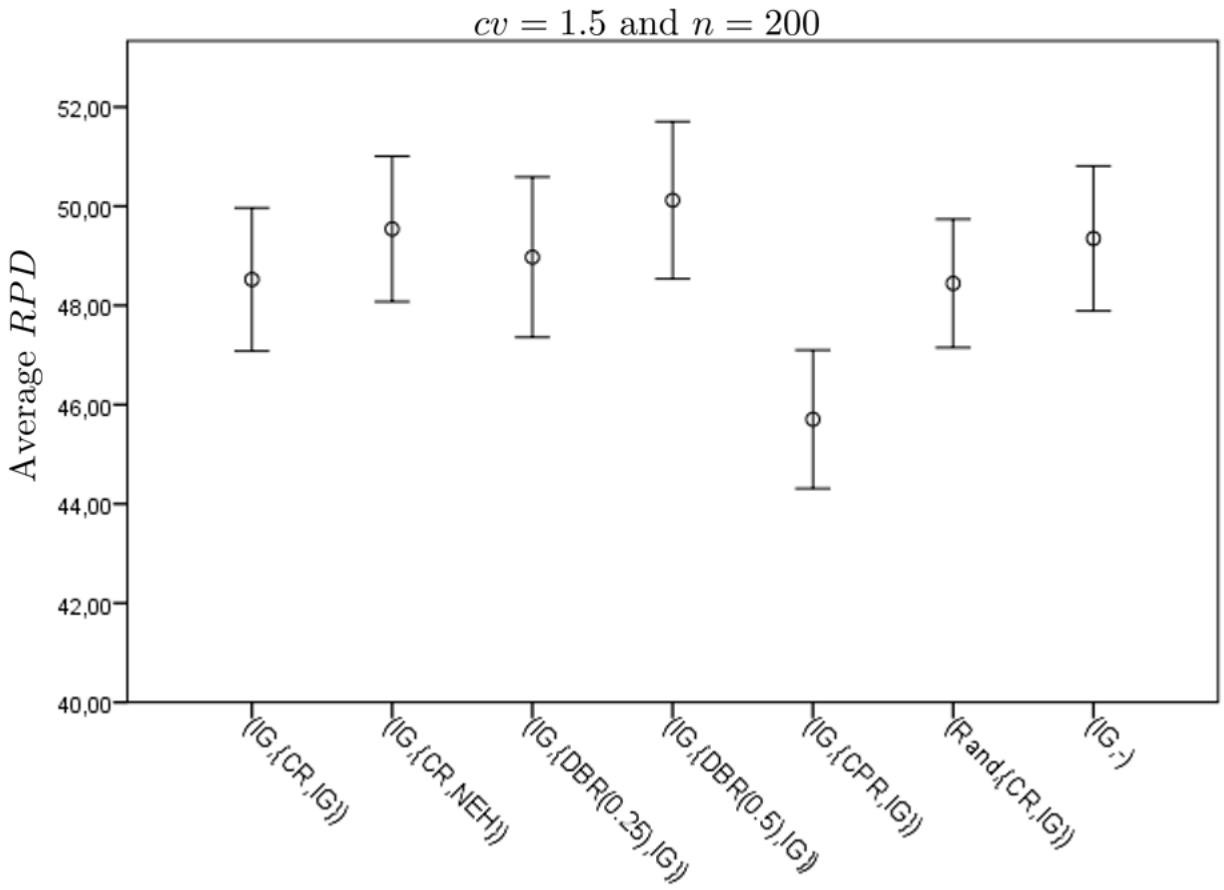


Figure 8: Mean and 95% CI for $cv = 1.5$ and $n = 200$.

pletion times, and on the disruption of the critical path respectively. All these policies are tested in scenarios with different variability of processing times, and also analysed using different procedures to generate the initial schedule and to perform the rescheduling of the remaining jobs. The computational results show that, when variability of the processing times is low to medium (a coefficient of variation between 0.5 and 1 in our research), rescheduling improves the performance in the shop floor, reducing the ARPD values with respect to the option of not to reschedule. In contrast, when the processing time variability is very high (a coefficient of variation of 1.5 in our experiments), there are no statistically significant differences between rescheduling and not rescheduling, even if the mean and the standard deviation of ARPD are lower in the first case. The results also make clear that it is critical to start with a good base schedule, and that event-driven policies are competitive with continuous rescheduling while requiring a much smaller number of reschedules. Particularly,

the proposed scheduling policy based on the critical path concept obtains the best results for all scenarios tested. Finally, the expected performance of re-sequencing does not seem to be affected by the number of machines in the shop, at least within the limits of our experiments. However, it is affected by the number of jobs, indicating a higher benefit from re-sequencing when the number of jobs is high.

The results obtained have a number of implications for schedulers and decision makers. First, our research is motivated by the technological advances in IT in the manufacturing domain –such as Industry 4.0 or SmartFactory–, which make real-time information on shop floor status instantly available for decision makers. However, there may be substantial costs in the deployment of this infrastructure. Our research shows that, at least within the limitations of our study, that investments in gathering real-time data at the scheduling level may pay off depending basically on the variability of the shop floor and, to a lesser extent, on the number of jobs to be scheduled. While there is a range of low/middle variability where this information can be used to provide a more efficient schedule, this is not the case for environments with very high variability.

The results also emphasize the importance of employing a very good base schedule, as poor-quality schedules cannot be repaired by high-quality rescheduling. In our research we have assumed that this base schedule is deterministic, but it would be also interesting to test the performance of base schedules obtained using (static) stochastic scheduling, or robust schedules.

The importance of the rescheduling procedure is also stressed: Contrary to a perhaps common intuition based on the results available for single-stage scheduling, continuous rescheduling does not necessarily is the best rescheduling strategy for flowshops. Instead, event-driven scheduling guided by the disruption of the initial schedule may yield better results. Particularly, rescheduling triggered by the concept of critical path has shown to provide the best overall results, being a policy extremely simple to implement (i.e. no parameter setting is required).

Finally, it has to be noted that stochastic modelling is sometimes used to capture the fact that there are no accurate estimations of the processing times. In view of the results obtained when the variability is very high, it would be advisable to devote time and resources to provide accurate processing times before deploying an (optimisation-based) scheduling and/or rescheduling system, as these may not reap the expected benefits in the case of highly variable processing times.

A number of possible avenues worth of research can be derived from our research. First, our study is limited to a specific layout and objective function. It will be interesting to check whether the conclusions obtained also extend to different settings and objectives. Particularly, due date related objectives seem to be very interesting, as the objective function would be a combination of stochastic (processing times) and deterministic (due dates) data, so it may result in a different behaviour than the one obtained for the makespan. Extending the work to other multi-stage settings such as the hybrid flowshop could be also interesting. Second, in view of the extensive CPU requirements of the scheduling and rescheduling algorithms employed – which may not be affordable in some manufacturing scenarios–, it would be interesting to develop new fast heuristics for rescheduling, as those currently available (borrowed from the deterministic flowshop scheduling case) do not perform well under the scenario with processing time variability. Third, in some manufacturing scenarios, the permutation constraint does not have to be necessarily enforced. Therefore, it could be interesting to analyse whether there are advantages in rescheduling without the permutation constraint, as the results for maximum tardiness in Swaminathan et al. (2007) seem to attest. Note that this option would provide more decision points for re-scheduling (i.e. $n \cdot m - 1$ as compared to $n - 1$ in our experiments) and a more extensive use of available data can be done.

References

Akturk, M. S. and Gorgulu, E. (1999). Match-up scheduling under a machine breakdown. *European Journal of Operational Research*, 112(1):81–97.

Aytug, H., Lawley, M., McKay, K., Mohan, S., and Uzsoy, R. (2005). Executing production schedules in the face of uncertainties: A review and some future directions. *European Journal of Operational Research*, 161(1):86–110.

Baker, K. and Altheimer, D. (2012). Heuristic solution methods for the stochastic flow shop problem. *European Journal of Operational Research*, 216(1):172–177.

Baker, K. and Trietsch, D. (2011). Three heuristic procedures for the stochastic, two-machine flow shop problem. *Journal of Scheduling*, 14(5):445–454.

Chen, C.-C., Chen, C.-L., Ciou, C.-Y., and Liu, J.-X. (2016). Communication scheduling scheme based on big-data regression analysis and genetic algorithm for cyber-physical factory automation. In *2016 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2016 - Conference Proceedings*, pages 2603–2608.

Church, L. and Uzsoy, R. (1992). Analysis of periodic and event-driven rescheduling policies in dynamic shops. *International Journal of Computer Integrated Manufacturing*, 5(3):153–163.

El-Bouri, A. (2012). A cooperative dispatching approach for minimizing mean tardiness in a dynamic flowshop. *Computers and Operations Research*, 39(7):1305–1314.

El Bouri, A. and Amin, G. R. (2015). A combined OWA-DEA method for dispatching rule selection. *Computers and Industrial Engineering*, 88:470–478.

Fernandez-Viagas, V., Ruiz, R., and Framinan, J. (2017). A new vision of approximate methods for the permutation flowshop to minimise makespan: State-of-the-art and computational evaluation. *European Journal of Operational Research*, 257(3):707–721.

Framinan, J., Leisten, R., and Ruiz, R. (2014). *Manufacturing Scheduling Systems: An Integrated View of Models, Methods, and Tools*. Springer.

Framinan, J. and Perez-Gonzalez, P. (2015). On heuristic solutions for the stochastic flowshop scheduling problem. *European Journal of Operational Research*, 246(2):413–420.

González-Neira, E., Montoya-Torres, J., and Barrera, D. (2017). Flow-shop scheduling problem under uncertainties: Review and trends. *International Journal of Industrial Engineering Computations*, 8 (4):399–426.

Hao, X. and Lin, L. (2010). Job shop rescheduling by using multi-objective genetic algorithm. In *Proceedings of the 40th International Conference on Computers and Industrial Engineering*.

Heger, J., Branke, J., Hildebrandt, T., and Scholz-Reiter, B. (2016). Dynamic adjustment of dispatching rule parameters in flow shops with sequence-dependent set-up times. *International Journal of Production Research*, 54(22):6812–6824.

Hopp, W. and Spearman, M. (2008). *Factory Physics (Third Edition)*. Irwin, Chicago.

Hozak, K. and Hill, J. A. (2009). Issues and opportunities regarding replanning and rescheduling frequencies. *International Journal of Production Research*, 47(18):4955–4970.

Katragjini, K., Vallada, E., and Ruiz, R. (2013). Flow shop rescheduling under different types of disruption. *International Journal of Production Research*, 51(3):780–797.

Liu, F., Wang, S., Hong, Y., and Yue, X. (2017a). On the robust and stable flowshop scheduling under stochastic and dynamic disruptions. *IEEE Transactions on Engineering Management*, 64(4):539–553.

Liu, W., Jin, Y., and Price, M. (2017b). New scheduling algorithms and digital tool for dynamic permutation flowshop with newly arrived order. *International Journal of Production Research*, 55(11):3234–3248.

Liu, X.-P., Liu, F., and Wang, J.-J. (2016). An enhanced memetic algorithm for combinational disruption management in sequence-dependent permutation flowshop. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9771:548–559.

Nawaz, M., Enscore Jr., E., and Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, 11(1):91–95.

Nowicki, E. and Smutnicki, C. (1996). A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1):160–175.

Ovacik, I. and Uzsoy, R. (1994). Exploiting shop floor status information to schedule complex job shops. *Journal of Manufacturing Systems*, 13(2):73–84.

Perez-Gonzalez, P. and Framinan, J. (2009). Scheduling permutation flowshops with initial availability constraint: Analysis of solutions and constructive heuristics. *Computers and Operations Research*, 36(10):2866–2876.

Pfeiffer, A., Kádár, B., and Monostori, L. (2007). Stability-oriented evaluation of rescheduling strategies, by using simulation. *Computers in Industry*, 58(7):630–643.

Rahman, H., Sarker, R., and Essam, D. (2013). Permutation flow shop scheduling with dynamic job order arrival. In *2013 IEEE Conference on Cybernetics and Intelligent Systems (CIS)*, pages 30–35.

Rahmani, D. and Heydari, M. (2014). Robust and stable flow shop scheduling with unexpected arrivals of new jobs and uncertain processing times. *Journal of Manufacturing Systems*, 33(1):84–92.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Sabuncuoglu, I. and Kizilisik, O. B. (2003). Reactive scheduling in a dynamic and stochastic fms environment. *International Journal of Production Research*, 41(17):4211–4231.

Swaminathan, R., Pfund, M., Fowler, J., Mason, S., and Keha, A. (2007). Impact of permutation enforcement when minimizing total weighted tardiness in dynamic flowshops with uncertain processing times. *Computers and Operations Research*, 34(10):3055–3068.

Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285.

Vallada, E., Ruiz, R., and Framinan, J. (2015). New hard benchmark for flowshop scheduling problems minimising makespan. *European Journal of Operational Research*, 240:666–677.

Vieira, G. E., Herrmann, J. W., and Lin, E. (2003). Rescheduling manufacturing systems: A framework of strategies, policies, and methods. *Journal of Scheduling*, 6(1):39–62.

Waschneck, B., Altenmüller, T., Bauernhansl, T., and Kyek, A. (2017). Production scheduling in complex job shops from an industrie 4.0 perspective: A review and challenges in the semiconductor industry. In *CEUR Workshop Proceedings*.