Simulation Modelling Practice and Theory 86 (2018) 55–71

Contents lists available at ScienceDirect

Simulation Modelling Practice and Theory

journal homepage: www.elsevier.com/locate/simpat

A simheuristic algorithm to set up starting times in the stochastic parallel flowshop problem

Sara Hatami*,[a], Laura Calvet[a,b], Victor Fernández-Viagas[c], José M Framiñán[c], Angel A. Juan[a,b]

[a] IN3 – Computer Science Department, Open University of Catalonia, Barcelona 08018, Spain
[b] Euncet Business School, Terrassa 08225, Spain
[c] Industrial Management, School of Engineering, University of Seville, Descubrimientos Ave., Seville E41092, Spain

ARTICLE INFO

Keywords:
Parallel scheduling problem
Stochastic scheduling problem
Simheuristics
Metaheuristics
Combinatorial optimization

ABSTRACT

This paper addresses the parallel flowshop scheduling problem with stochastic processing times, where a product composed of several components has to be finished at a particular moment. These components are processed in independent parallel factories, and each factory can be modeled as a permutation flowshop. The processing time of each operation at each factory is a random variable following a given probability distribution. The aim is to find the robust starting time of the operations at each factory in a way that all the components of the product are completed on a given deadline with a user-defined probability. A simheuristic algorithm is proposed in order to minimize each of the following key performance indicators: *(i)* the makespan in the deterministic version; and *(ii)* the expected makespan or a makespan percentile in the stochastic version. A set of computational experiments are carried out to illustrate the performance of the proposed methodology by comparing the outputs under different levels of stochasticity.

1. Introduction

Nowadays, the manufacturing industry faces important challenges, including: fierce competitiveness, short product life cycles, increasing in the speed of product innovation, high product variety and quality, rising customer expectations, etc. Industries need to find proper strategies to cope with these challenges and remain successful in the market. One of these strategies is the use of distributed manufacturing systems [44], with contrasted benefits in terms of higher product quality, lower production costs, and fewer management risks [11,37,59]. These systems consist of several production entities which operate in parallel.

In distributed or parallel manufacturing systems, the individual strengths of each entity are joined to achieve a common goal of the entire system, so the complexity of the manufacturing process is shared among different production entities. This results in reasonable levels of risk and cost, as well as in new market opportunities. In contrast, single manufacturing centers are often not able to produce a diversity of products within reasonable cost levels due to a lack of technology and skills [55,61]. As a result, single manufacturing centers are less and less frequent, while distributed, parallel, or multi-plant manufacturing systems are increasing in popularity [44,45].

On the one hand, constructing collaborative manufacturing systems helps industries to face market global challenges in an efficient way. On the other hand, the optimization of these systems is more complicated. This optimization usually has a significant

* Corresponding author.
E-mail address: shatami@uoc.edu (S. Hatami).

https://doi.org/10.1016/j.simpat.2018.04.005
Received 14 October 2017; Received in revised form 26 February 2018; Accepted 26 April 2018
Available online 27 April 2018
1569-190X/ © 2018 Elsevier B.V. All rights reserved.

effect on production performance. The production operations (including the scheduling of each entity) of these systems are organized in such a way that the needs of the entire system are covered and its objectives accomplished. The optimization of these systems has received considerable attention from practitioners and the research community in recent years.

A well-established problem in the context of parallel manufacturing systems is the so-called distributed permutation flowshop scheduling problem (DPFSP) [45]. Here, a set of distributed factories with flowshop configurations are considered. The factories are required to generate a product composed of various jobs/components. Each factory has to process a certain number of jobs, and all of them have to be completed on or before a given deadline. Typically, the DPFSP involves two decisions: *(i)* assigning each job to a factory, and *(ii)* determining a job sequence for each factory.

The classical DPFSP assumes a static environment and deterministic processing times to simplify the problem. However, real-world manufacturing systems are dynamic and often exposed to uncertainties and unforeseen events, such as machine breakdowns, changes in the due date, unavailability of operators, lack of materials, etc. [52].

Our paper addresses the stochastic version of the parallel flowshop scheduling problem (SPFSP). We assume that the components have already been assigned to the factories, and deal with job sequencing within each factory. In addition, the processing time of each operation in each factory is a random variable, which approaches our model to the real-life system. The objective is to find a robust starting time for each flowshop factory. The job sequence of each factory should start to process at the latest possible time, while completing all jobs on or before a given deadline. Since stochastic processing times are considered, we will only be able to guarantee that the product is finished by then with a given probability. Notice that this probability will depend on the probability of each factory to finish its part on time.

In our model, unforeseen events related to the lack of machine reliability, lack of raw materials, etc. are captured by making the processing times to be random variables. It is widely known in literature [30] that machine breakdowns, planned outages, or re-manufacturing constitute a source of processing times variability. Therefore, it is a usual practice to model these different causes as processing times variability.

The problem under consideration can be also related to the permutation flowshop scheduling problem with stochastic times (PFSPST). The literature on this problem is not extensive, especially when compared with the permutation flowshop scheduling problem (PFSP) [16,17,31,40]. However, the former problem is becoming increasingly popular [4,39]. Since the PFSP is an $\mathcal{NP}$-hard problem when the number of machines is equal to or higher than 3 [22], the problem considered in this paper will be $\mathcal{NP}$-hard too. As a consequence, it is essential to focus on designing metaheuristic approaches for obtaining high-quality solutions in reasonable CPU times.

We propose an approach to solve this problem that relies on a simheuristic algorithm [34]. Simheuristics are a class of efficient optimization algorithms that extend metaheuristic frameworks in a natural and flexible way by integrating simulation. While there are some methodologies for solving the PFSPST, most present some shortcomings such as making hard assumptions on probability distributions of processing times or being able to solve only small-sized instances. In contrast, simheuristics constitute a simple approach, able to cope with these shortcomings [26].

Thus, the main contributions of this work are: *(i)* the realistic extension of the deterministic optimization problem by introducing random variables that allow to model real-life uncertainty and unexpected events (including machine failures or breakdowns); and *(ii)* the development of a flexible simulation-optimization methodology capable of solving the underlying stochastic optimization problem. Being based on simulation, our approach is not limited to a reduced list of probability distributions but, instead, it can be employed with almost any probability distribution –like the Weibull or the log-normal ones, which are extraordinarily flexible to model processing times from historical observations.

The rest of the paper is organized as follows: Section 2 provides a formal definition of the problem under analysis. Related work is reviewed in Section 3. Section 4 describes our solving approach. Section 5 presents the computational experiments carried out, whereas the results are shown in Section 6. Finally, Section 7 offers some conclusions and identifies lines of future research.

2. Problem definition

Consider a set F of f parallel manufacturing factories (Fig. 1). The shop configuration of each factory is a PFSP, which is a particular case of the flowshop scheduling problem (FSP) [32]. In the FSP, there is a set M of m machines and a set N of n jobs that must be processed by each machine. Each job starts to process from the first machine to the last one. Therefore, the number of operations per job is equal to the number of machines. The jth operation of job i is processed by machine j, and can start if the $j-1$th operation on machine $j-1$ has been completed and machine j is free. In the classical version, processing times are supposed to be known in advance and deterministic. Other common assumptions are [6]: *(i)* all operations and jobs are independent and available for processing at time zero; *(ii)* all machines are continuously available and there are no breakdowns; *(iii)* each machine can process at most one job at a time; *(iv)* each job can be processed by only one machine at a time; *(v)* once an operation of a particular job on a given machine has started, it cannot be interrupted (*i.e.*, no preemption is allowed until the processing has been completed); *(vi)* setup and removal times are sequence-independent and are included in the processing times or are negligible; and *(vii)* in-process storage is considered infinite. In the FSP, there are $(n!)^m$ possible solutions since the number of job permutations per machine is $n!$. The PFSP is a specific case which assumes that all machines have the same job permutation and job passing is not allowed. This problem has $n!$ possible solutions.

In this manufacturing layout, a product consists of various jobs/components which have to be processed by the machines located at the factories. In this paper, we overcome the deterministic assumption by allowing stochastic processing times, i.e.: the processing time of job i in machine j, P_{ij}, is considered a random variable following a known probability distribution –either theoretical or

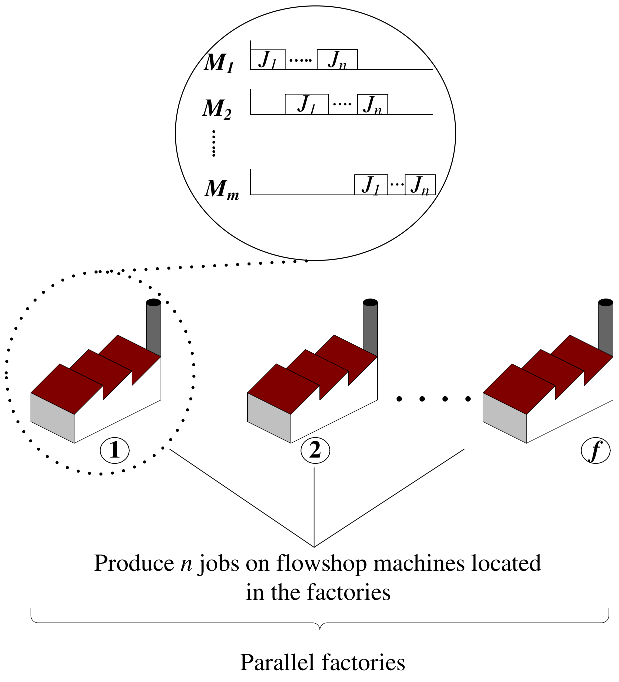
M_1
M_2
M_m
J_1
J_n
1
2
f
Produce n jobs on flowshop machines located in the factories
Parallel factories

Fig. 1. A schematic diagram of the parallel flowshop scheduling problem.

empirical. Logically, a product is considered finished when all its jobs have been completed.

It is required that all components of the product are completed by a given deadline $\tilde{d}$ with a user-defined probability of p (or higher). Note that this common deadline represents a realistic consideration when the jobs should be later assembled in a final stage, either in the same company or by a customer.

Consequently, it is intended that the processing operations for the component at factory k should terminate by the deadline. In a PFSP with a deadline, a specific job sequence has an associated makespan, and the starting time can be set as the deadline minus the makespan. In contrast, the PFSPST is characterized by having random makespans for a given job sequence. Therefore, in our setting, at least one of the three following approaches could be considered:

- To ignore the stochastic nature of the problem and replace the random variables by a representation (for example, their expected value). In our case, this means solving a deterministic version of the problem using the average processing times of the jobs and minimizing the makespan. While ignoring the stochasticity may provide solutions of poor quality, it is not necessarily the case [19]. This is due to the fact that a pure optimization algorithm is faster than one using simulation and, as a consequence, may visit more solutions during a limited amount of time. Thus, if the level of stochasticity is low, there is a chance that solutions found are robust enough to have a good performance in a stochastic environment. This approach is labeled as *makespan* (M) in the following.
- To minimize the expected makespan. This approach stresses the average behavior of the layout. However, if the starting time is set as the deadline minus the expected makespan, there is no guarantee that all processing operations will be completed on time. This approach is labeled as *expected makespan* (EM) in the following. The corresponding value for each factory k is denoted by EM_k.
- To ensure that the final product will be finished on time with a probability p. This option allows the decision-maker to include a constraint that sets the probability of finishing on time. For this reason, in addition to the EM, an interval of time is considered at each factory k, IT_k. The starting time is set as the deadline minus the expected makespan minus a “safety” time interval. Considering this interval increases the probability that all processing operations are completed on time. This approach is labeled as *makespan percentile* (PM) in the following. The makespan percentile value for each factory k is denoted by PM_k. Assuming that factories are independent from each other, p can be computed as: $p = \prod_{k=1}^{f} p_k$. Also, if an equal allocation of probabilities is assumed, we have $p_k = \sqrt[f]{p}$. Therefore, the problem is equivalent to ensure that factory k will finish its jobs with a probability p_k. In order to do so, the p_k-th makespan percentile can be computed for each factory k given a sample of makespans and its starting time can be set as the deadline minus the makespan percentile.

Fig. 2 shows the schematic diagram of the considered problem, illustrating the concepts of starting time, expected makespan, and makespan percentile. As it is shown in this figure, there are f parallel manufacturing factories with flowshop configurations, and each factory processes a predetermined set of jobs (*e.g.*, factory 1 processes jobs $J_1, J_2, ..., J_{i_1}$). All jobs in f factories should be completed by the deterministic deadline $\tilde{d}$ with a common probability of at least p. Each factory k finishes the process of its determined set of jobs before $\tilde{d}$ with a probability of p_k. The expected makespan is shown for each factory (*e.g.*, the expected makespan for the job sequence of factory 1 is denoted by EM_1). The interval time, which increases the probability that jobs are completed before $\tilde{d}$, is also shown

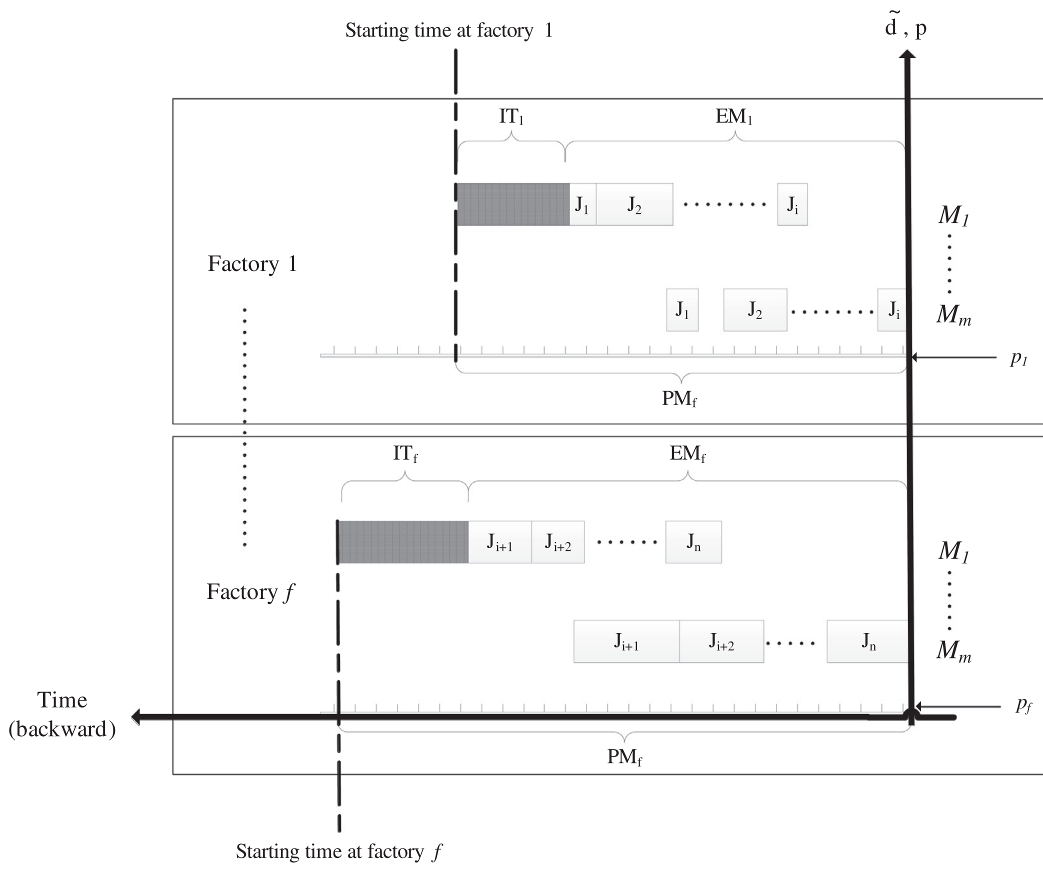

Fig. 2. Starting time, expected makespan and makespan percentile in the parallel flowshop scheduling problem with stochastic processing times.

(*e.g.*, the interval time at factory 1 is denoted by IT_1). The makespan percentile, which is the sum of the interval time and the expected makespan, is also shown for each factory (makespan percentile at factory 1 is denoted by PM_1). The starting time of the operations, which is equal to $\tilde{d}$ minus the makespan percentile, is shown for each factory (*e.g.*, the starting time at factory 1 is denoted by SP_1).

Clearly, the choice between the last two approaches depends on the risk-aversion of the decision-maker. For example, if she prefers to focus on the worst outputs (*i.e.*, the largest makespans), it is better to minimize the makespan percentile requiring a high probability threshold. On the contrary, if she prefers to analyze the average case, she should focus on minimizing the expected makespan.

The solutions with minimum expected makespan are the job sequences with less average completion time in a stochastic condition. In these solutions, the latest starting time to process the jobs is equal to the given deadline minus the expected makespan. But these solutions do not guarantee that jobs are finished on time with a given probability. Notice that the solutions with a lower makespan percentile contain a “safety-stock” interval of time that increases the probability of completing all jobs by the given deadline. The objective is to find a robust starting time for each factory, so each one starts to process at the latest possible time while processing all components on time. We propose a simheuristic approach considering three different objective functions: makespan (assuming deterministic processing times), expected makespan, and a user-defined makespan percentile. Our goal is to study and compare these different scenarios. In Section 4, we present some variants of an algorithm able to deal with each of the aforementioned scenarios.

3. Related work

In this section, we review some of the works related to the problem under study, which has not been addressed yet in the scientific literature. Even so, different streams are related to our problem, namely: the PFSPST and the distributed flowshop scheduling problem (DFSP), the latter assuming deterministic processing times.

3.1. Permutation flowshop problem with stochastic processing times

Baker and Trietsch [5] designed heuristics for addressing the 2-machine PFSPST, where the processing times are independent random variables following specific probability distributions. Later, Baker and Altheimer [4] presented a methodology for the *m*-

machine version. In addition, several variations of the PFSPST have been analyzed. For instance, Allaoui et al. [3] and Choi and Wang [12] worked on the stochastic hybrid flowshop scheduling problem, aiming at minimizing the expected makespan. The same problem was tackled by Kianfar et al. [39] with the goal of minimizing the average tardiness of jobs. A novel approach is applied in [64] for tackling the multi-objective stochastic PFSP, where both the flow time and delay time of jobs are minimized.

An interesting line is related to uncertainty. Basically, there are two categories: proactive (or robust) scheduling and reactive scheduling. For works falling in the first category, Roy [53] proposed constructing an original predictive schedule. The basic aim is to find schedules that do not require re-computation from scratch (or significant changes) when confronting disruptions. These works may consider probability distributions or sets of scenarios. The predicting approach is used in some fields, such as cloud environment. Thus, for example, Bhimani et al. [7] propose the FiM methodology, which is able to provide an accurate prediction of parallel computation time for large-size datasets. For this, stochastic Markov models and machine learning models are employed. Al Kattan and Maragoud [1], Ghezail et al. [23], and Liu et al. [42] addressed the PFSP with uncertainty implementing proactive scheduling strategies.

Reactive scheduling consists in revising and re-optimizing schedules when unexpected events take place. A classic option is to obtain a predictive scheduling and then try to repair it according to the actual state of the system. A comprehensive review on rescheduling under disruptions is provided by Katragjini et al. [38].

Some authors employ exact methods for addressing the PFSPST. A common disadvantage in many of these methods is that they need to assume a specific set of probability distributions, and can only be employed to solve relatively small instances. Moreover, it may be difficult to adapt them for handling dependencies among processing times. On the contrary, simulation techniques enable researchers to deal with these situations in a natural way. Two interesting examples are provided in [4,33]. The former authors, for instance, propose an approach combining heuristics and simulation. They test three heuristic methods: two relying on the CDS heuristic [9] and one on the NEH heuristic [47].

3.2. *Distributed flowshop scheduling problem*

The DFSP has several similarities with the problem under study: there are *f* identical permutation flowshops where *n* jobs have to be processed [45]. However, in the DFSP the jobs have not been assigned to each flowshop, so this assignment becomes part of the decision problem. The DFSP was first studied by David et al. [13] to model a glass industry considering non-delay flowshops and a batch production mode. Note that each factory is treated as a line in our paper, but the mathematical scheduling problem is the same. The DFSP has been also studied under different names: *e.g.*, parallel flowline [58] and parallel flowshops [10]. The particular two-machine-flowshop layout in each factory has been solved using approximate algorithms by Zhang and Van De Velde [63] and Al-Salem [2]. This particular problem turns to be a pure assignment problem due to Johnson's rule [32]. For a general configuration of *m* machines, Naderi and Ruiz [45] have proposed and compared several mixed integer linear programming models and constructive heuristics to solve the problem. Regarding iterated optimization algorithms, the problem has received an increasing attention for makespan minimization in the last years. Gao and Chen [20] proposed a genetic algorithm, using local search phases, based on interchange and insertion of jobs. A tabu search algorithm is proposed by Gao et al. [21]. An iterated greedy (IG) algorithm without local search phases is presented by Lin et al. [41]. A scatter search algorithm with a reference set made up of solutions and restarts mechanisms is proposed by Naderi and Ruiz [46]. Fernandez-Viagas and Framinan [15] presented an IG algorithm with bounded local search phases employing properties of the problem to reduce the space of solutions. Recently, Ribas et al. [51] have proposed several constructive heuristics and two simple iterated algorithms (IG algorithm and ILS) with variable neighborhood searches for the DFSP but with zero-buffer flowshops (blocking constraint).

A particular case of the DFSP refers to the so-called distributed assembly flowshop scheduling problem, which combines the DFSP with assembly scheduling concepts. In this problem, a distributed flowshop composed of *f* identical flowshops is followed by a single assembly operation. Then, *n* jobs consisting each one of *f* components have to be assembled after each component has been manufactured in one of the flowshops. This decision problem includes job assignment plus the scheduling of jobs in the assembly line. The main references for this problem are [28,29]. In the first reference, the authors consider the objective of makespan minimization, while in the second sequence-dependent setup times are studied. In both cases the problem is addressed using approximate algorithms. As in the assembly scheduling problems, we are not aware of references dealing with stochastic processing times.

4. Proposed approach

To address the problem described, we present an algorithm considering different objective functions: the ILS$_M$ algorithm considers the deterministic version of the problem and minimizes the makespan, while the SIM-ILS$_{EM}$ and the SIM-ILS$_{MP}$ algorithms minimize the expected makespan and the makespan percentile, respectively. For each solution returned by an algorithm, the (deterministic) makespan, the expected makespan, and the makespan percentile are computed. The aim of working with different objective functions is to study and compare their behavior. While simulation techniques are used in the SIM-ILS$_{EM}$ and the SIM-ILS$_{MP}$ algorithms, the ILS$_M$ algorithm, which works with average processing times, skips the related procedures. From here, we use 'SIM-ILS algorithm' to refer to the basic structure of the SIM-ILS$_{EM}$ and the SIM-ILS$_{MP}$ algorithms. Note that in the ILS$_M$ and the SIM-ILS$_{EM}$ algorithms the PFSPs can be separately solved, while these problems are interrelated in the SIM-ILS$_{MP}$ because of the probability.

Currently, there is a lack of methodologies for solving stochastic combinatorial optimization problems such as the one under consideration, and most of the existing ones present important drawbacks. Simheuristics represent a powerful alternative, which integrates simulation into metaheuristic-driven frameworks. One drawback that most exact methods face is related to the size of the instances: they are not able to solve large-size instances in short computing times. On the contrary, metaheuristic-based approaches do not suffer this problem. In fact, simheuristics are able to obtain good solutions very fast because simulation (which may be time-consuming) is only used to assess a subset of the solutions visited. Another typical drawback in most traditional approaches is related to the probability distributions they can use to model processing times. Being based on simulation, simheuristics do not make any assumption, so any probability distribution can be used to model stochastic processing times. As a consequence, simheuristics have been used in a wide range of fields such as vehicle routing [27,35], arc routing [24], scheduling [25], healthcare [18], telecommunication systems [8], logistics [14,48], and even finance [50].

We propose a SIM-ILS algorithm combining the iterated local search (ILS) metaheuristic with Monte Carlo simulation (MCS). The ILS is a simple methodology that generates a sequence of solutions by applying an iteratively local search to modify the current search point [43]. As the literature shows, ILS is an efficient framework to solve the PFSP. The IG metaheuristic, which follows a similar structure, was first applied to the PFSP considering makespan minimization by Ruiz and Stützle [54]. Its outstanding performance and simple implementation have boosted its application to other scheduling problems (*e.g.*, [49]). In our approach, the metaheuristic searches for promising solutions while MCS techniques are employed to assess their performance in a stochastic environment. The promising solutions are returned by the metaheuristic when solving a (deterministic) PFSP instance, which is created from the original PFSPST instance by replacing the random variables P_{ij} by constant values p_{ij} using the means ($p_{ij} = E[P_{ij}]$). Note that a solution for the PFSP is also a solution for the PFSPST. MCS is applied to a given solution in order to compute the expected makespan or makespan percentile. This methodology assumes that there is a strong correlation between solutions for the PFSP and the PFSPST.

Our algorithm works with a list of best stochastic solutions found and the best deterministic solution found. The best deterministic one is the job sequence with the smallest makespan referring to the PFSP instance. Depending on the objective considered, the best stochastic solutions found are the job sequences with the smallest expected makespans or makespan percentiles, referring to the PFSPST instance. The algorithm starts solving the PFSP. The obtained result is improved by means of a local search and, afterward, is set as the best deterministic solution and the best stochastic solution. As mentioned before, the MCS is employed to found promising deterministic solutions in order to assess their performance, which allows to identify the best stochastic solutions. During the algorithm execution, the best stochastic solutions are saved in a list with length l. This list is sorted iteratively in increasing order of the considered objective function value. Thus, the solution at the first position is considered as the best stochastic solution. A parameter-free acceptance criterion is applied to decide whether the base solution is replaced by a new one. The steps of our algorithm are detailed in Fig. 3 and explained below.

4.1. Generation of the initial solution

A biased-randomized version of the classical NEH heuristic [47] described in [33] are proposed to generate initial solutions. This randomized version is able to provide different initial solutions when is repeatedly executed using different seeds for the random number generator. The NEH heuristic is an iterative algorithm with two stages. First, an initial order is generated by sorting the jobs by total completion time on all machines in non-increasing order. In the second stage, jobs are iteratively inserted into a partial sequence according to the order. Jobs are inserted at the position of the partial sequence that results in the minimum makespan. Since all steps are deterministic, there is only one possible solution. In the biased-randomized version, a skewed distribution probability is employed to assign a probability of being selected to each job. Following the logic behind the classical heuristic, the jobs with higher completion times are assigned a higher probability. The discrete version of the decreasing triangular distribution is used.

4.2. Solution improvement

An iterative procedure using “shift-to-left” as the first-improvement type pivoting rule is applied in different parts of our algorithm to improve solutions. This procedure has been proposed in several works (*e.g.*, [54]). Each iteration of the procedure consists of three steps. In the first, a position s is randomly selected, without repetition, from the current job sequence. The selected positions are saved in a selection list. In the second step, the job placed in the position s is removed from the sequence and the “shift-to-left” movement is applied, *i.e.*, the insertion of the job in each possible position at the left side of s is tested (see Fig. 4). The makespan of each option is calculated through the accelerations of Taillard [56]. Finally, the job is inserted in the position resulting in the sequence with the smallest makespan. The iteration of these steps is continued until all positions have been selected or a better solution is achieved. If there is an improvement, the procedure is restarted with an empty selection list.

4.3. Simulation

The assessment of a solution applying MCS follows these steps: *(i)* a number of iterations (*numsim*) is considered to repeat the simulation process; *(ii)* a value is generated for each random variable (*i.e.*, job processing time) according to the probability distribution associated, and the makespan is computed; *(iii)* the previous step is repeated *numsim* times; and *(iv)* a performance measure is computed, e.g.: the expected makespan, $E[C_{max}]$, or the makespan percentile for a probability *pro*, $P[C_{max}]^{pro}$. While the assessment of solutions during the search is done quickly (*i.e.*, *numsim* is relatively small, represented by MCS_s in the flowchart), a long simulation (*numsim* is relatively large, MCS_l) is used at the end to provide accurate estimates related to the best deterministic and stochastic solutions.

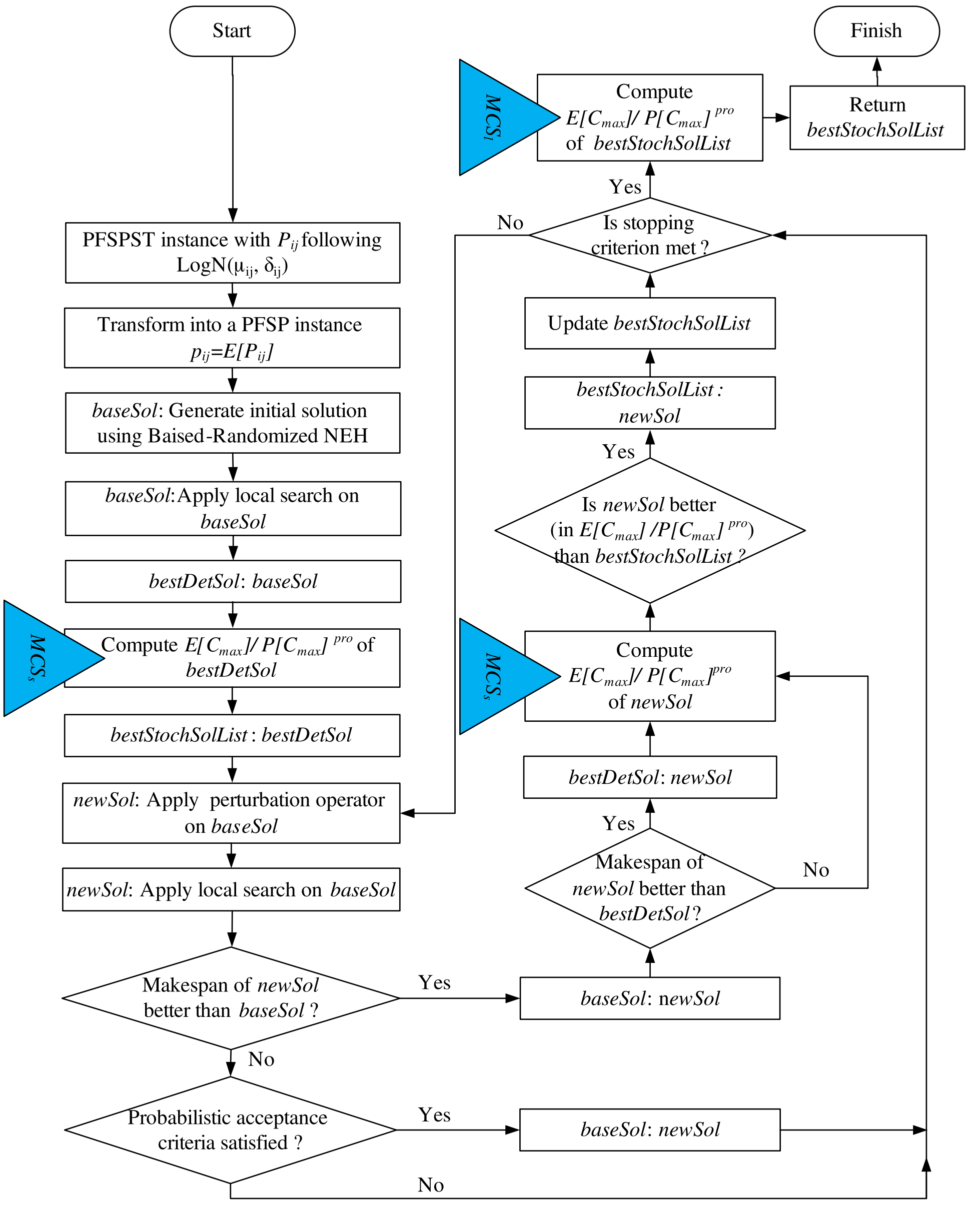
Start
PFSPST instance with P_{ij} following LogN(μ_{ij}, δ_{ij})
Transform into a PFSP instance $p_{ij}=E[P_{ij}]$
baseSol: Generate initial solution using Baised-Randomized NEH
baseSol:Apply local search on *baseSol*
bestDetSol: *baseSol*
MCS_s
Compute $E[C_{max}]/P[C_{max}]^{pro}$ of *bestDetSol*
bestStochSolList : *bestDetSol*
newSol: Apply perturbation operator on *baseSol*
newSol: Apply local search on *baseSol*
Makespan of *newSol* better than *baseSol* ?
Yes
No
baseSol: *newSol*
Probabilistic acceptance criteria satisfied ?
Yes
No
baseSol: *newSol*
Makespan of *newSol* better than *bestDetSol*?
Yes
No
bestDetSol: *newSol*
MCS_s
Compute $E[C_{max}]/P[C_{max}]^{pro}$ of *newSol*
Is *newSol* better (in $E[C_{max}]/P[C_{max}]^{pro}$) than *bestStochSolList* ?
Yes
bestStochSolList : *newSol*
Update *bestStochSolList*
Is stopping criterion met ?
No
Yes
MCS_l
Compute $E[C_{max}]/P[C_{max}]^{pro}$ of *bestStochSolList*
Return *bestStochSolList*
Finish

Fig. 3. Flowchart of the SIM-ILS algorithm.

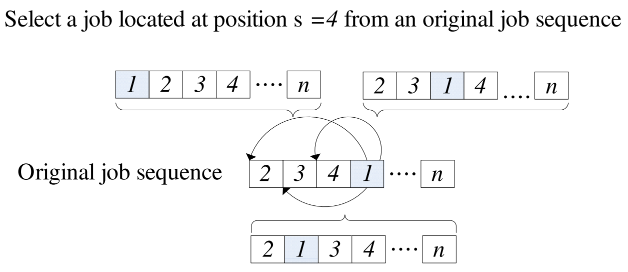

Fig. 4. “shift-to-left” movement.

4.4. Iterated local search

A series of steps are performed iteratively during the search. Initially, a perturbation operator is applied to change the region of the current solution space and then the new solution is improved using the local search explained in Section 4.2. The simple and efficient "enhanced-swap" operator proposed by [36] is used to modify the solution. It takes three steps: *(i)* two different positions are selected randomly from the current job sequence; *(ii)* the jobs at these positions are interchanged; and *(iii)* the "shift-to-left" movement is applied to both jobs.

In the second step, the algorithm decides whether the new solution is accepted. If it has a smaller makespan than the current base solution, then the latter is replaced by the new solution. In this case, the best deterministic solution is accordingly updated (*i.e.*, replaced by the new solution if this has a smaller makespan). Afterwards, a short simulation is applied to check whether the best stochastic solution list has also to be updated considering the objective function value. Finally, if the new solution does not provide a smaller makespan than the current base solution, an acceptance criterion is applied (see Section 4.5). These steps are repeated until the stopping criterion, based on the elapsed CPU time, is reached.

4.5. Acceptance criterion

Our algorithm assigns an acceptance probability to the new solutions that are worse than the current base solution. This criterion, which prevents the algorithm from getting stuck in a local optimum [29], is based on the probabilistic acceptance criterion of the simulated annealing [60,62]. Given a new solution π_n with a worse makespan than the current base solution π_c, the acceptance criterion decides if it is accepted or not. Let $C_{Max}(\pi_c)$ and $C_{Max}(\pi_n)$ denote the makespans of each solution. In the simulated annealing the acceptance of π_n depends on the probabilistic mechanism shown in Eq. (1), where *random* is a random number uniformly distributed between 0 and 1, and *temp* is a parameter.

$$random \leq e^{-\frac{C_{Max}(\pi_n)-C_{Max}(\pi_c)}{temp}} \tag{1}$$

This mechanism is simplified in two aspects. First, the parameter is removed. Secondly, the difference of makespans (which can be the same for instances with distinct quality) is replaced by the relative percentage difference: $RPD = \frac{C_{Max}(\pi_n) - C_{Max}(\pi_c)}{C_{Max}(\pi_c)} \cdot 100$. Thus, this acceptance criterion (Eq. (2)) is simpler, avoiding the need for parameter fine-tuning.

$$random \leq e^{-RPD} \tag{2}$$

5. Computational experiments

The algorithms described in the previous section have been implemented as Java applications and tested on 27 instances. A standard personal computer, Intel QuadCore i5 CPU at 3.2 GHz and 4GB RAM with Windows 7, has been used to execute all tests. This section provides the description of the instances, the tests carried out, and the numerical results. The analysis of the results is presented in the next section.

5.1. Set of instances and test

Since no benchmark instances exist for the problem analyzed, a new set is constructed based on the Taillard instances [57]. Taillard's original benchmark consists of 120 flowshop scheduling instances with $n \in \{20, 50, 100, 200, 500\}$ jobs and $m \in \{5, 10, 20\}$ machines. For each job i on machine j, the corresponding processing times p_{ij} are generated using a uniform [1,99] distribution. For each problem size, there are ten instances in the testbed. In order to adapt this testbed to our problem, we generate 27 instances in the following manner: For a given *n, m*, we use the first 9 out of the 10 Taillard's instances of this size (i.e.: Ta_n_m_1 to Ta_n_m_9). Then, for a given $f \in \{2, 3, 4\}$ we generate one instance where there are $n \cdot f$ jobs to be scheduled. For $f = 2$, the first n jobs are to be scheduled in factory 1 with the processing times given in Ta_n_m_1, the second n jobs in factory 2 with the processing times given in Ta_n_m_2. For $f = 3$, the first n jobs are to be scheduled in factory 1 with the processing times given in Ta_n_m_3, the second n jobs in factory 2 with the processing times given in Ta_n_m_4, and the third n jobs in factory 3 with the processing times given in Ta_n_m_5. Finally, for $f = 4$, the data from instances Ta_n_m_6 to Ta_n_m_9 are used in a similar manner. By doing so, we construct 27 instances with the main parameters given in Table 1. The actual processing times for each job are assumed to follow a log-normal distribution with mean p_{ij} and variance σ_{ij}^2 set to $c \cdot p_{ij}$ (thus, the higher the mean processing time, the higher the variability). Note that p_{ij} is given

Table 1
Instance description.

	n								
	20			50			100		
f / m	5	10	20	5	10	20	5	10	20
2	Ins. 1	Ins. 4	Ins. 7	Ins. 10	Ins. 13	Ins. 16	Ins. 19	Ins. 22	Ins. 25
3	Ins. 2	Ins. 5	Ins. 8	Ins. 11	Ins. 14	Ins. 17	Ins. 20	Ins. 23	Ins. 26
4	Ins. 3	Ins. 6	Ins. 9	Ins. 12	Ins. 15	Ins. 18	Ins. 21	Ins. 24	Ins. 27

Table 2
Results considering low level of variability ($c = 0.25$) and general probability $p = 90\%$.

	ILS_M					SIM-ILS_{EM}				SIM-ILS_{MP}				
Instance	C_{max} (1)	$E[C_{max}]$ (2)	$P[C_{max}]^{pro}$ (3)	Gap (2-1) (%)	Gap (3-2) (%)	$E[C_{max}]$ (4)	$P[C_{max}]^{pro}$ (5)	Gap (4-2) (%)	Gap (5-3) (%)	$E[C_{max}]$ (6)	$P[C_{max}]^{pro}$ (7)	Gap (6-4) (%)	Gap (7-5) (%)	Mean CPU time
1	1505	1516.00	1555.54	0.73	2.61	1512.33	1552.96	−0.24	−0.17	1514.19	1551.75	0.12	−0.08	5.00
2	1758	1772.30	1829.89	0.81	3.25	1762.47	1824.72	−0.55	−0.28	1765.96	1822.13	0.20	−0.14	3.99
3	1862	1869.94	1949.62	0.43	4.26	1864.56	1944.33	−0.29	−0.27	1865.41	1942.73	0.05	−0.08	4.50
4	2154	2169.84	2216.99	0.74	2.17	2167.72	2213.02	−0.10	−0.18	2169.41	2212.05	0.08	−0.04	9.99
5	2829	2854.76	2927.28	0.91	2.54	2852.35	2923.38	−0.08	−0.13	2854.23	2921.13	0.07	−0.08	10.00
6	3179	3194.56	3295.33	0.49	3.15	3189.44	3294.94	−0.16	−0.01	3191.61	3288.55	0.07	−0.19	7.99
7	3413	3451.80	3503.35	1.14	1.49	3440.75	3495.76	−0.32	−0.22	3442.72	3493.16	0.06	−0.07	19.99
8	4306	4333.25	4421.54	0.63	2.04	4332.04	4424.23	−0.03	0.06	4334.49	4418.29	0.06	−0.13	20.00
9	5733	5760.93	5904.21	0.49	2.49	5752.36	5890.05	−0.15	−0.24	5756.43	5884.47	0.07	−0.09	14.00
10	2960	2987.24	3040.81	0.92	1.79	2960.68	3020.68	−0.89	−0.66	2964.55	3019.89	0.13	−0.03	12.46
11	3250	3285.55	3352.58	1.09	2.04	3272.02	3349.40	−0.41	−0.09	3273.94	3346.25	0.06	−0.09	12.49
12	3378	3417.10	3505.31	1.16	2.58	3392.12	3501.08	−0.73	−0.12	3395.54	3490.67	0.10	−0.30	12.49
13	3572	3620.75	3669.11	1.36	1.34	3620.75	3669.11	0.00	0.00	3620.75	3669.11	0.00	0.00	25.01
14	4031	4093.01	4172.92	1.54	1.95	4083.66	4163.44	−0.23	−0.23	4084.43	4156.70	0.02	−0.16	25.00
15	4574	4620.28	4728.14	1.01	2.33	4605.65	4720.76	−0.32	−0.16	4609.00	4714.98	0.07	−0.12	24.99
16	5058	5132.22	5194.14	1.47	1.21	5131.72	5191.09	−0.01	−0.06	5131.72	5191.09	0.00	0.00	49.99
17	6039	6113.22	6205.43	1.23	1.51	6109.34	6208.90	−0.06	0.06	6113.31	6204.41	0.06	−0.07	49.97
18	7013	7090.64	7215.40	1.11	1.76	7081.97	7215.77	−0.12	0.01	7085.22	7204.73	0.05	−0.15	49.98
19	5797	5840.48	5913.77	0.75	1.25	5810.07	5891.05	−0.52	−0.38	5818.16	5889.75	0.14	−0.02	24.98
20	5819	5854.95	5967.56	0.62	1.92	5825.96	5939.00	−0.50	−0.48	5832.76	5935.59	0.12	−0.06	24.95
21	6065	6104.18	6236.87	0.65	2.17	6078.89	6220.23	−0.41	−0.27	6083.24	6217.93	0.07	−0.04	24.98
22	6235	6324.25	6392.90	1.43	1.09	6324.25	6392.90	0.00	0.00	6326.47	6388.82	0.04	−0.06	49.98
23	6414	6502.24	6598.61	1.38	1.48	6498.90	6592.36	−0.05	−0.09	6498.90	6592.36	0.00	0.00	50.01
24	7183	7289.80	7422.74	1.49	1.82	7283.76	7418.55	−0.08	−0.06	7283.75	7407.78	0.00	−0.15	50.00
25	7603	7697.50	7763.14	1.24	0.85	7697.50	7763.14	0.00	0.00	7697.50	7763.14	0.00	0.00	100.03
26	8598	8710.46	8823.77	1.31	1.30	8710.46	8823.77	0.00	0.00	8710.46	8823.77	0.00	0.00	100.01
27	9892	10,038.66	10,178.40	1.48	1.39	10,029.92	10,175.82	−0.09	−0.03	10,029.92	10,175.82	0.00	0.00	99.99
Mean				1.02	1.99			−0.24	−0.15			0.06	−0.08	32.69

Table 3
Results considering medium level of variability ($c = 1$) and general probability $p = 90\%$.

	ILS$_M$					SIM-ILS$_{EM}$				SIM-ILS$_{MP}$				
Instance	C_{max} (1)	$E[C_{max}]$ (2)	$P[C_{max}]^{pro}$ (3)	Gap (2-1) (%)	Gap (3-2) (%)	$E[C_{max}]$ (4)	$P[C_{max}]^{pro}$ (5)	Gap (4-2) (%)	Gap (5-3) (%)	$E[C_{max}]$ (6)	$P[C_{max}]^{pro}$ (7)	Gap (6-4) (%)	Gap (7-5) (%)	Mean CPU time
1	1505	1541.15	1617.25	2.40	4.94	1533.01	1614.37	−0.53	−0.18	1533.96	1606.78	0.06	−0.47	5.00
2	1758	1799.07	1927.61	2.34	7.14	1782.58	1901.69	−0.92	−1.34	1787.49	1897.95	0.28	−0.20	3.99
3	1862	1880.42	2046.20	0.99	8.82	1876.25	2041.74	−0.22	−0.22	1879.09	2030.26	0.15	−0.56	4.50
4	2154	2208.99	2298.54	2.55	4.05	2203.58	2295.21	−0.25	−0.14	2205.94	2287.00	0.11	−0.36	10.00
5	2829	2898.36	3035.07	2.45	4.72	2896.75	3033.22	−0.06	−0.06	2899.76	3032.76	0.10	−0.02	9.99
6	3179	3223.10	3424.39	1.39	6.25	3215.97	3414.35	−0.22	−0.29	3215.97	3414.35	0.00	0.00	8.00
7	3413	3517.97	3617.33	3.08	2.82	3493.77	3603.77	−0.69	−0.37	3495.26	3594.01	0.04	−0.27	19.99
8	4306	4387.69	4552.74	1.90	3.76	4380.10	4564.70	−0.17	0.26	4382.97	4552.50	0.07	−0.27	19.99
9	5733	5811.38	6094.35	1.37	4.87	5792.84	6067.04	−0.32	−0.45	5797.46	6055.08	0.08	−0.20	14.00
10	2960	3030.64	3136.25	2.39	3.48	2979.60	3094.51	−1.68	−1.33	2983.69	3090.15	0.14	−0.14	12.49
11	3250	3336.39	3502.73	2.66	4.99	3321.23	3475.18	−0.45	−0.79	3326.65	3466.45	0.16	−0.25	12.49
12	3378	3482.60	3673.45	3.10	5.48	3431.44	3638.61	−1.47	−0.95	3435.06	3627.96	0.11	−0.29	12.51
13	3573	3710.11	3810.50	3.84	2.71	3708.59	3810.36	−0.04	0.00	3708.59	3810.36	0.00	0.00	25.00
14	4031	4181.10	4332.06	3.72	3.61	4162.39	4322.68	−0.45	−0.22	4174.02	4317.41	0.28	−0.12	25.01
15	4574	4704.41	4911.53	2.85	4.40	4683.63	4901.01	−0.44	−0.21	4686.14	4896.47	0.05	−0.09	25.00
16	5058	5249.31	5356.33	3.78	2.04	5243.28	5349.75	−0.11	−0.12	5243.28	5349.75	0.00	0.00	50.00
17	6039	6241.34	6419.70	3.35	2.86	6226.07	6409.99	−0.24	−0.15	6229.50	6405.71	0.05	−0.07	49.98
18	7013	7205.94	7440.63	2.75	3.26	7205.49	7448.99	−0.01	0.11	7205.94	7440.63	0.01	−0.11	49.98
19	5797	5897.76	6042.62	1.74	2.46	5855.19	6010.80	−0.72	−0.53	5866.03	6001.97	0.19	−0.15	24.96
20	5819	5954.39	6150.09	2.33	3.29	5874.48	6086.12	−1.34	−1.04	5887.17	6081.12	0.22	−0.08	24.92
21	6065	6181.98	6426.37	1.93	3.95	6128.44	6403.68	−0.87	−0.35	6144.92	6395.87	0.27	−0.12	24.99
22	6233	6449.74	6575.00	3.48	1.94	6441.78	6569.46	−0.12	−0.08	6441.78	6569.46	0.00	0.00	50.00
23	6415	6663.16	6858.39	3.87	2.93	6633.10	6821.64	−0.45	−0.54	6642.01	6820.06	0.13	−0.02	49.99
24	7183	7464.61	7726.46	3.92	3.51	7428.87	7676.50	−0.48	−0.65	7440.15	7674.73	0.15	−0.02	49.98
25	7600	7863.12	7990.74	3.46	1.62	7863.12	7990.74	0.00	0.00	7863.12	7990.74	0.00	0.00	100.00
26	8592	8897.85	9103.21	3.56	2.31	8897.85	9103.21	0.00	0.00	8897.85	9103.21	0.00	0.00	99.98
27	9888	10,267.27	10,539.56	3.84	2.65	10,250.60	10,535.72	−0.16	−0.04	10,252.56	10,529.88	0.02	−0.06	99.97
Mean				2.78	3.88			−0.46	−0.36			0.10	−0.14	32.69

Table 4
Results considering high level of variability (c = 1.5) and general probability p = 90%.

	ILS_M					$SIM\text{-}ILS_{EM}$				$SIM\text{-}ILS_{MP}$				
Instance	C_{max} (1)	$E[C_{max}]$ (2)	$P[C_{max}]^{pro}$ (3)	Gap (2-1) (%)	Gap (3-2) (%)	$E[C_{max}]$ (4)	$P[C_{max}]^{pro}$ (5)	Gap (4-2) (%)	Gap (5-3) (%)	$E[C_{max}]$ (6)	$P[C_{max}]^{pro}$ (7)	Gap (6-4) (%)	Gap (7-5) (%)	Mean CPU time
1	1505	1558.65	1652.71	3.56	6.03	1543.93	1645.57	−0.94	−0.43	1547.66	1634.96	0.24	−0.64	5.00
2	1758	1815.86	1971.38	3.29	8.56	1793.26	1936.85	−1.24	−1.75	1796.58	1932.27	0.19	−0.24	3.99
3	1862	1899.64	2088.00	2.02	9.92	1883.39	2087.17	−0.86	−0.04	1887.72	2070.03	0.23	−0.82	4.50
4	2154	2230.49	2338.89	3.55	4.86	2223.55	2331.53	−0.31	−0.31	2225.38	2325.86	0.08	−0.24	10.00
5	2829	2923.50	3100.37	3.34	6.05	2920.98	3095.04	−0.09	−0.17	2925.05	3089.05	0.14	−0.19	9.99
6	3179	3250.82	3500.13	2.26	7.67	3231.39	3488.72	−0.60	−0.33	3239.13	3474.39	0.24	−0.41	8.00
7	3413	3550.25	3693.79	4.02	4.04	3521.54	3653.49	−0.81	−1.09	3522.88	3647.18	0.04	−0.17	20.00
8	4306	4417.80	4645.59	2.60	5.16	4408.96	4633.95	−0.20	−0.25	4414.71	4618.62	0.13	−0.33	19.99
9	5733	5829.65	6148.97	1.69	5.48	5815.95	6148.41	−0.23	−0.01	5820.54	6141.05	0.08	−0.12	14.00
10	2960	3051.56	3186.60	3.09	4.43	2989.60	3128.79	−2.03	−1.81	2989.60	3128.79	0.00	0.00	12.46
11	3250	3369.09	3547.77	3.66	5.30	3348.04	3530.25	−0.62	−0.49	3354.05	3526.75	0.18	−0.10	12.50
12	3378	3510.02	3741.10	3.91	6.58	3455.00	3711.78	−1.57	−0.78	3459.46	3691.02	0.13	−0.56	12.49
13	3570	3745.06	3862.43	4.90	3.13	3745.06	3862.43	0.00	0.00	3745.06	3862.43	0.00	0.00	24.99
14	4031	4221.27	4401.91	4.72	4.28	4206.78	4396.52	−0.34	−0.12	4210.35	4395.90	0.08	−0.01	24.99
15	4574	4743.71	4997.22	3.71	5.34	4729.26	5001.05	−0.30	0.08	4737.43	4987.14	0.17	−0.28	24.99
16	5058	5312.82	5448.19	5.04	2.55	5304.92	5443.07	−0.15	−0.09	5304.92	5443.07	0.00	0.00	50.01
17	6039	6301.25	6533.04	4.34	3.68	6289.15	6517.20	−0.19	−0.24	6293.81	6507.38	0.07	−0.15	49.97
18	7013	7282.57	7597.07	3.84	4.32	7273.14	7594.89	−0.13	−0.03	7277.18	7565.94	0.06	−0.38	49.99
19	5797	5967.38	6140.62	2.94	2.90	5881.63	6066.46	−1.44	−1.21	5889.21	6054.73	0.13	−0.19	24.99
20	5819	5990.94	6236.24	2.95	4.09	5908.93	6161.28	−1.37	−1.20	5916.77	6155.52	0.13	−0.09	24.94
21	6065	6230.30	6563.94	2.73	5.36	6169.34	6505.67	−0.98	−0.89	6172.18	6485.97	0.05	−0.30	24.98
22	6237	6531.98	6675.40	4.73	2.20	6509.74	6672.00	−0.34	−0.05	6510.28	6662.21	0.01	−0.15	49.99
23	6411	6718.36	6953.37	4.79	3.50	6707.55	6922.70	−0.16	−0.44	6707.55	6922.70	0.00	0.00	50.00
24	7183	7519.80	7820.43	4.69	4.00	7499.62	7802.18	−0.27	−0.23	7499.62	7802.18	0.00	0.00	50.00
25	7603	7970.91	8150.20	4.84	2.25	7962.01	8126.37	−0.11	−0.29	7962.01	8126.37	0.00	0.00	100.00
26	8600	9011.27	9264.51	4.78	2.81	9011.27	9264.51	0.00	0.00	9011.27	9264.51	0.00	0.00	99.98
27	9892	10,386.80	10,731.39	5.00	3.32	10,354.76	10,707.00	−0.31	−0.23	10,354.76	10,707.00	0.00	0.00	99.99
Mean				3.74	4.73			−0.58	−0.46			0.09	−0.20	32.69

in Taillard's testbed, and c is a parameter to be controlled in the experiments.

Table 1 gathers the following characteristics for each instance: name, the number of jobs per factory (n), number of machines (m) and number of factories (f). In real-life applications, empirical distributions based on historical data may be used. Instances are available from the authors upon request.

Three levels of c (small, medium, and high) are considered and set to 0.25, 1 and 1.5, respectively. The values 80%, 90%, and 95% are considered for the general probability p (used only for the SIM-ILS$_{MP}$ algorithm). The maximum computational time for solving the PFSPST of each factory is limited to $0.05 \cdot n \cdot m$ seconds, which seems a reasonable amount for real-life applications. Each instance has been run ten times, using each time a different seed which is randomly generated to feed the biased-randomization process. Only the best result among the ten runs is stored. In order to assess the solutions, 600, and 1000 runs are employed during the algorithm and at the end, respectively. Note that the selection of these values are mainly driven by the computing time available. Thus, if more time is available, then these values can be incremented to obtain better and more accurate results.

5.2. Results

Results are displayed in Tables 2–4, where each table represents a specific level of processing time variability: low, medium, and high. Due to space limitations and the fact that results show similar trends for all three values of general probability, only those related to 90% are shown. The composition of the tables is as follows. The first column identifies the instance. The next five summarize the results of the ILS$_M$ algorithm, which considers makespan minimization. For each instance, they show the following information regarding the best solution found: $C_{max}(1)$, $E[C_{max}](2)$, $P[C_{max}]^{pro}(3)$, the gap between the first two measures, computed as: $(E[C_{max}](2) - C_{max}(1))/C_{max}(1) \cdot 100$, and the gap between the second and the third ones. While the first gap represents the 'extra' processing time, on average, for applying a solution assuming deterministic processing times, the second focuses on percentiles, showing the additional processing time required to finish the product with a probability of 90%. The next four columns provide the following results of the SIM-ILS$_{EM}$ algorithm, which minimizes the expected makespan: $E[C_{max}](4)$, $P[C_{max}]^{pro}(5)$, gap between the expected makespan of the best solutions found by the ILS$_M$ and the SIM-ILS$_{EM}$ algorithms, and the gap of percentiles among the same solutions. The third gap, which is expected to be null or negative, shows the benefit of using a simheuristic approach (*i.e.*, taking into account the variability of the processing times) in terms of expected makespan. The fourth gap quantifies the difference of percentiles. Similarly, the next four columns refer to the best solution found by the SIM-ILS$_{MP}$ algorithm, which minimizes the makespan percentile. In particular, they contain: $E[C_{max}](6)$, $P[C_{max}]^{pro}(7)$, and gaps of expected makespans and percentiles between the best solutions found by the SIM-ILS$_{EM}$ and SIM-ILS$_{MP}$ algorithms. These gaps allow us to quantify the processing time difference based on whether we minimize one measure or the other. Finally, the last column shows the mean computational time of the three solutions obtained. In addition, a row is added at the end of each table to gather the mean gaps and computational time among instances.

Figs. 5–7 summarize the information of the gaps in Tables 2–4. Particularly, Fig. 5 shows boxplots of the gaps related to the ILS$_M$ algorithm, *i.e.*, the gaps between $E[C_{max}]$ and C_{max}, and between $P[C_{max}]^{pro}$ and $E[C_{max}]$. Fig. 6 compares the SIM-ILS$_{EM}$ and ILS$_M$ algorithms in terms of $E[C_{max}]$ and $P[C_{max}]^{pro}$. Similarly, Fig. 7 illustrates the differences between the SIM-ILS$_{MP}$ and SIM-ILS$_{EM}$ algorithms in terms of the same performance measures.

Focusing on instance 14, Fig. 8 represents the 30 solutions found (resulting of 3 algorithms and 10 seeds). Each column is a measure, and line formats are used to distinguish algorithms. As the previous figure, this analysis provides insights about a "potential" trade-off between the measures. Additionally, this figure gives information about the effect of using multiple seeds.

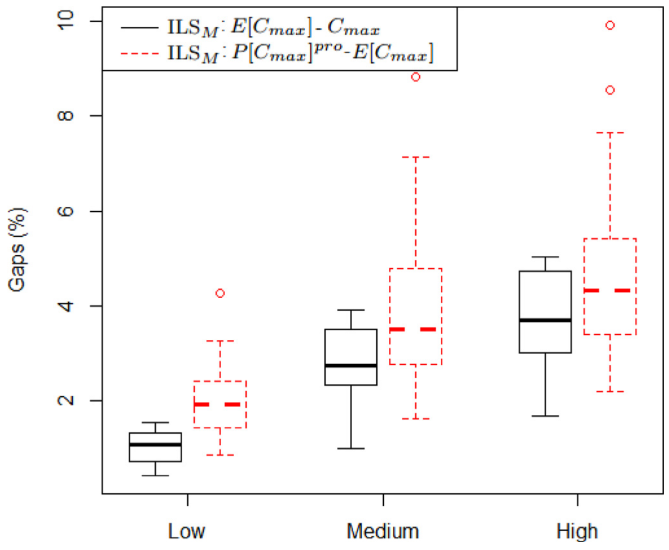

Fig. 5. Boxplots of gaps between different performance measures regarding the ILS$_M$ algorithm, which consider different levels of stochasticity (medium level of variability and p = 90%).

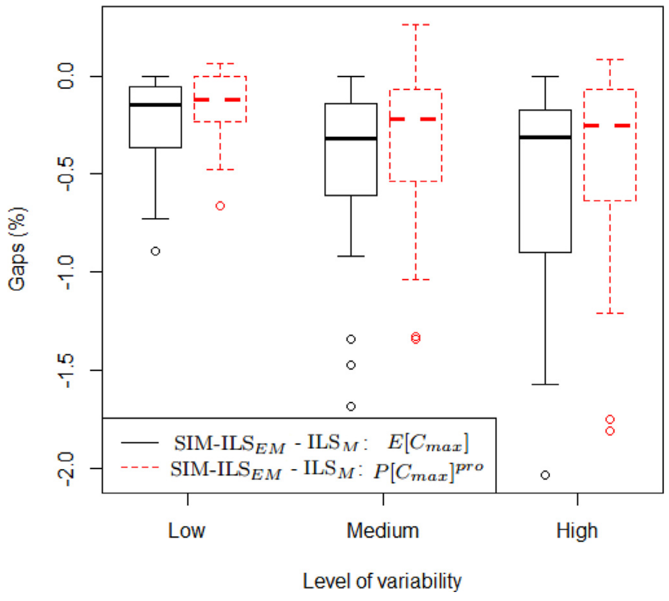

Fig. 6. Boxplots of gaps comparing the performance of the SIM-ILS$_{EM}$ and ILS$_M$ algorithms (medium level of variability and p = 90%).

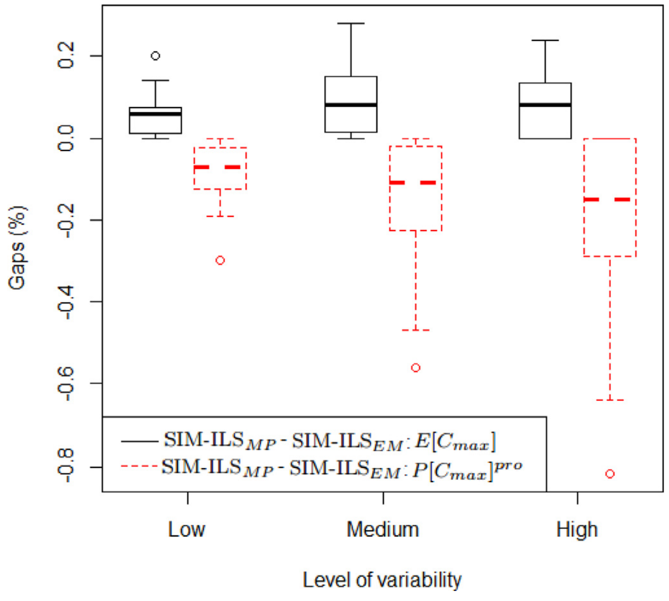

Fig. 7. Boxplots of gaps comparing the performance of the SIM-ILS$_{MP}$ and SIM-ILS$_{EM}$ algorithms (medium level of variability and p = 90%).

Fig. 9 illustrates the relationship between probability required, variability level of the processing times and $P[C_{max}]^{pro}$ for the instance 14 using the SIM-ILS$_{MP}$ algorithm. Finally, Fig. 10 displays the effects of different instance characteristics on $P[C_{max}]^{pro}$ considering a medium level of variability and a probability of 90%. First, an analysis of variance was carried out to identify which factors and pairwise interactions had a statistically significant effect on the results. For each of these elements (single factors or a pair of them), a figure is drawn which shows the mean value associated with each level of the factor or combination of levels for the pair of factors. Given the randomness in the generation of instances, we expect that all factors have significant positive effects.

6. Analysis of results

Tables 2–4 provide detailed information on the performance of our algorithms. The following comments refer to the results of the ILS$_M$ algorithm. Mean gaps between C_{max} and $E[C_{max}]$ for small, medium, and high levels of variability are 1.02%, 2.78%, and 3.74%, respectively. These values between $E[C_{max}]$ and $P[C_{max}]^{pro}$ are 1.99%, 3.88%, and 4.73%. These values quantify the extra processing time required, on average, when variability is not considered, and the processing time needed to satisfy the deadline with a

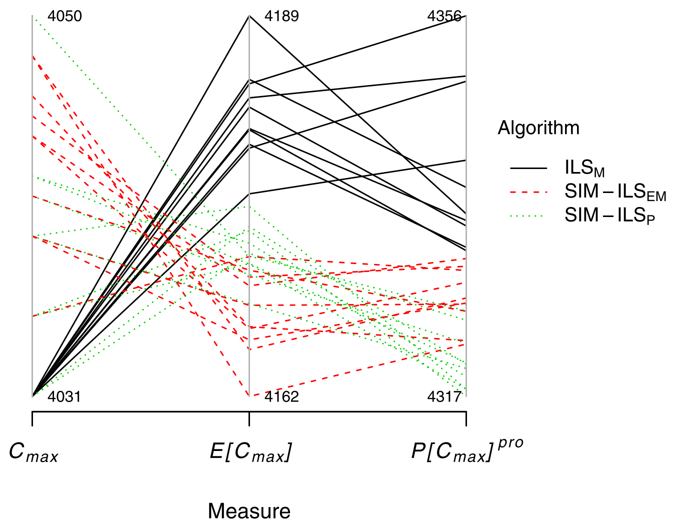

Fig. 8. Parallel coordinates plot showing different measures for solutions found with the different algorithms for instance 14, considering a medium level of variability and 10 seeds.

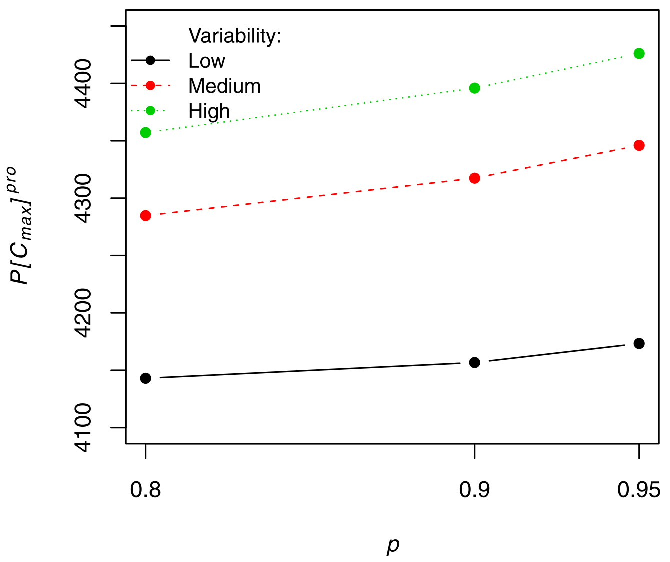

Fig. 9. $P[C_{max}]^{pro}$ as function of general probability and variability level for instance 14 considering the SIM-ILS$_{MP}$ algorithm.

probability of 90%. For example, in the scenario of low variability, the processing time will be on average 1.02% higher than that expected ignoring the stochasticity, and the processing time needed to finish with the specific probability will be 1.99% higher than the $E[C_{max}]$. Both gaps increase as the variability is incremented. Comparing the results of the ILS$_M$ and the SIM-ILS$_{EM}$ algorithms, the mean gaps of $E[C_{max}]$ (– 0.24%, – 0.46% and – 0.58%) and $P[C_{max}]^{pro}$ (– 0.15%, – 0.36% and – 0.46%) quantify the benefits of using a simheuristic algorithm. Regarding the results of the SIM-ILS$_{EM}$ and SIM-ILS$_{MP}$ algorithms, the mean gaps of $E[C_{max}]$ (0.06%, 0.10% and 0.09%) and $P[C_{max}]^{pro}$ (– 0.08%, – 0.14% and – 0.20%) at the different level of variability evidence the benefits of using one or the other approach. Thus, the main findings are: *(i)* ignoring the variability in processing times may have an important effect on the performance measures (even in scenarios with a low level of variability); *(ii)* the solutions found by the SIM-ILS$_{EM}$ and the SIM-ILS$_{MP}$ algorithms are relatively similar in terms of these measures but not equal; and *(iii)* the gaps tend to increase with the variability, *i.e.*, minimizing the expected makespan is almost equivalent to consider the makespan percentile when the variability is low, but the difference increases as the variability is incremented. As a consequence, a decision-maker has to assess whether she prefers to minimize the expected makespan (*i.e.*, processing finished at the deadline, on average) or the percentile (*i.e.*, be sure that the processing will be finished on or before the deadline with a given probability), which may be seen as a more conservative or risk-aversion approach.

Regarding the ILS$_M$ algorithm, Fig. 5 shows that the gaps between $E[C_{max}]$ and C_{max} increase as the level of stochasticity gets higher. The same applies for the gaps between $P[C_{max}]^{pro}$ and $E[C_{max}]$. The distributions are relatively symmetric, and have only a few

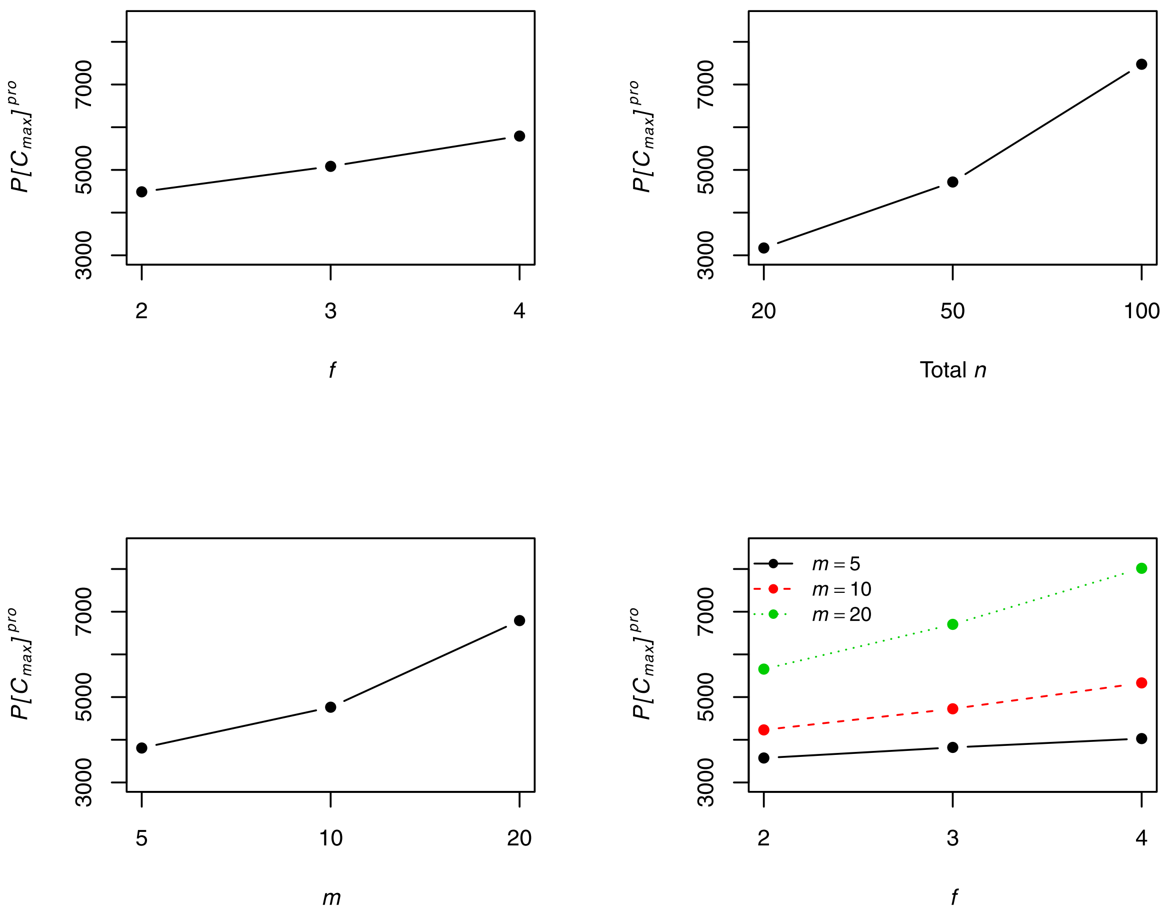

Fig. 10. Effect of different instance characteristics on $P[C_{max}]^{pro}$ considering the SIM-ILS$_{MP}$ algorithm.

outliers. The benefit of using the SIM-ILS$_{EM}$ algorithm instead of the ILS$_M$ algorithm in terms of $E[C_{max}]$ is displayed in Fig. 6. The gaps tend to be higher as the level of stochasticity is increased, and the distributions are not symmetric. The trend is not so clear when studying the gaps of $P[C_{max}]^{pro}$. Finally, boxplots in Fig. 7, which compare the performance of the SIM-ILS$_{MP}$ and SIM-ILS$_{EM}$ algorithms, show that there is much more variability in the gaps of $P[C_{max}]^{pro}$, and they tend to be more significant, increasing with the level of stochasticity.

Fig. 8 shows that there is a stronger correlation between the simheuristic-based algorithms, in the sense that the profiles are similar. It is interesting to analyze the differences among the solutions obtained with multiple seeds. For instance, while solutions of the ILS$_M$ algorithm have the same or a similar C_{max}, these solutions may differ significantly in the other measures (*i.e.*, there are solutions more robust than others). For the instance studied, the ranges of the last two measures are higher than that of the first.

Fig. 9 represents a valuable tool for a decision-maker. It analyses the relationship between the probability required to process a product on or before the deadline and the processing time needed. As the probability tends to 1 (*i.e.*, no risk) the processing time increases. Instead of having a single solution, the decision-maker may choose the best option (given risk-aversion, company policies/situation, etc.) among many. As expected, for a given p value, $P[C_{max}]^{pro}$ increases as the variability is incremented.

Fig. 10 reveals that factors total n, m, f, and the interaction between f and m have statistically significant and positive effects on $P[C_{max}]^{pro}$. The ranges related to total n and m are the highest. While the effects of f and total n seem lineal, the effect of m draws a convex function. Focusing on the interaction, it can be concluded that the effect of f is positive for any value of m, but $P[C_{max}]^{pro}$ increases as m is incremented.

7. Conclusions

The manufacturing industry is becoming increasingly complex and competitive. Companies need powerful optimization algorithms to design proper strategies that make them efficient in order to remain in the market. Although there is an extensive literature on classical scheduling problems, there is a lack of works on richer and more realistic problems. In this context, our work studies a novel problem called parallel stochastic flowshops scheduling problem with stochastic processing times. It consists of the manufacturing of a product that requires several jobs that are performed in independent factories. The sub-problem of each factory can be modeled as a permutation flowshop scheduling problem with stochastic processing times. All factories are expected to finish on or before a given deadline. This problem describes several real-life applications where a company acquires intermediate products from others and assembles them to obtain a final product with a higher added value.

Three algorithms are proposed to deal with this problem, one for each objective function to be minimized: the makespan (ignoring the stochasticity), the expected makespan, and the makespan percentile associated with a user-defined probability *p*. This percentile is the value below which a given proportion *p* of makespans fall when simulating scenarios. Thus, it can be interpreted as follows: if the starting time in a factory is set to the deadline minus this percentile, the processing of the product will be finished by the deadline with a probability *p*. While all algorithms rely on the iterated local search metaheuristic, the second and the third ones are simheuristic algorithms, which integrate Monte Carlo simulation techniques in order to deal with the stochasticity. Note that the second algorithm is intended to provide good results on average, whereas the third one aims at guaranteeing that the manufacturing will be finished, with a given probability, on or before the deadline. A set of computational experiments allow us to compare the algorithms in terms of makespan, expected makespan, and makespan percentile. It is proven that: *(i)* gaps among algorithms for each measure increase as the level of stochasticity is incremented; *(ii)* while there is a strong correlation between simheuristic algorithms (in the sense that solutions having the best performance in terms of expected makespan are also of good quality regarding makespan percentile, and the other way around), this correlation is weaker between the first algorithm and any of the others; *(iii)* in some cases, the differences between the second and the third algorithms may be significant, so a priority must be set by the decision-maker; *(iv)* the fact that the algorithms are so fast enables the running of the third one considering different probabilities, which provides a deeper insight of the relationship between probability (related to the risk-aversion, *i.e.*, how sure the decision-maker wants to be about finishing at a given deadline or before) and makespan percentile (*i.e.*, how much time she needs to start before the deadline); *(v)* the effect of using different seeds is significant; and *(vi)* the makespan percentile linearly depends on the number of factories, jobs, machines, and the interaction between the number of factories and machines.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Economy and Competitiveness and FEDER (TRA2015-71883-REDT, CYTED2014-P514RT0013, DPI2013-44461-P, DPI2016-80750-P).

References

[1] I. Al Kattan, R. Maragoud, Performance analysis of flowshop scheduling using genetic algorithm enhanced with simulation, Int. J. Ind. Eng.: Theory Appl. Pract. 15 (1) (2008) 62–72.
[2] A. Al-Salem, A heuristic to minimize makespan in proportional parallel flow shops, Int. J. Comput. Inf. Sci. 2 (2) (2004) 98–107.
[3] H. Allaoui, S. Lamouri, M. Lebbar, A Robustness Framework for a Stochastic Hybrid Flow Shop to Minimize the Makespan, Proceedings of the International Conference on Service Systems and Service Management, (2006), pp. 1097–1102. Troyes, France.
[4] K. Baker, D. Altheimer, Heuristic solution methods for the stochastic flow shop problem, Eur. J. Oper. Res. 216 (1) (2012) 172–177.
[5] K. Baker, D. Trietsch, Three heuristic procedures for the stochastic, two-machine flow shop problem, J. Scheduling 14 (5) (2011) 445–454.
[6] K.R. Baker, Introduction to Sequencing and Scheduling, John Wiley & Sons, 1974.
[7] J. Bhimani, N. Mi, M. Leeser, Z. Yang, Fim: performance prediction for parallel computation in iterative data processing applications, 10th IEEE International Conference on Cloud Computing (CLOUD). IEEE, 2017, (2017).
[8] G. Cabrera, A. Juan, D. Lázaro, J.M. Marquès, I. Proskurnia, A simulation-optimization approach to deploy internet services in large-scale systems with user-provided resources, Simulation 90 (6) (2014) 644–659.
[9] H. Campbell, R. Dudek, M. Smith, Heuristic algorithm for the n job, m machine sequencing problem, Manage. Sci. 16 (10) (1970) 630–637.
[10] D. Cao, M. Chen, Parallel flowshop scheduling using tabu search, Int. J. Prod. Res. 41 (13) (2003) 3059–3073.
[11] F.T. Chan, S. Chung, P. Chan, An adaptive genetic algorithm with dominated genes for distributed scheduling problems, Expert Syst. Appl. 29 (2) (2005) 364–371.
[12] S. Choi, K. Wang, Flexible flow shop scheduling with stochastic processing times: a decomposition-based approach, Comput. Ind. Eng. 63 (2) (2012) 362–373.
[13] W. David, A. Kusiak, A. Artiba, A scheduling problem in glass manufacturing, IIE Trans. (Inst. Ind. Eng.) 28 (2) (1996) 129–139.
[14] J. de Armas, A.A. Juan, J.M. Marquès, J.P. Pedroso, Solving the deterministic and stochastic uncapacitated facility location problem: from a heuristic to a simheuristic, J. Oper. Res. Soc. 68 (10) (2017) 1161–1176.
[15] V. Fernandez-Viagas, J. Framinan, A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 53 (4) (2015) 1111–1123.
[16] V. Fernandez-Viagas, J.M. Framinan, Efficient non-population-based algorithms for the permutation flowshop scheduling problem with makespan minimisation subject to a maximum tardiness, Comput. Oper. Res. 64 (2015) 86–96.
[17] V. Fernandez-Viagas, J.M. Framinan, NEH-Based heuristics for the permutation flowshop scheduling problem to minimise total tardiness, Comput. Oper. Res. 60 (2015) 27–36.
[18] C. Fikar, A.A. Juan, E. Martinez, P. Hirsch, A discrete-event driven metaheuristic for dynamic home service routing with synchronised trip sharing, Eur. J. Ind. Eng. 10 (3) (2016) 323–340.
[19] J.M. Framinan, P. Perez-Gonzalez, On heuristic solutions for the stochastic flowshop scheduling problem, Eur. J. Oper. Res. 246 (2) (2015) 413–420.
[20] J. Gao, R. Chen, A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem, Int. J. Comput. Intell. Syst. 4 (4) (2011) 497–508.
[21] J. Gao, R. Chen, W. Deng, An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (3) (2013) 641–651.
[22] M.R. Garey, D.S. Johnson, R. Sethi, The complexity of flowshop and jobshop scheduling, Math. Oper. Res. 1 (2) (1976) 117–129.
[23] F. Ghezail, H. Pierreval, S. Hajri-Gabouj, Analysis of robustness in proactive scheduling: a graphical approach, Comput. Ind. Eng. 58 (2) (2010) 193–198.
[24] S. Gonzalez-Martin, A.A. Juan, D. Riera, M.G. Elizondo, J.J. Ramos, A simheuristic algorithm for solving the arc routing problem with stochastic demands, J. Simul. (2017) 1–14.
[25] E.M. Gonzalez-Neira, D. Ferone, S. Hatami, A.A. Juan, A biased-randomized simheuristic for the distributed assembly permutation flowshop problem with stochastic processing times, Simul. Modell. Pract. Theory 79 (2017) 23–36.
[26] A. Grasas, A.A. Juan, H.R. Lourenço, Simils: a simulation-based extension of the iterated local search metaheuristic for stochastic combinatorial optimization, J. Simul. 10 (1) (2016) 69–77.
[27] A. Gruler, C.L. Quintero-Araújo, L. Calvet, A.A. Juan, Waste collection under uncertainty: a simheuristic based on variable neighbourhood search, European Journal of Industrial Engineering 11 (2) (2017) 228–255.
[28] S. Hatami, R. Ruiz, C. Andrés-Romano, The distributed assembly permutation flowshop scheduling problem, Int. J. Prod. Res. 51 (17) (2013) 5292–5308.

[29] S. Hatami, R. Ruiz, C. Andrés-Romano, Heuristics and metaheuristics for the distributed assembly permutation flowshop scheduling problem with sequence dependent setup times, Int. J. Prod. Econ. 169 (2015) 76–88.
[30] W. Hopp, M. Spearman, Factory Physics, 3rd Ed., Irwin, Chicago, 2008.
[31] C.Y. Hsu, P.C. Chang, M.H. Chen, A linkage mining in block-based evolutionary algorithm for permutation flowshop scheduling problem, Comput. Ind. Eng. 83 (2015) 159–171.
[32] S. Johnson, Optimal two- and three-stage production schedules with setup times included, Nav.Res.Logist.Q. 1 (1954) 61–68.
[33] A.A. Juan, B.B. Barrios, E. Vallada, D. Riera, J. Jorba, A simheuristic algorithm for solving the permutation flow shop problem with stochastic processing times, Simul. Modell. Pract. Theory 46 (2014) 101–117.
[34] A.A. Juan, J. Faulin, S.E. Grasman, M. Rabe, G. Figueira, A review of simheuristics: extending metaheuristics to deal with stochastic combinatorial optimization problems, Oper. Res. Perspect. 2 (2015) 62–72.
[35] A.A. Juan, J. Faulin, J. Jorba, J. Caceres, J.M. Marquès, Using parallel & distributed computing for real-time solving of vehicle routing problems with stochastic demands, Ann Oper Res 207 (1) (2013) 43–65.
[36] A.A. Juan, H.R. Lourenço, M. Mateo, R. Luo, Q. Castella, Using iterated local search for solving the flow-shop problem: parallelization, parametrization, and randomization issues, Int. Trans. Oper. Res. 21 (1) (2014) 103–126.
[37] K.B. Kahn, S.E. Kay, R.J. Slotegraaf, S. Uban, The PDMA Handbook of New Product Development, 3rd ed., John Wiley & Sons, Inc, Hoboken, New Jersey, 2013.
[38] K. Katragjini, E. Vallada, R. Ruiz, Flow shop rescheduling under different types of disruption, Int. J. Prod. Res. 51 (3) (2013) 780–797.
[39] K. Kianfar, S. Fatemi Ghomi, A. Oroojlooy Jadid, Study of stochastic sequence-dependent flexible flow shop via developing a dispatching rule and a hybrid ga, Eng. Appl. Artif. Intell. 25 (3) (2012) 494–506.
[40] Q. Lin, L. Gao, X. Li, C. Zhang, A hybrid backtracking search algorithm for permutation flow-shop scheduling problem, Comput. Ind. Eng. 85 (2015) 437–446.
[41] S.W. Lin, K.C. Ying, C.Y. Huang, Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm, Int. J. Prod. Res. 51 (16) (2013) 5029–5038.
[42] Q. Liu, S. Ullah, C. Zhang, An improved genetic algorithm for robust permutation flowshop scheduling, Int. J. Adv. Manuf. Technol. 56 (1–4) (2011) 345–354.
[43] H.R. Lourenço, O.C. Martin, T. Stützle, Iterated Local Search, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, Kluwer Academic Publishers, Norwell, MA, 2002, pp. 321–353.
[44] C. Moon, J. Kim, S. Hur, Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain, Comput. Ind. Eng. 43 (1) (2002) 331–349.
[45] B. Naderi, R. Ruiz, The distributed permutation flowshop scheduling problem, Comput. Oper. Res. 37 (4) (2010) 754–768.
[46] B. Naderi, R. Ruiz, A scatter search algorithm for the distributed permutation flowshop scheduling problem, Eur. J. Oper. Res. 239 (2) (2014) 323–334.
[47] M. Nawaz, E. Enscore Jr., I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega 11 (1) (1983) 91–95.
[48] A. Pagès-Bernaus, H. Ramalhinho, A.A. Juan, L. Calvet, Designing e-commerce supply chains: a stochastic facility–location approach, Int. Trans. Oper. Res. (2017), http://dx.doi.org/10.1111/itor.12433.
[49] Q.K. Pan, R. Ruiz, An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem, Omega 44 (2014) 41–50.
[50] J. Panadero, J. Doering, R. Kizys, A. Juan, A. Fito, A variable neighborhood search simheuristic for project portfolio selection under uncertainty, J. Heuristics (2018).
[51] I. Ribas, R. Companys, X. Tort-Martorell, Efficient heuristics for the parallel blocking flow shop scheduling problem, Expert Syst. Appl. 74 (2017) 41–54.
[52] F.A. Rodammer, K.P. White, A recent survey of production scheduling, IEEE Trans. Syst. Man Cybern. 18 (6) (1988) 841–851.
[53] B. Roy, Robustness in operational research and decision aiding: a multi-faceted issue, Eur. J. Oper. Res. 200 (3) (2010) 629–638.
[54] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, Eur. J. Oper. Res. 177 (3) (2007) 2033–2049.
[55] A. Sluga, P. Butala, G. Bervar, A multi-agent approach to process planning and fabrication in distributed manufacturing, Comput. Ind. Eng. 35 (3) (1998) 455–458.
[56] E. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, Eur. J. Oper. Res. 47 (1) (1990) 65–74.
[57] E. Taillard, Benchmarks for basic scheduling problems, Eur. J. Oper. Res. 64 (2) (1993) 278–285.
[58] G. Vairaktarakis, M. Elhafsi, The use of flowlines to simplify routing complexity in two-stage flowshops, IIE Trans. (Inst. Ind.Eng.) 32 (8) (2000) 687–699.
[59] B. Wang, Integrated Product, Process and Enterprise Design, Chapman & Hall, London, 1997.
[60] C. Wang, D. Mu, F. Zhao, J.W. Sutherland, A parallel simulated annealing method for the vehicle routing problem with simultaneous pickup-delivery and time windows, Comput. Ind. Eng. 83 (2015) 111–122.
[61] L. Wang, W. Shen, Q. Hao, An overview of distributed process planning and its integration with scheduling, Int. J. Comput. Appl. Technol. 26 (1/2) (2006) 3.
[62] V.F. Yu, S.Y. Lin, A simulated annealing heuristic for the open location-routing problem, Comput. Oper. Res. 62 (2015) 184–196.
[63] X. Zhang, S. Van De Velde, Approximation algorithms for the parallel flow shop problem, Eur. J. Oper. Res. 216 (3) (2012) 544–552.
[64] Q. Zhou, X. Cui, Research on Multi-objective Flow Shop Scheduling with Stochastic Processing Times and Machine Breakdowns, Proceedings of International Conference on Service Operations and Logistics and Informatics, (2008), pp. 1718–1724. Troyes, France.