Computers & Operations Research 40 (2013) 1979–1990

Contents lists available at SciVerse ScienceDirect

Computers & Operations Research

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

Taking advantage of a diverse set of efficient production schedules: A two-step approach for scheduling with side concerns

Selçuk Gören [a,*], Henri Pierreval [b]

[a] Abdullah Gül University, Faculty of Engineering and Natural Sciences, Department of Industrial Engineering, Aşık Veysel Bulvarı, Erciyes Teknopark, No: 4/67-A, 38039, Melikgazi, Kayseri, Turkey

[b] Clermont University, IFMA, LIMOS, UMR CNRS 6158, Campus de Clermont-Ferrand, Les Cézeaux, F-63175 Aubière, Cedex, France

ARTICLE INFO

Available online 4 March 2013

Keywords:
Multimodal optimization
Hybrid flow shop
Genetic algorithm
Robustness
Stability
Production scheduling
Disruptions
Uncertainty

ABSTRACT

In many practical scheduling problems, the concerns of the decision-maker may not be all known in advance and therefore may not be included in the initial problem definition as an objective function and/or as constraints. In such a case, the usual techniques of multi-objective optimization become inapplicable. To cope with this problem and to facilitate handling the concerns of the decision-maker, which can be implicit or qualitative, a dedicated methodological framework is needed. In this paper we propose a new two-step framework. First, we aim at obtaining a set of schedules that can be considered efficient with respect to a performance measure and at the same time different enough from one another to enable flexibility in the final choice. We formalize this new problem and suggest to address it with a multimodal optimization approach. Niching considerations are discussed for common scheduling problems. Through the flexibility induced with this approach, the additional considerations can be taken into account in a second step, which allows decision-makers to select an appropriate schedule among a set of sound schedules (in contrast to common optimization approaches, where usually a single solution is obtained and it is final). The proposed two-step approach can be used to handle a wide range of underlying scheduling problems. To show its potential and benefits we illustrate the framework on a set of hybrid flow shop instances that have been previously studied in the literature. We develop a multimodal genetic algorithm that employs an adapted version of the restricted tournament selection for niching purposes in the first step. The second step takes into account additional concerns of the decision-maker related to the ability of the schedules to absorb the negative effects due to random machine breakdowns. Our computational experiments indicate that the proposed framework is capable of generating numerous high-performance (mostly optimal) schedules. Additionally, our computational results demonstrate that the proposed framework provides the decision-maker a high flexibility in dealing with subsequent side concerns, since there are drastic differences in the capabilities of the efficient solutions found in Step 1 to absorb the negative impacts of machine breakdowns.

1. Introduction

Production scheduling is the process of deciding how to allocate limited resources to a variety of competing tasks. There has been a vast body of research on production scheduling problems since their original formulation in late 1950s. Even though these formulations typically involve optimizing certain aspect of system performance (articulated as an objective function or *metric*), the issue of what metrics to use in assessing the quality of a schedule is far from being trivial. General practice is to optimize a performance measure (usually an increasing function of job completion times, i.e., a *regular* measure) which is deemed to address the most important aspects of the shop floor performance. We call this measure the *principal measure* throughout this paper.

Despite most existing research concentrates on optimizing one or more principal measures, in certain environments in practice, there is more to scheduling than what is generally considered in the literature. An example is the environments where there are *side concerns* in addition to the principal performance measure, such as the schedule's conformance to omitted constraints in the modelling phase, economies of the schedule's implementation, the ability of the shop floor to position the work, tooling and operators in a way that will smooth the execution of the schedule, the capacity allocation between competing production lines, and

* Corresponding author. Tel.: +90 352 224 88 00.
E-mail addresses: selcuk.goren@agu.edu.tr, selcuk.goren@gmail.com (S. Gören), henri.pierreval@ifma.fr (H. Pierreval).

0305-0548/$ - see front matter © 2013 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.cor.2013.02.016

the ability of the schedule to absorb the negative effects of the unforeseen future disruptions. Another example is the case where a number of users (from different departments of the company, or even from different companies in the supply chain) with diverse individual agenda should decide on a production schedule collectively. These issues are discussed in more detail in Section 2. In such environments, it may be too difficult, if not impossible, to consider everything in advance, in the schedule generation phase. Moreover, some concerns of the decision-maker may be intrinsically qualitative and therefore it may difficult to come up with exact mathematical formulations that express the concerns properly as objective functions and/or constraints [5]. Consequently, the well-known techniques of multi-objective scheduling (see e.g., [42]) may become inapplicable.

Despite its obvious pertinence to scheduling practice, this problem has not been considered in the literature to the best of our knowledge. In this paper, we tackle with this problem using a divide and conquer policy. Specifically, we propose a new framework that divides the problem into two sub-problems by deferring the detailed consideration of the side concerns to alleviate this difficulty. In the first sub-problem, the aim is to obtain several alternative schedules with good values of the principal performance measure. The objective of the second sub-problem is to choose a schedule that can be found satisfactory in terms of the side concerns among those alternatives.

We suggest to address the first sub-problem of finding a set of alternatives with good values of principal performance as a *multimodal optimization* problem. Multimodal optimization deals with searching all the local optima of a multimodal function. Classical techniques of optimization would need multiple restart points and multiple runs in the hope that a different solution may be discovered every run, with no guarantee however. Evolutionary algorithms (EAs), due to their population based approach, provide a natural advantage. The challenge of the using EAs for multimodal optimization is to find and to maintain multiple local optima.

In this respect, *niching* is suggested by many authors (e.g., [48]), and the references therein). Niching is a generic term referred to as the technique of finding and preserving multiple stable niches, or favourable parts of the solution landscape possibly around local optima, so as to prevent convergence to a single solution.

Evolutionary algorithms are known to be efficient in the scheduling area (see e.g., [8]), and for multimodal optimization [41]. However, multimodal scheduling problem seems to have been little addressed in the literature. Among the possible reasons for this, we note that many niching techniques in the literature require a distance metric to discover whether two individuals occupy the same niche or not. The majority of the niching algorithms use Euclidean distance for this purpose. Although this metric is appropriate when the search space is $\mathbb{R}^n$, it is generally not suitable for the search spaces encountered in combinatorial optimization problems. Therefore, the multimodal evolutionary algorithms in the literature are almost exclusively developed for the optimization of functions from $\mathbb{R}^n$ to $\mathbb{R}$. In Section 3 we demonstrate how the permutation theory can be used to measure the distance between individuals for the problem handled in this paper. We also illustrate the second sub-problem of considering side concerns on the specific example of generating robust or stable schedules in the face of random machine breakdowns in Section 4.

The rest of this paper is organized as follows: In Section 2, we give the rationale behind the proposed two-step approach and define the problem addressed in this paper. Section 3 illustrates our approach on a hybrid flow shop problem. We propose a multimodal genetic algorithm to employ in the first step. We deal with the second sub-problem and illustrate our approach on a specific example of generating robust or stable schedules in the face of random machine breakdowns in Section 4. We test the performance of the proposed algorithm in Section 5. Finally, we make our concluding remarks and identify possible future research directions in Section 6.

2. Motivation and problem definition

Scheduling is a decision-making process that is far from being isolated. It is in interaction with other decision-making processes such as production planning, order review/release, due date quotation, and lot sizing; for schedules are not just time/machine/operation assignments, but they are also programs that guide purchasing of raw material, negotiation of due-dates with the customers, tool, fixture and operator assignments, determination of work-force and overtime levels, and transportation of raw material and finished goods. Moreover, the implications of a schedule are not necessarily confined within the borders of a company. It is increasingly becoming common for companies to share their production schedules with their suppliers with the trend towards increasing collaboration between the elements of supply chains, as noted in [25]. In other words, the schedule of a shop floor has implications that are collectively experienced by a number of different persons with different responsibilities. We refer to them as *partners* in the rest of the article.

Kempf et al. [25] discuss the issue of assessing the quality of production schedules (and proper metric selection) in detail. A significant part of the difficulties encountered in such an endeavor are attributed to the existence of numerous uses for a production schedule. Aytug et al. [4] also examine the ways in which manufacturing organizations use production schedules with a perspective of executional uncertainties and come up with several purposes. The different functions of schedules that the authors identify range from "the use as a capacity checking tool for a higher-level production planning system" to "providing visibility of future plans within the shop floor" or to "providing degrees of freedom for reactive scheduling". The authors conclude that a schedule is often used for different purposes by different partners, which are often trying to achieve different goals. The decision-maker should take into account these different partners and their goals in order to generate valid and applicable schedules.

The well-known techniques of multicriteria scheduling immediately comes to one's mind when encountered with such a situation. Unfortunately, the concerns of different partners may not be all known in advance, can be implicit or qualitative, and therefore may not be included in the initial problem definition as an objective function and/or as constraints. A reasonable approach to alleviate this problem can be to provide the partners with the flexibility and freedom of choice introduced by generating more than a single alternative schedule. Such an approach would significantly increase the chance of obtaining a schedule that is found acceptable and reasonable by all the involved partners. In fact, it is beneficial to have a choice between more than one alternative regardless of the particular goals of the partners. If the schedule is used as a capacity check for higher-level reasoning, for example, the alternative schedules can be compared with each other in terms of peak load intervals to select the fittest solution. On the other hand, if the schedule is used to provide visibility of future plans, the alternative schedules can be compared in terms of the ease to bring the system and the resources in line with a new schedule (system reconfiguration in words of [4]) in case of disruptions. Note that it is not always possible to come up with a well-defined mathematical expression

to measure the schedule's reconfigurability, and the best option can be to resort to the expert opinion, i.e., active involvement of the decision-maker. The decision-maker may know which schedule is preferable when presented with a few alternatives, but may find it hard to express this preference in a principal measure [22].

The goals of different partners may be conflicting with each other. For instance, while a customer representative may be looking for a schedule with better delivery performance, a representative from the manufacturing department may prefer a schedule with fewer set ups and longer production runs, yet a representative from the finance department may opt for a schedule that involves as little work-in-progress inventory as possible. Still, having more than one alternative increases the chance to agree on a compromise, whereas having only one schedule at hand may force some groups to reluctantly accept an unsatisfactory (at least from their perspective) schedule.

Moreover, it may be insufficient to generate only one schedule from a practical point of view. Practitioners and researchers agree that it is very difficult, if not impossible, to model stochastic and dynamically changing state of the shop floor exactly. Models solved to generate production schedules often include (sometimes implicitly) several assumptions and tend to omit various aspects of the shop floor for the sake of simplicity. Even though several alternative solutions may perform similarly in terms of the principal measure that is being optimized, their performance may differ drastically with respect to conformance to omitted constraints, economies of implementation, the ability of the shop floor to position the work, tooling and operators in a way that will smooth the execution of the schedule, the capacity allocation between competing production lines, or the decision-maker's preferences (which are not necessarily always quantitative) and expert opinions. In fact, the shop floor personnel are often inclined to override the schedule generated by an automated decision-making tool, such as an expert system, not because of the schedule's performance in terms of the principal metric but because of its performance in terms of these side concerns. Providing several schedules instead of just one and letting the human decision-makers actively involve in making the final decision may also enable efficiency. Such an interactive decision-making process would comfort the shop floor personnel and would make it easier for them to adhere to the schedule *they chose*.

To sum up, it is desirable to provide the decision-maker with several good solutions, instead of just one. Nonetheless, most studies in the literature generally concentrate on generating a single schedule and therefore overlook the aforementioned benefits of generating several good solutions. We should note that the idea of generating a family of schedules has already been considered in the literature. For example, Artigues et al. [3] extend the "process first schedule later scheme" in [45] to generate a family of schedules. The families of schedules are represented by ordered group assignments that define a sequence of groups for each machine such that the operations within every group are totally permutable. The authors study on minimizing the number of groups and on maximizing the number of schedules represented by the groups. The aim is to cope with uncertainty by generating several similar (so-called *compatible*) schedules so that the decision-maker can easily switch from a schedule to another to react to disruptions if need arises. In contrast to such a schedule multiplicity, in this paper, we concern ourselves with generating *diverse* rather than compatible alternatives. We want the schedules to be significantly different from one another to provide the decision-makers the true benefit of choice. The existence of different alternatives is the prerequisite to the concept of choice: if alternatives are too similar, there is no selection in the real sense of the word.

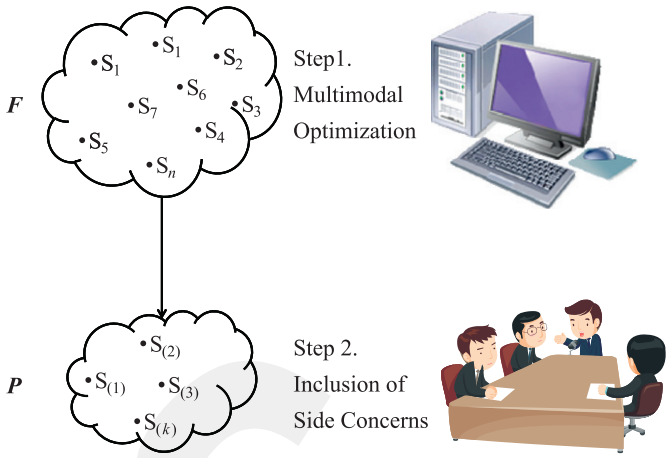

Fig. 1. The global framework of the proposed approach.

2.1. Formal problem definition

In the light of the above discussions, an appropriate solution methodology should satisfy two objectives when scheduling for a shop floor when there are side concerns in addition to the primal performance metric(s): provide a diverse set of schedules that perform well (i.e., optimal or near-optimal), and ensure that the selected schedule is in line with the side concerns. In this paper, we propose a two-step approach where each step aims at achieving one of these objectives. In the first step, we obtain a diverse set of promising solutions, all of which perform well in terms of the principal metric. In the second step, we analyse the promising solutions obtained in the previous step in terms of their compliance with the side concerns of the decision-maker. Specifically, assume that the feasible region to the scheduling problem at hand is denoted by $\mathbb{F}$, and let $z=f(S), S\in\mathbb{F}$, denote the performance of a schedule S in terms of the principal metric f. Let z^* denote the optimal value of the principal measure, i.e., $z^*=f(\tilde{S})$ where $\tilde{S}\in\arg\min_{S\in\mathbb{F}} f(S)$. In the first step, we aim at obtaining a diverse set of promising solutions $\mathbb{P}\subset\mathbb{F}$, with the property $f(S)-z^*\leq\varepsilon, \forall S\in\mathbb{P}$, where $\varepsilon>0$ is a small number in $\mathbb{R}$. Employing multimodal optimization can be an interesting way to address this problem. In the second step, each $S\in\mathbb{P}$ is analyzed in terms of side concerns. In Section 4, we illustrate this step on a specific example of a hybrid flow shop subject to random machine breakdowns, where the side concern of the decision-maker is the ability of the generated schedule to absorb the negative effects of unforeseen random machine breakdowns. Fig. 1 summarizes the proposed global framework.

3. Sub-problem 1–obtaining the alternatives: illustration on a hybrid flow shop

In this section, we illustrate the proposed two-step approach on a hybrid flow shop. This problem is chosen because hybrid flow shops are quite common, especially in the process industry where multiple machines are available at each stage as well as in certain flexible manufacturing environments. A hybrid flow shop can be seen as an extension of two classical scheduling environments: the classical flow shop and identical parallel-machine environments. To precisely define the hybrid flow shop scheduling problem, assume that a set $J=\{1,2,\ldots,n\}$ of n simultaneously available jobs must be sequentially processed on a set of k stages. Each job is processed first at stage 1, then at stage 2, …, and finally at stage k. At stage i, M_i identical parallel machines are available. Each job $j\in J$ can only be processed on one machine at

a time and consists of k operations (O_{j1},O_{j2} …,O_{jk}). An operation O_{ji} has a processing time p_{ij} and has to be processed without preemption on only one of the machines at stage i. The objective is to find a schedule which minimizes the maximum completion time or makespan defined as the elapsed time from the start of the first operation of the first job at stage 1 to the completion of the last operation of the last job at stage k.

If there are only two stages and there is a single machine at each stage, the above hybrid flow shop problem is solvable in $O(n \log n)$ time by the well-known algorithm developed by Johnson [24]. Other than this case, the hybrid flow shop scheduling problem is NP-hard in the strong sense even if there are only two available machines at one of the stages [23].

The first research papers about hybrid flow shop scheduling appear in the 1970s. Salvador [39] published one of the pioneer papers on hybrid flow shops by modelling the production system in the synthetic fiber industry as a no-wait hybrid flow shop. In 1979, Garey and Johnson [14] showed that the hybrid flow shop problem with makespan objective is NP-complete, and along the years the NP-nature of most hybrid flow shop scheduling problems has been shown. Therefore, several exact methods as well as a large number of heuristics and approximation algorithms have been proposed for different hybrid flow shop configurations. During the last decade, research in this area has focused on more realistic problems, including sequence-dependent setups on machines, machine eligibility, time lags on operations, precedence constraints among jobs, etc. in order to bridge the gap between theory and practice. We refer the reader to [29,35,37,44] for a detailed and extensive review of the state of the art.

The objective of the first sub-problem is to obtain a diverse set of schedules with good makespan values. This objective can be achieved using multimodal optimization techniques. The algorithm class of our choice is the genetic algorithms because of two reasons: (1) genetic algorithms have been successfully applied to generate good production schedules in the scheduling literature and (2) niching techniques to achieve multimodality have been thoroughly investigated in the evolutionary algorithms literature. In the next section, we give the details of the proposed multimodal genetic algorithm.

3.1. Step 1: the proposed genetic algorithm

Genetics algorithms have been successfully applied to production scheduling problems throughout the years [12]. For a review of studies that are specific to the hybrid flow shop environment, see [35,37]. In this section, we present the details of our implementation, which can also handle the diversity requirement for the first sub-problem.

Initial population: The initial population consists of *pop_size* randomly generated individuals and *pop_size* is kept constant through the generations.

Encoding of individuals: For scheduling applications, the encoding scheme most frequently used in the literature is simply the sequence of jobs. For hybrid flow shops, considering the fact that different permutations of jobs may occur at different stages, a logical encoding of a solution can have k strings, each being a permutation of 1,2, …,n, corresponding to the job list at different stages. As also noted in [6,33], however, one can easily observe that a free and random search can only be conducted at the first stage for different permutations of jobs, where all jobs are available. For the subsequent stages, permutations that deviate from the permutation of the preceding stage may result in a schedule with considerable unnecessary waiting times. Hence, we consider the sequence of jobs only at stage 1 in the encoding of a solution. We then decode each individual to a full schedule by using a List Scheduling algorithm. In this encoding, a string of n integers is used to represent an individual. Such a string π corresponds to the job list at stage 1 and is a permutation of 1,2, …,n. Each integer in the permutation, representing a job, is called a gene in genetic algorithm terminology. We use the notation π_i to indicate the gene at position i of chromosome π. Similarly, let $p_\pi(i)$ denote the position at which gene i is located in π.

Decoding of individuals: An individual is decoded to a full schedule by employing the List Scheduling algorithm proposed by Graham [18]. List Scheduling algorithm constructs the schedule by iteratively assigning the jobs to the machines according to the list at the first stage such that job completion times are minimized. Then, a new list is obtained based on an ascending order of the completion times in the first stage. At the second stage, the jobs are assigned to the machines according to the new list and so on. It should be noted that this decoding scheme preserves the sequence of jobs in the list while assigning the jobs to the machines at a stage. This implies that a job will not be scheduled before the job that precedes it in the list even though there are enough machines and time available. On the other hand, since there are multiple machines at a stage, two jobs may start at the same time if there are enough machines available even though they are not at the same position in the list. Observe that List Scheduling algorithm creates a semi-active schedule since jobs are scheduled as soon as possible by preserving the job order defined by the list. Furthermore, List Scheduling algorithm does not necessarily yield a permutation schedule.

We illustrate the decoding scheme with a small numerical example. Consider five jobs to be scheduled in a two-stage hybrid flow shop, with two identical machines at both stages. The processing times are given in Table 1.

Assume that the sequence (1, 2, 3, 4, 5) is decoded. The resulting Gantt-chart is given in Fig. 2. Observe that the list for the second stage is (1, 2, 4, 3, 5) so we do not only consider permutation schedules.

Fitness: The fitness of an individual is taken as the makespan (C_{max}) value of the corresponding full schedule obtained by the List Scheduling algorithm as described above.

Reproduction: The population is arranged in *pop_size*/2 random couples and each couple is eligible for reproduction. New solutions (offspring) for the next generation are obtained by applying crossover and mutation genetic operators to this mating pool. In employing crossover operator, the objective is to introduce improvement and to explore more of the fitness landscape. In other words, crossover helps genetic algorithm to converge to the optimal solution. Mutation introduces variability and diversity to the population so that the search can escape from local optima. Related to these two genetic operators, we define the crossover rate (c), which is the probability of applying the crossover operator to the parents, and the mutation rate (m), which is the probability of applying the mutation operator to a child.

Crossover: We randomly select one of the parents and call it $P1$. The other parent is called $P2$. Our crossover operator starts with the first step of the Similar Job Order Crossover (SJOX) operator proposed in [36]. In other words, we examine both $P1$ and $P2$ gene by gene and the identical jobs occupying the same positions of the both parents are directly copied to the offspring into that position. We continue by the New Crossover Operator (NXO) proposed in

Table 1
Decoding example.

j	1	2	3	4	5
p_{1j}	1	3	5	2	5
p_{2j}	3	4	3	4	1

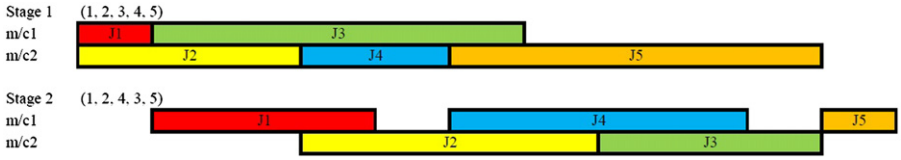

Fig. 2. Decoding example.

[33] to fill the remaining genes of the offspring. A gene of a parent is said to be available if this gene is not part of the offspring yet. Similarly, a gene of a parent is not available if this gene is already part of the offspring or if the previous gene is the last gene of the parent. Furthermore a gene is said to fit better if its total processing time requirement is less. NXO begins by copying the first available gene of $P1$ to the offspring. This gene is named the selected gene. Then, each job is scheduled one by one until the offspring is fully specified. The next gene of the selected gene in $P1$ and $P2$ are called *next_gene*1 and *next_gene*2. If both *next_gene*1 and *next_gene*2 are available, the fitter one is copied to the offspring. If only one of the *next_gene*1 and *next_gene*2 is available and the other one is unavailable, the available one is copied to the offspring. Finally, if none of the *next_gene*1 and *next_gene*2 is available, for each parent, next available genes (starting from the positions of *next_gene*1 and *next_gene*2, respectively, and returning to the first gene if the last gene is reached during the search) are found and fitter of them is copied to the offspring. The copied gene is the new selected gene and the process continues in the same fashion until the whole offspring is obtained. The other offspring is obtained in the same way starting with $P2$ instead of $P1$.

We consider the following example to illustrate the construction of the offspring according to this crossover operator. Consider a five-job problem with the following parent chromosomes: $P1=(3,5,1,2,4)$ and $P2=(4,2,1,5,3)$. We illustrate the construction of the child when we begin with $P1$. The other child is obtained starting with $P2$ instead.

1. Since gene 1 is in the third position in both parents, we begin with copying gene 1 into the third position in the child. Child: (*, *, 1, *, *).
2. We start with $P1$ as explained above, hence the first gene of the child is 3. Child: (3, *, 1, *, *).
3. *next_gene*1 $\leftarrow 5$, *next_gene*2 is not available since 3 is the last gene in $P2$. We select 5 as the next gene of the child. Child: (3, 5, 1, *, *).
4. *next_gene*1 is not available because 1 is already part of the child. *next_gene*2 is not available either because 3 is also in the child. We search $P1$ and $P2$ for next available genes starting from 1 and 3, respectively. Hence, *next_gene*1 $\leftarrow 2$ and *next_gene*2 $\leftarrow 4$. Assume that the total processing time requirement of job 4 is less than that of job 2. Then, job 4 is fitter than job 2 and job 4 becomes part of the child. Child: (3, 5, 1, 4, *).
5. *next_gene*1 = *next_gene*2 = 2. Since 2 is available, it is the selected gene. Child (3, 5, 1, 4, 2).

Mutation: We use insertion mutation operator, which is widely used in the literature. This mutation operator simply randomly selects a gene and inserts it in a random position.

Selection and niching: The fundamental feature of the first subproblem is the requirement of *multiple* efficient solutions that are sufficiently *different* from one another. As a consequence, it is necessary to manage a multimodal optimization process. In the domain of evolutionary algorithms, this issue can be handled using niching, which aims at reducing the effect of genetic drift (change in the relative frequency in which a gene occurs in a population due to random sampling and chance) resulting from the selection operator and at converging to multiple, highly fit, and significantly different solutions. Niching algorithms and techniques constitute an important research area in genetic and evolutionary computation. A wide range of niching approaches have been investigated in the literature, including sharing (e.g., [9,16,17,34]), crowding (e.g., [2,10,19,30,31]) and clustering (e.g., [1,21,47]). Sharing algorithms introduce niches by dividing the population into different subpopulations according to the similarity of the individuals. When a certain number of individuals occupy a particular niche, it becomes favorable for other individuals to search for a new niche available in the search space. Thus, sharing algorithms find an equilibrium between the number of individuals occupying a niche and the payoff of the niche. Sharing is usually implemented by degrading an individual's fitness due to the presence of other individuals in its neighborhood. The amount of degradation depends on the number of other individuals in the same niche and their proximity to the individual under consideration. Clustering algorithms follow the same line of reasoning. The population is divided into niches as usual but the fitness of an individual is calculated based on the distance between the individual and its niche centroid. This significantly reduces the time complexity of niching, compared to sharing. Crowding is motivated by analogy with competition for limited resources among similar members of a natural population. Dissimilar population members, often of differing species, occupy different environmental niches, and therefore do not typically compete for resources. On the other hand, similar individuals tend to occupy the same environmental niches and compete for the same limited resources. When a niche has reached its carrying capacity, weaker members of that niche will be crowded out of the population by stronger members. Crowding is generally implemented by setting up local tournaments between newly generated offspring and a certain number of similar parents.

In this study, we use a crowding algorithm, namely, the Restricted Tournament Selection [19] as the niching method. Harik [19] tests his method on several multimodal problems having the number of peaks varying from 5 to 32. The algorithm is shown to be able to maintain individuals at all the peaks, even though some of the peaks gradually lose a number of elements. It is also able to maintain all the global optima in a massively multimodal problem. Singh and Deb [41] also find restricted tournament selection very competitive in locating and maintaining multiple modes in their study. Restricted Tournament selection is easy to implement. Specifically, consider the couple of parents A and B, which are eligible to reproduce by applying crossover operator yielding two new individuals, A' and B' (after possibly applying mutation). A' and B' are then candidates to be placed into the population. For each of A' and B', we scan w (window size) more members of the population and pick the individual that most closely resembles the candidate from those w elements. Let these elements be called A'' and B''. A' then competes with A'' and if A' wins, it is allowed to enter the population and A'' is crowded out. A similar competition occurs

between B' and B''. This kind of tournament will restrict an entering individual from competing with others that are too different for the sake of maintaining diversity and preventing genetic drift.

To measure how close two individuals resemble each other, one needs a distance metric. The majority of the niching algorithms use Euclidean distance between two individuals for this purpose. Although this metric is appropriate for multimodal optimization of functions such as $f:\mathbb{R}^n \rightarrow \mathbb{R}$, the search space we consider in this study is composed of permutations of jobs $1,2,\ldots,n$, and the Euclidean distance does not make much sense. Several distance metrics from the permutation theory and algorithms to compute them are reviewed in [40]. The distance metric we propose in this study is the minimum number of swaps required to obtain a permutation starting from the other. This distance can be calculated in $O(n^2)$ time. To briefly review it, we have to first introduce permutation multiplication and inverse permutations. For the sake of notational convenience, we can see a permutation as a bijective function, where $\pi(i)=\pi_i$. Then the permutation product $(\pi \cdot \pi')$ is defined as the composition of the bijective functions π and π', that is $(\pi \cdot \pi')_i = \pi'(\pi(i))$. Let I denote the identity permutation, which corresponds to the identity function. Then for each permutation π, there exists a permutation π^{-1} such that $\pi \cdot \pi^{-1} = \pi^{-1} \cdot \pi = I$. Such a permutation can be obtained as $\pi_i^{-1} = p_\pi(i)$. Let $d(\pi,\pi')$ denote the distance between the permutations π and π'. Then we have $d(\pi,\pi') = d(\pi^{-1} \cdot \pi', I)$. The right hand side of the previous equation is the distance of the permutation $\pi^{-1} \cdot \pi'$ to the identity permutation I, which is simply the count of the couples of genes that have the wrong relative order in $\pi^{-1} \cdot \pi'$.

The proposed genetic algorithm can now be described as follows:

- Input
  - Parameters for genetic algorithm: *pop_size*,c,m,w, stopping criterion
  - Parameters for scheduling problem: n,k,m_i,p_{ij}: $i=1,2,\ldots,k$ and $j \in J$.
- Steps
  1. Generate initial population with pop size random individuals, where each individual will be a permutation of integers from 1 to n, representing a sequence of jobs at stage 1.
  2. Divide the population into *pop_size*/2 couples randomly.
  3. Mate couples according to crossover rate c by applying the crossover operator.
  4. Mutate new individuals by applying the mutation operator according to mutation rate m.
  5. For each offspring A', select window size w individuals from the population randomly.
  6. Find the closest individual A'' to A' among these w individuals.
  7. Use List Scheduling algorithm to decode A' and A'' into full schedules and then calculate their corresponding C_{max} values.
  8. If A' is not worse than A'', replace A'' with A'. Otherwise disregard the offspring A'.
  9. Apply steps 2–8 until the stopping criterion is met.

Note that even though the above algorithm is specifically developed for the hybrid flow shop problem, the proposed two-step framework is quite generic. The only assumption we make is that the underlying scheduling problem can be solved with an evolutionary algorithm. This is needed to be able to use niching approaches to handle diversity (which is convenient in evolutionary algorithms but other population-based metaheuristics such as particle swarm optimization can also be considered). For an EA, one needs to define a chromosome encoding and the evolutionary operators. One may think that this restricts the class of scheduling problems that can be addressed. However, many studies in the literature present evolutionary approaches to solve (sometimes quite complex) scheduling problems. These algorithms can be adapted to be used within the proposed two-step framework. Of course, one needs to verify that the distance metric we suggest is compatible with the new chromosome encoding scheme and elaborate on a new distance metric if needed. Nonetheless, as most encoding schemes in the literature involve the sequence of operations on machines, the idea of using distance between permutations as the distance metric remains convenient (possibly after small modifications to aggregate several permutation-distances into a single schedule-distance). Therefore other machine scheduling problems can be handled by the proposed methodology.

Having obtained a set of different enough efficient alternatives in this step, the problem is now reduced to selecting a schedule that can be found acceptable in terms of the side concerns of the decision-maker. This is the objective of the second sub-problem, which is handled in the next section.

4. Sub-problem 2-considering side concerns: illustration on an environment with random machine breakdowns

The final choice is made in this step after a thorough analysis of the alternatives obtained in the previous step. In other words, a solution is chosen on the basis of the decision-maker's expertise among the alternatives in $\mathbb{P} \subset \mathbb{F}$. Note that this choice is not based on the performances of the alternatives as they all have acceptable performance levels. Other considerations such as conformance to omitted constraints, economies of implementation, the ability of the shop floor to position the work, tooling and operators in a way that will smooth the execution of the schedule, the capacity allocation between competing production lines, etc. can play an important role in the final decision. Such types of considerations can be quite difficult to include in the problem formulation in advance.

In the hybrid flow shop illustration we consider in this paper, assume that, given the current conditions of the company, it is necessary to avoid the negative effects of the executional uncertainties present in the shop floor as much as possible. In other words, the company will not consider a schedule plausible if there is a risk of serious degradation in the quality of the schedule in the face of possible disruptions. Specifically, the primary concern of the decision-maker is to generate an efficient schedule, which is measured in terms of the makespan. But in addition to that, despite being secondary to have a good makespan performance, the decision-maker still wants the schedule to be as protected as possible against deterioration caused by executional uncertainties.

To handle this concern, we first take a closer look at the scheduling literature for the studies in the face of random disruptions. In this body of research, two new concepts, namely robustness and stability, are taken into account. The schedules are required to keep high performance (usually in terms of a principal metric) in the face of uncertainties. In other words, it is desired that their performance are insensitive to the negative impacts of the disruptions. This is the concern of robustness. It is also required that, when a schedule is executed in the shop floor, the realized schedule does not deviate much from its initial version. This is because many activities (besides production) are

planned based on the production schedule. Examples of such activities are placing orders for raw material, transportation of raw material or end products, negotiating due-dates with customers, planning workforce level, evaluating the amount of needed overtime, etc. It is important that the unforeseen disruptions affect the plans for these activities as little as possible. This is the concern of stability. Several measures to assess the robustness and stability of schedules have been developed in the literature. We refer the reader to [4,20,38,43] for a detailed review of the research on scheduling in the face of uncertainties. Sabuncuoglu and Goren [38] also review several robustness and stability measures.

4.1. The machine breakdown/repair process, robustness and stability measures

In the hybrid flow shop we consider in this paper, all $\sum_{i=1}^{k} M_i$ machines are subject to random breakdowns. The up times of machine m have independent and identical general distribution $G_{1m}(t)$. Similarly, the down times (i.e., the times that the machine is not in operation due to breakdown) are independent and identically distributed according to a general distribution $G_{2m}(t)$. We assume that all n jobs are released at time $t=0$. Let U_{m1}, U_{m2} … be the sequence of up times and D_{m1}, D_{m2} …be the sequence of down times for machine m. That is, machine m is operational from time 0 to U_{m1}, when the first breakdown occurs. The machine then takes time D_{m1} to be repaired and is again available for processing from time $U_{m1}+D_{m1}$ to time $U_{m1}+D_{m1}+U_{m2}$, and so on. We assume the right-shift rescheduling scheme, i.e., when the machine fails the operations are shifted to the right on the Gantt-chart a sufficient amount to just accommodate the repair duration if necessary. The work already performed on the affected operation is not lost and processing resumes from where it left off. We refer to the actual schedule that is executed on the shop floor as the realized schedule. Let $c_j(S)$ denote the completion time of job j in the initial schedule S and $C_j(S_r)$ denote the completion time of job j in the realized schedule. Note that $C_j(S_r)$ is a random variable.

Even though the proposed two-step frame is especially useful when the side concerns of the decision-maker cannot be expressed as closed-form mathematical functions, for the sake of simplicity and ease of illustration, we will use some measures to demonstrate the potential benefit of searching for multiple, different enough, good-performance solutions instead of just one. Specifically, we evaluate the robustness and stability performances of the schedules obtained in the first step from a proactive viewpoint with the help of four measures that are discussed in what follows.

Robustness measure R1: The majority of the recent studies that explicitly address robustness involve expected realized performance as a robustness measure. The expected realized performance can be a robustness measure by itself as in the study [45] or can be a part of it as implemented in [28]. The first robustness measure we consider is thus the expected makespan:

$$R1(S) = \mathrm{E}\left[\max_{j\in J}\{C_j(S_r)\}\right]. \quad (1)$$

Robustness measure R2: The other robustness measure we consider is based on *regret*. Regret is the opportunity cost in terms of principal measure that is incurred because the exact locations of breakdowns in the time horizon and the repair durations are unknown at the beginning of the planning horizon. If we knew the breakdown times and repair durations ahead of time, we would begin with an initial schedule S^* with the minimum realized makespan instead of S. The difference between the makespans of S and S^* is the regret associated with the schedule S. This measure is closely related to the minimax type of objective functions that are used in robust optimization literature (see [26]). Specifically,

$$R2(S) = \mathrm{E}\left[\max_{j\in J}\{C_j(S_r)\} - \max_{j\in J}\{C_j(S_r^*)\}\right], \quad (2)$$

where

$$S^* \in \arg\min_{S}\left\{\max_{j\in J}\{C_j(S_r)\}\right\}. \quad (3)$$

Stability measure S1: A stable schedule is one that does not deviate much from the initial schedule in the face of disruptions. The deviation is generally measured in terms of the differences between the job completion times in the initial and realized schedules. Hence, a typical stability measure is a non-decreasing function of the deviation of job completion times. The most frequently used function in the literature is the summation of absolute values, which is first proposed in [46]

$$S1(S) = \mathrm{E}\left[\sum_{j=1}^{n} |C_j(S_r) - c_j(S)|\right]. \quad (4)$$

Stability measure S2: The other stability measure that we use in this paper is the total variance of the realized completion times. It is obvious that the smaller the variance, the less the dispersion and hence, more stability, literally:

$$S2(S) = \sum_{j=1}^{n} \mathrm{Var}[C_j(S_r)]. \quad (5)$$

5. Computational experiments

Our computational study aims at analyzing the performance of the proposed genetic algorithm in terms of (1) minimizing the makespan and (2) identifying significantly diverse solutions. Related with the second aim, we illustrate the potential benefits of the proposed two-step framework in terms of side concerns using a specific example, where the machines are subject to random machine breakdown and the decision-maker concerns himself/herself with protecting the schedules against the negative effects of such breakdowns. The proposed genetic algorithm is coded in C++ and run on a notebook with an Intel Core i3-2310M processor and 2048 MB of memory running under Windows 7 Professional 32-bit edition. We take $pop\_size = 500, c = 50\%$, $m = 10\%, w = 100$ in our computational study. The proposed genetic algorithm is run for 1600 CPU seconds before stopping.

The test problems used in our experiments are the benchmark problems that are presented in [7]. The problem sizes varies from 10-job × 5-stage to 15-job × 10-stage. Processing times have a uniform distribution in the range of (3, 20). Three characteristics that define a problem are number of jobs, number of stages and number of identical machines at each stage. An instance is said to have the configuration $M_1 M_2 \ldots M_k$, where M_i is the number of identical parallel machines in stage i. The instances are divided into four categories (i.e., a, b, c and d) according to their machine configuration. The instances of types a, b, c and d have the configurations 33133, 13333, 33233, and 33333, respectively.

The same problems are previously studied in the literature. While a branch and bound (B&B) algorithm that is improved with the use of satisfiability tests and time-bound adjustments is used in [32] an artificial immune system (AIS) is adopted in [11] to solve the same problem instances. Both studies limit their CPU time by 1600 s. If an optimal solution is not found within 1600 s, the search is stopped and the best solution is accepted as the final schedule. The authors in both studies present the lower bounds for the problem instances and the relative gap from the lower

bounds (*LB*s). In our study, we also calculate this optimality gap, which is formulated as

$$\%Gap = 100 \times \frac{C^*_{max} - LB}{LB},$$

where C^*_{max} is the best makespan found by the algorithm and *LB* is the lower bound. The comparison of the proposed multimodal genetic algorithm (GA) with the AIS and B&B algorithms is summarized in Tables 2 and 3.

In Tables 2 and 3, the column entitled "#" indicates the number of different solutions in the final population of the proposed genetic algorithm (i.e., $\mathbb{P}$). The third and fourth columns show the best and the worst makespan values, respectively, in $\mathbb{P}$. The fifth column presents the percentage gap between the best and the worst makespan values (i.e., $100 \times (\max C_{max} - \min C_{max})/\min C_{max}$). The lower bounds and the percentage deviations from the lower bounds are also given in the table. Recall that the CPU time limit on all three algorithms is 1600 s.

From the tables, we can observe that the proposed genetic algorithm performs very well; it solves at least 61 out of 77 instances to the optimality and the average optimality gap is 1.67% over all instances.

As explained in the previous studies, the machine configurations have an important effect on the complexity of the problems, which affects solution quality. Néron et al. [32] identify some of the problems as hard problems. For these problems, the optimal solutions cannot be reached in a short time. The difficulty with these problems basically stems from their machine configurations (all these problems are *c* or *d* type problems). In other words, hard problems do not have a bottleneck stage with only a single machine. There are 24 problems in that group (the *c* and *d* types of 10×5 and 15×5 problems). The rest of the problems (all *a*, *b* types and 10×10 *c* type problems) are referred to as easy problems. In Tables 2 and 3, instance names typed in boldface represent the hard problems. As it can be observed from the tables, all three algorithms perform better for easy problems in terms of average optimality gap. For both hard and easy problems, we observe that the performances of the proposed genetic algorithm and the AIS algorithm are similar and better than the performance of the B&B algorithm in terms of average optimality gap. We also observe that the number of alternative solutions identified by the proposed algorithm is generally greater for easy problems than for hard problems. In addition, the percentage gap in makespan values is smaller for easy problems. In other words, for easy problems, the proposed genetic algorithm generates more solutions with better performance. To sum up, the proposed genetic algorithm performs similar to the AIS algorithm (and better than the B&B algorithm). As an additional benefit, it

Table 2
Comparison of alternative algorithms.

Instance	#	Cmax			LB	%Optimality Gap		
		Min	Max	%Gap		GA	AIS	B&B
j10c5a2	300	88	88	0.00	88	0.00	0.00	0.00
j10c5a3	224	117	117	0.00	117	0.00	0.00	0.00
j10c5a4	210	121	121	0.00	121	0.00	0.00	0.00
j10c5a5	183	122	122	0.00	122	0.00	0.00	0.00
j10c5a6	121	110	110	0.00	110	0.00	0.00	0.00
j10c5b1	392	130	130	0.00	130	0.00	0.00	0.00
j10c5b2	355	107	107	0.00	107	0.00	0.00	0.00
j10c5b3	282	109	109	0.00	109	0.00	0.00	0.00
j10c5b4	329	122	127	4.10	122	0.00	0.00	0.00
j10c5b5	360	153	153	0.00	153	0.00	0.00	0.00
j10c5b6	391	115	115	0.00	115	0.00	0.00	0.00
j10c5c1	73	68	71	4.41	68	0.00	0.00	0.00
j10c5c2	89	74	75	1.35	74	0.00	0.00	0.00
j10c5c3	107	72	73	1.39	71	1.41	1.40	0.00
j10c5c4	79	66	68	3.03	66	0.00	0.00	0.00
j10c5c5	101	78	79	1.28	78	0.00	0.00	0.00
j10c5c6	86	69	70	1.45	69	0.00	0.00	0.00
j10c5d1	106	66	67	1.52	66	0.00	0.00	0.00
j10c5d2	129	74	75	1.35	73	1.37	0.00	0.00
j10c5d3	90	64	65	1.56	64	0.00	0.00	0.00
j10c5d4	76	70	71	1.43	70	0.00	0.00	0.00
j10c5d5	108	66	68	3.03	66	0.00	0.00	0.00
j10c5d6	135	62	63	1.61	62	0.00	0.00	0.00
j10c10a1	126	139	139	0.00	139	0.00	0.00	0.00
j10c10a2	101	158	160	1.27	158	0.00	0.00	0.00
j10c10a3	155	148	148	0.00	148	0.00	0.00	0.00
j10c10a4	162	149	149	0.00	149	0.00	0.00	0.00
j10c10a5	122	148	148	0.00	148	0.00	0.00	0.00
j10c10a6	110	146	147	0.68	146	0.00	0.00	0.00
j10c10b1	300	163	163	0.00	163	0.00	0.00	0.00
j10c10b2	295	157	158	0.64	157	0.00	0.00	0.00
j10c10b3	290	169	169	0.00	169	0.00	0.00	0.00
j10c10b4	273	159	159	0.00	159	0.00	0.00	0.00
j10c10b5	296	165	167	1.21	165	0.00	0.00	0.00
j10c10b6	295	165	166	0.61	165	0.00	0.00	0.00
j10c10c1	86	115	116	0.87	113	1.77	1.80	12.40
j10c10c2	92	119	120	0.84	116	2.59	2.60	0.00
j10c10c3	91	116	118	1.72	98	18.37	18.40	35.70
j10c10c4	94	120	121	0.83	103	16.50	16.50	31.10
j10c10c5	98	126	127	0.79	121	4.13	4.10	19.80
j10c10c6	122	106	108	1.89	97	9.28	9.30	15.50

Table 3
Comparison of alternative algorithms continued.

Instance	#	Cmax			LB	%Optimality Gap		
		Min	Max	%Gap		GA	AIS	B&B
j15c5a1	267	178	178	0.00	178	0.00	0.00	0.00
j15c5a2	421	165	165	0.00	165	0.00	0.00	0.00
j15c5a3	347	130	130	0.00	130	0.00	0.00	0.00
j15c5a4	314	156	156	0.00	156	0.00	0.00	0.00
j15c5a5	433	164	164	0.00	164	0.00	0.00	0.00
j15c5a6	403	178	178	0.00	178	0.00	0.00	0.00
j15c5b1	429	170	170	0.00	170	0.00	0.00	0.00
j15c5b2	433	152	152	0.00	152	0.00	0.00	0.00
j15c5b3	434	157	157	0.00	157	0.00	0.00	0.00
j15c5b4	449	147	147	0.00	147	0.00	0.00	0.00
j15c5b5	414	166	166	0.00	166	0.00	0.00	0.00
j15c5b6	389	175	175	0.00	175	0.00	0.00	0.00
j15c5c1	97	85	86	1.18	85	0.00	0.00	0.00
j15c5c2	110	91	91	0.00	90	1.11	1.10	0.00
j15c5c3	151	87	88	1.15	87	0.00	0.00	0.00
j15c5c4	108	90	91	1.11	89	1.12	0.00	1.10
j15c5c5	107	74	76	2.70	73	1.37	1.40	15.10
j15c5c6	124	91	91	0.00	91	0.00	0.00	0.00
j15c5d1	451	167	167	0.00	167	0.00	0.00	0.00
j15c5d2	97	84	85	1.19	82	2.44	2.40	3.70
j15c5d3	82	83	84	1.20	77	7.79	7.80	24.70
j15c5d4	103	84	85	1.19	61	37.70	37.70	65.60
j15c5d5	132	80	81	1.25	67	19.40	19.40	44.80
j15c5d6	110	81	82	1.23	79	2.53	3.80	10.10
j15c10a1	432	236	236	0.00	236	0.00	0.00	0.00
j15c10a2	361	200	200	0.00	200	0.00	0.00	0.00
j15c10a3	400	198	198	0.00	198	0.00	0.00	0.00
j15c10a4	425	225	225	0.00	225	0.00	0.00	0.00
j15c10a5	380	182	182	0.00	182	0.00	0.00	0.50
j15c10a6	377	200	200	0.00	200	0.00	0.00	0.00
j15c10b1	441	222	222	0.00	222	0.00	0.00	0.00
j15c10b2	448	187	187	0.00	187	0.00	0.00	0.00
j15c10b3	460	222	222	0.00	222	0.00	0.00	0.00
j15c10b4	459	221	221	0.00	221	0.00	0.00	0.00
j15c10b5	420	200	200	0.00	200	0.00	0.00	0.00
j15c10b6	423	219	219	0.00	219	0.00	0.00	0.00
Averages								
Hard	118.79			1.48		3.18	3.13	6.88
Easy	302.15			0.29		0.99	0.99	2.17
All	245.00			0.66		1.67	1.66	3.64

Table 4
Benefit of having alternative schedules.

Instance	R1			R2			S1			S2		
	Min	Max	%Gap	Min	Max	%Gap	Min	Max	%Gap	Min	Max	%Gap
j10c5a2	97.9	113.27	15.70	9.61	24.97	159.83	84.31	172.29	104.35	828.82	1923.25	132.05
j10c5a3	124.06	152.08	22.59	6.8	34.82	412.06	65.55	140.67	114.60	610.98	1875.07	206.90
j10c5a4	126.33	147.02	16.38	5.2	25.89	397.88	64.85	161.4	148.88	611.43	1884.12	208.15
j10c5a5	127.53	145.02	13.71	5.34	22.83	327.53	68.75	169.08	145.93	681.69	1874.23	174.94
j10c5a6	115.68	127.49	10.21	5.48	17.3	215.69	62.62	130.93	109.09	608.5	1529.34	151.33
j10c5b1	134.56	159.32	18.40	4.52	29.28	547.79	75.43	214.43	184.28	729.99	2355.67	222.70
j10c5b2	113.05	142.65	26.18	5.9	35.51	501.86	80.94	208.35	157.41	801.55	2446.94	205.28
j10c5b3	117.11	136.75	16.77	7.79	27.43	252.12	76.96	191.62	148.99	876.82	2457.56	180.28
j10c5b4	129.73	153.05	17.98	7.55	30.87	308.87	71.41	190.76	167.13	735.14	2418.91	229.04
j10c5b5	154.78	170.11	9.90	1.78	17.11	861.24	54.45	149.96	175.41	545.06	1881.97	245.28
j10c5b6	122.33	147.34	20.44	7.16	32.18	349.44	80.81	215.46	166.63	772.21	2824.66	265.79
j10c5c1	85.7	91.38	6.63	12.36	18.04	45.95	101.44	135.3	33.38	1005.27	1554.76	54.66
j10c5c2	89.98	101.47	12.77	11.39	22.88	100.88	95.23	146.34	53.67	1032.63	1709.16	65.52
j10c5c3	89.73	99.34	10.71	10.85	20.47	88.66	91.37	154.34	68.92	901.13	1572.23	74.47
j10c5c4	89.82	95.2	5.99	13.21	18.59	40.73	116.13	159.48	37.33	1168.92	1884.29	61.20
j10c5c5	93.08	106.87	14.82	11.3	25.09	122.04	79.18	147.84	86.71	907.66	1664.75	83.41
j10c5c6	92.55	97.15	4.97	14.29	18.89	32.19	116.31	160.55	38.04	1194.64	1771.27	48.27
j10c5d1	89.05	96.84	8.75	13.42	21.21	58.05	120.5	165	36.93	1142.67	1816.77	58.99
j10c5d2	96.76	105.21	8.73	14.98	23.43	56.41	123.5	190.39	54.16	1347.45	2168.18	60.91
j10c5d3	85.92	93.66	9.01	13.86	21.6	55.84	127.75	176.02	37.78	1225.91	1874.83	52.93
j10c5d4	92.64	100.92	8.94	12.47	20.75	66.40	127.9	186	45.43	1269.86	1987.01	56.47
j10c5d5	90.73	95.23	4.96	13.99	18.48	32.09	126.12	162.95	29.20	1209.51	1771.52	46.47
j10c5d6	85.73	89.07	3.90	14.61	17.95	22.86	116.76	155.2	32.92	1135.69	1593.24	40.29
j10c10a1	155.03	171.8	10.82	14.6	31.37	114.86	131.54	237.49	80.55	1443.62	3046.08	111.00
j10c10a2	171.32	186.08	8.62	11.19	25.95	131.90	127.43	214.32	68.19	1517.63	3124.87	105.90
j10c10a3	155.01	165.65	6.86	6.77	17.41	157.16	121.79	199	63.40	1363.69	3037.24	122.72
j10c10a4	163.61	183.27	12.02	13.42	33.08	146.50	140.63	266.66	89.62	1473.45	3494.26	137.15
j10c10a5	157.91	175.99	11.45	9.43	27.51	191.73	154.92	276.56	78.52	1880.13	3667.99	95.09
j10c10a6	155.46	171.23	10.14	8.03	23.8	196.39	145.42	226.73	55.91	1602.54	3053.58	90.55
j10c10b1	168.02	195.75	16.50	4.92	32.65	563.62	99.33	288.36	190.31	1116.34	4241.64	279.96
j10c10b2	166.87	205.77	23.31	9.65	48.55	403.11	114.9	379.48	230.27	1357.88	4815.06	254.60
j10c10b3	175.11	204.56	16.82	5.95	35.41	495.13	111.66	332.94	198.17	1290.29	4261.64	230.29
j10c10b4	169.01	200.98	18.92	9.68	41.65	330.27	119.65	327.29	173.54	1598.35	4362.78	172.96
j10c10b5	176.02	210.92	19.83	10.77	45.67	324.05	139.4	325.38	133.41	1508.66	4324.76	186.66
j10c10b6	173.3	209.45	20.86	8.09	44.25	446.97	88.49	368.76	316.73	920.83	4267.74	363.47
j10c10c1	150.23	156.87	4.42	19	25.64	34.95	234.17	301.97	28.95	2224.43	3148.23	41.53
j10c10c2	147.93	157.96	6.78	19.98	30.02	50.25	212.55	282.54	32.93	2605.06	3527.6	35.41
j10c10c3	146.61	162.62	10.92	16.11	32.11	99.32	206.6	313.43	51.71	2509.38	3704.54	47.63
j10c10c4	153.6	165.07	7.47	19.89	31.36	57.67	245.15	319.78	30.44	2240.73	3649.25	62.86
j10c10c5	157.8	169.76	7.58	18.85	30.81	63.45	227.13	313.24	37.91	2618.2	3955.08	51.06
j10c10c6	138.52	154.52	11.55	18.25	34.25	87.67	222.04	318.35	43.38	2154.74	3550.22	64.76

provides 245 solutions on the average with high performance (0.66% gap between the worst and the best on the average) instead of just a single schedule.

Next, we demonstrate the benefit of using the proposed two-step framework. Here, the side concern is the ability of the generated schedules to absorb the negative effects of unforeseen future disruptions. In order for the proposed two-step approach to work properly, it is necessary to be able to obtain significantly different schedules at the end of the first stage. For the sake of simplicity and ease of illustration, we test the performance of the generated schedules in $\mathbb{P}$ in terms of the robustness and stability measures explained before to see if the difference within the population is enough to harvest the benefits of having more than one alternative in handling side concerns.

The values of the robustness and stability measures in the second stage are estimated by simulating the schedules for 100 replications. In simulation runs, a breakdown occurs after a machine is kept busy for a time period whose length is drawn from a Gamma distribution with a shape parameter of 0.7. In each replication, for each S in the final population $\mathbb{P}$ of the genetic algorithm, a corresponding S_r is constructed by inserting repair durations that are drawn from a Gamma distribution with a shape parameter of 1.4. The means of downtime and uptime distributions are taken as 5 and 20, respectively. In other words, we have $D_{im} \sim \Gamma(1.4, \frac{5}{1.4})$ and $U_{im} \sim \Gamma(0.7, \frac{20}{0.7}), \forall(i,m)$. The same downtime and uptime distributions are used for all the machines. The choice of shape parameters is in compliance with the recommendation in [27] in the absence of real data. In the calculation of $R2$, in each replication, the regret of a schedule S is estimated as the opportunity loss in its makespan as compared to the makespan of the best schedule for that replication in $\mathbb{P}$. In other words, (3) is replaced with

$$S^* \in \arg\min_{S \in \mathbb{P}} \left\{ \max_{j \in J} C_j(S_r) \right\}. \quad (6)$$

The expectations in (1), (2), and (4) are estimated by the sample means of 100 observations. Similarly, the variances in (5) are estimated by the sample variances of 100 observations. Tables 4 and 5 summarize the results. In the tables, for all measures (i.e., $R1$, $R2$, $S1$, and $S2$), the minimum value and the maximum value within the population are reported. To asses the difference within the population, we also report the percentage deviation between the minimum and the maximum.

It can be observed from the tables that, even though the performances of the schedules in $\mathbb{P}$ are close to one another in terms of the makespan, their performances in terms of robustness and stability are very different. The variation is drastic especially for the robustness measure based on regret (i.e., $R2$) and the

Table 5
Benefit of having alternative schedules continued.

Instance	R1			R2			S1			S2		
	Min	Max	%Gap	Min	Max	%Gap	Min	Max	%Gap	Min	Max	%Gap
j15c5a1	181.22	200.72	10.76	3.2	22.71	609.69	84.26	190.72	126.35	831	2464.01	196.51
j15c5a2	167.36	198.12	18.38	2.36	33.12	1303.39	84.98	262.94	209.41	753.9	3177.48	321.47
j15c5a3	133.24	167.78	25.92	3.24	37.78	1066.05	81.92	218.74	167.02	800.22	2724.08	240.42
j15c5a4	157.58	169.06	7.29	1.58	13.06	726.58	70.3	159.05	126.24	725.85	2058.88	183.65
j15c5a5	165.96	209.52	26.25	1.96	45.52	2222.45	83.34	183.96	120.73	878.17	2711.78	208.80
j15c5a6	180.07	228.45	26.87	2.07	50.45	2337.20	80.12	211.73	164.27	853.12	2900.14	239.95
j15c5b1	171.33	188.28	9.89	1.33	18.28	1274.44	80.22	302.68	277.31	863.89	3419.48	295.82
j15c5b2	153.98	167.69	8.90	1.98	15.69	692.42	71.9	193.99	169.81	664.33	2668.29	301.65
j15c5b3	160.3	180.82	12.80	3.3	23.82	621.82	89.61	250.04	179.03	921.88	3382.62	266.93
j15c5b4	153.38	181.38	18.26	6.33	34.34	442.50	104.85	262.15	150.02	1159.05	3695.72	218.86
j15c5b5	169.98	186.57	9.76	3.98	20.57	416.83	72.24	224.74	211.10	648.24	3194.16	392.74
j15c5b6	177.6	192.56	8.42	2.6	17.56	575.38	74.05	202.89	173.99	688.22	2990.13	334.47
j15c5c1	109.32	119	8.85	15.67	25.35	61.77	176.49	278.42	57.75	2024.61	3640.87	79.83
j15c5c2	119.14	124.71	4.68	16.31	21.88	34.15	210.79	260.18	23.43	2027.79	3138.47	54.77
j15c5c3	110.46	114.6	3.75	16.42	20.57	25.27	181.25	226.68	25.06	1805.35	2786.84	54.37
j15c5c4	111.43	122.06	9.54	16.69	27.32	63.69	164.39	238.16	44.87	1816.1	2901.73	59.78
j15c5c5	98.77	106.03	7.35	17.16	24.42	42.31	195.72	248.14	26.78	2203.48	3336.51	51.42
j15c5c6	120.75	128.9	6.75	19.72	27.87	41.33	253.86	319.67	25.92	2524.38	4113.26	62.94
j15c5d1	209.89	213.07	1.52	5.87	9.06	54.34	381.86	453.12	18.66	3554.46	5585.48	57.14
j15c5d2	114.68	123.7	7.87	17.13	26.15	52.66	223.14	327.19	46.63	2325.01	4047.35	74.08
j15c5d3	114.38	120.47	5.32	18.31	24.41	33.32	221.61	311.56	40.59	2287.69	3869.22	69.13
j15c5d4	112.91	122.75	8.71	18.38	28.22	53.54	214.39	305.74	42.61	2359.48	3568.08	51.22
j15c5d5	110.21	120.99	9.78	13.05	23.83	82.61	238.75	349.67	46.46	2305.15	4643.49	101.44
j15c5d6	104.06	114.42	9.96	15.77	26.14	65.76	168.2	272.01	61.72	1978.88	3253.3	64.40
j15c10a1	238.68	255.05	6.86	2.68	19.05	610.82	113.89	336.5	195.46	1230.88	5039.61	309.43
j15c10a2	207.64	230.49	11.00	7.56	30.41	302.25	138.86	363.11	161.49	1473.4	5410.85	267.24
j15c10a3	205.28	231.78	12.91	7.28	33.78	364.01	129.22	436.56	237.84	1571.82	5647.36	259.29
j15c10a4	228.27	243.44	6.65	3.27	18.44	463.91	110.91	365.73	229.75	1239.19	4999.36	303.44
j15c10a5	188.29	223.26	18.57	6.22	41.19	562.22	140.7	495.41	252.10	1460.04	6544.27	348.23
j15c10a6	207.58	237.24	14.29	7.5	37.16	395.47	151.89	457.24	201.03	1890.23	6684.59	253.64
j15c10b1	275.22	287.38	4.42	13.31	25.48	91.44	485.22	670.21	38.12	4923.36	8626.05	75.21
j15c10b2	237.04	242.82	2.44	16.74	22.52	34.53	548.78	637.97	16.25	4863.13	8030.03	65.12
j15c10b3	278.59	292.75	5.08	13.86	28.03	102.24	557.48	779.91	39.90	4975.21	10 263.56	106.29
j15c10b4	277.12	286.93	3.54	16.22	26.03	60.48	556.07	708.31	27.38	4329.97	10 353	139.10
j15c10b5	251	258.11	2.83	16.33	23.45	43.60	497.25	616.77	24.04	4271.63	8103.84	89.71
j15c10b6	276.09	284.23	2.95	19.15	27.3	42.56	567.27	702.63	23.86	4347.23	7834.71	80.22
Averages												
Hard			7.68			55.54			42.29			61.84
Easy			12.98			445.12			133.00			191.95
All			11.33			323.69			104.73			151.40

stability measures. We also observe that the average variation for easy instances are larger than that for hard instances. This is intuitive because the proposed genetic algorithm generates more schedules for easy problems than for hard problems (see Tables 2 and 3).

These results allow for a useful choice for the final schedule. In fact, we do not dub any single schedule as the unique correct schedule. For example, a decision-maker that is mainly interested in robustness may find a different schedule more appealing than the choice of a decision-maker who is concerned with stability. The important point is that the proposed method allows for this choice by providing different possible solutions. Specifically, Tables 4 and 5 show that, by sacrificing 0.66% from the makespan, the decision-maker can gain up to 11% in $R1$, 324% in $R2$, 105% in $S1$ and 151% in $S2$ on the average. These results indicate that the schedules we obtain at the end of Step 1 are very different from each other (at least for $R2$, $S1$, and $S2$), and provide enough evidence that generated schedules can be very different in terms of any other concern the decision-maker could have. Hence, the proposed method has the potential to be useful when the decision-maker's concern is a very complicated one which cannot even be mathematically formulated, as long as the decision-maker is capable of consistently comparing provided alternatives with one another in terms of their compatibility with his/her preferences. Note that the concerns of the decision-maker need not be unique or even exist in the beginning. Inherent characteristics of a solution alternative can also be a side concern: e.g., assume that at the time the decision is made (i.e., at the end of Step 1) a transportation activity has just completed earlier than expected and it is now preferable for a particular operation to precede others. The proposed framework allows incorporation of such concerns by providing several different alternatives so that a suitable schedule can be chosen.

To sum up, our computational experiments indicate that the proposed genetic algorithm is capable of generating a high number of high quality schedules (in terms of the makespan) whose performance vary significantly in terms of their ability to absorb the negative effects of unforeseen machine breakdowns. Hence, it can be effectively used by the decision-makers to facilitate handling their side concerns.

6. Concluding remarks and future research directions

In certain scheduling environments, the decision-maker has important side concerns in addition to minimizing the principal performance measure. Examples of these concerns can be schedule's conformance to omitted constraints in the modelling phase, economies of the schedule's implementation, the ability of the shop floor to position the work, tooling and operators in a way

that will smooth the execution of the schedule, the capacity allocation between competing production lines, the ability of the schedule to absorb the negative effects of the unforeseen future disruptions, etc. Such concerns of the decision-maker may not be all known in advance, can be implicit or qualitative, and therefore may not be included in the initial problem definition as an objective function and/or as constraints. In these cases, the frequently used techniques of multi-objective optimization become inapplicable. To address this problem, in this paper, we have proposed a new two-step framework. Specifically, we decompose the problem into two sub-problems and handle each sequentially. The decomposition is achieved by deferring the detailed consideration of the side concerns. In the first sub-problem, the aim is to obtain several alternative schedules with good performance values, which are different from each other significantly. The objective of the second sub-problem is to choose a schedule that complies with the side concerns of the decision-maker among the alternatives obtained in the previous step. We propose a multimodal genetic algorithm to solve the first sub-problem. Our computational experiments on a set of benchmark problems from the literature indicate that the proposed algorithm not only performs well when compared with the existing state-of-the-art methods in terms of minimizing the makespan, but that it is also quite effective in obtaining a diverse set of good (mostly optimal) alternative solutions. Note that the proposed two-step framework is capable of handling other scheduling problems as well without extensive modifications. The only assumption we make is that the underlying scheduling problem can be solved with an evolutionary algorithm to be able to use niching approaches to handle diversity (which is convenient in evolutionary algorithms but other population-based metaheuristics such as particle swarm optimization can also be considered).

In summary, having a diverse set of schedules that perform well in terms of the principal measure provides the decision-makers with flexibility. There are a number of ways to draw the advantage of this flexibility, one of which, as illustrated in this paper, is to exploit it to discover robust or stable schedules by spending a reasonable computational effort. We identify several future research directions.

First, the proposed two-step framework, using some preference aggregation methods in social choice theory in the second step, can be utilized to identify a *collective* schedule in the environments where several decision-makers with different perspectives collaborate.

Second, employing a more interactive approach in the solution of the second sub-problem to include decision-maker's expertise and preferences more accurately as in [13] seems to be of practical importance.

In addition, this study can be extended to more complex problems, including sequence-dependent setups on machines, machine eligibility, time lags on operations, precedence constraints among jobs, etc. Along the same lines, the proposed two-step approach can be applied to a wider range of scheduling environments, including job shops and open shops.

Finally, a graphical analysis of the robustness and stability of the schedules in the second step as in [15] seems to be of practical value.

References

[1] Ando S, Sakuma J, Kobayashi S. Adaptive isolation model using data clustering for multimodal function optimization. In: Proceedings of genetic and evolutionary computation conference (GECCO-05); 2005. p. 1417–24.

[2] Ando S, Sakuma J, Kobayashi S. Sample-based crowding method for multimodal optimization in continuous domain. In: The 2005 IEEE congress on evolutionary computation; 2005. p. 1867–74.

[3] Artigues C, Billaut J-C, Esswein C. Maximization of solution flexibility for robust shop scheduling. European Journal of Operational Research 2005;165:314–28.

[4] Aytug H, Lawley MA, McKay K, Mohan S, Uzsoy R. Executing production schedules in the face of uncertainties: a review and some future directions. European Journal of Operational Research 2005;161:86–119.

[5] Baccouche A, Huyet A-L, Pierreval H, Feyech B. Multimodal optimization of simulated systems. In: Proceedings of the 7th EUROSIM congress on modelling and simulation; 2010. p. 52.

[6] Besbes W, Loukil T, Teghem J. Using genetic algorithm in the multiprocessor flow shop to minimize the makespan. In: 2006 International conference on service systems and service management, vol. 2; 2006. p. 1228–33.

[7] Carlier J, Néron E. An exact method for solving the multiprocessor flowshop. R.A.I.R.O Operations Research 2000;34:1–25.

[8] Chaari T. Un algorithme génétique pour l'ordonnancement robuste: application au problème du flow shop hybride. PhD thesis. University of Valenciennes and Hainaut-Cambresis & Faculty of Economics and Management of Sfax; 2010.

[9] Darwen P, Yao X. Every niching method has its niche: fitness sharing and implicit sharing compared. In: Proceedings parallel problem solving from nature – PPSN IV. New York: Springer; 1996. p. 398–407.

[10] DeJong KA. An analysis of the behavior of a class of genetic adaptive systems. PhD thesis. Ann Arbor: University of Michigan; 1975.

[11] Engin O, Döyen A. A new approach to solve hybrid flow shop scheduling problems by artificial immune system. Future Generation Computer Systems 2004;20:1083–95.

[12] Fleury G, Gourgand M. Genetic algorithms applied to workshop problems. International Journal of Computer Integrated Manufacturing 1998;11(2):183–92.

[13] Garcia-Hernandez L, Pierreval H, Salas-Morera L, Arauzo-Azofra A. An interactive genetic algorithm with c-means clustering for the unequal area facility layout problem. In: Proceedings of the 10th international conference on intelligent systems design and applications; 2010.

[14] Garey MR, Johnson DS. Computers and intractability: a guide to the theory of NP-completeness. San Francisco: Freeman; 1979.

[15] Ghezail F, Pierreval H, Hajri-Gabouj S. Analysis of robustness in proactive scheduling: a graphical approach. Computers and Industrial Engineering 2010;58(2):193–8.

[16] Goldberg DE, Deb K, Horn J. Massive multimodality, deception and genetic algorithms. Technical Report IlliGAL (Illinois Genetic Algorithms Laboratory) Report No 92005, University of Illinois, Urbana; 1992.

[17] Goldberg DE, Richardson J. Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the second international conference on genetic algorithms. Erlbaum, Hillsday, NJ; 1987. pp. 41–9.

[18] Graham RL. Bounds for certain multiprocessing anomalies. The Bell System Technical Journal 1966;45:1563–81.

[19] Harik GR. Finding multimodal solutions using restricted tournament selection. In: Proceedings of the sixth international conference on genetic algorithms. San Mateo, CA: Morgan Kaufmann; 1995. pp. 24–31.

[20] Herroelen W, Leus R. Project scheduling under uncertainty: survey and research potentials. European Journal of Operational Research 2005;165(2):289–306.

[21] Hocaoglu C, Sanderson AC. Multimodal function optimization using minimal representation size clustering and its application to planning multipaths. Evolutionary Computation 1997;5(1):81–104.

[22] Hoogeveen H. Multicriteria scheduling. European Journal of Operational Research 2005;167(3):592–623.

[23] Hoogeveen JA, Lenstra JK, Veltman B. Preemptive scheduling in a two-stage multiprocessor flow shop is NP-Hard. European Journal of Operational Research 1997;89:172–5.

[24] Johnson SM. Optimal two- and three-stage production schedules with set up times included. Naval Research Logistics Quarterly 1954:61–8.

[25] Kempf K, Uzsoy R, Smith S, Gary K. Evaluation and comparison of production schedules. Computers in Industry 2000;42:203–20.

[26] Kouvelis P, Yu G. Robust discrete optimization and its applications. Dordrecht, The Netherlands: Kluwer Academic Publishers; 1997.

[27] Law AM, Kelton WD. Simulation modeling and analysis. 3rd ed. Singapore: McGraw-Hill; 2000.

[28] Leon VJ, Wu SD, Storer RH. Robustness measures and robust scheduling for job shops. IIE Transactions 1994;26(5):32–43.

[29] Linn R, Zhang W. Hybrid flow shop scheduling: a survey. Computers and Industrial Engineering 1999;37:57–61.

[30] Mahfoud SW. Niching methods for genetic algorithms. PhD thesis, University of Illinois at Urbana-Champaign, Urbana, IL; 1995.

[31] Mengshoel OJ, Goldberg DE. Probabilistic crowding: deterministic crowding with probabilistic replacement. In: Proceedings of the genetic and evolutionary computation conference (GECCO-99); 1999. p. 409–16.

[32] Néron E, Baptiste P, Gupta JND. Solving hybrid flow shop problem using energetic reasoning and global operations. Omega, The International Journal of Management Science 2001;29:501–11.

[33] Oguz C, Ercan MF. A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. Journal of Scheduling 2005;8:323–51.

[34] Pétrowski A. A clearing procedure as a niching method for genetic algorithms. In: Proceedings of the 1996 IEEE international conference on evolutionary computation; 1996. p. 798–803.

[35] Ribas I, Leisten R, Framiñan JM. Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. Computers and Operations Research 2010;37:1439–54.
[36] Ruiz R, Maroto C. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. European Journal of Operational Research 2006;169:781–800.
[37] Ruiz R, Vázquez-Rodríguez JA. The hybrid flow shop scheduling problem. European Journal of Operational Research 2010;205(1):1–18.
[38] Sabuncuoglu I, Goren S. Hedging production schedules against uncertainty in manufacturing environment with a review of robustness and stability research. International Journal of Computer Integrated Manufacturing 2009; 22(2):138–57.
[39] Salvador MS. A solution to a special class of flow shop scheduling problems. In: Elmaghraby SE, editor. Symposium on the theory of scheduling and its applications. Berlin: Springer; 1973. p. 83–91.
[40] Schiavinotto T, Stützle T. A review of metrics on permutations for search landscape analysis. Computers and Operations Research 2007;34:3143–53.
[41] Singh G, Deb K. Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In: Proceedings of the 8th annual conference on genetic and evolutionary computation. Seattle, Washington: 2006. p. 1305–12.
[42] T'Kindt V, Billaut J-C. Multicriteria scheduling: theory, models and algorithms. Springer; 2006.
[43] Vieira GE, Herrmann JW, Lin E. Rescheduling manufacturing systems: a framework of strategies, policies, and methods. Journal of Scheduling 2003;6(1): 39–62.
[44] Vignier A, Billaut JC, Proust C. Hybrid flowshop scheduling problems: state of the art. R.A.I.R.O Operations Research 1999;33:117–83.
[45] Wu SD, Byeon E-S, Storer RH. A graph-theoretic decomposition of the job shop scheduling problem to achieve scheduling robustness. Operations Research 1999;47(1):113–24.
[46] Wu SD, Storer RH, Chang P-C. One-machine rescheduling heuristics with efficiency and stability as criteria. Computers and Operations Research 1993;20(1):1–14.
[47] Yin X. A fast genetic algorithm with sharing scheme using cluster analysis methods in multimodal function optimization. In: Forrest S, editor. Proceedings of the fifth international conference on genetic algorithms. San Mateo, CA: Morgan Kaufman; 1993.
[48] Yu EL, Suganthan PN. Ensemble of niching algorithms. Information Sciences 2010;180(15):2815–33.