The International Journal of Advanced Manufacturing Technology (2018) 97:2781–2808
https://doi.org/10.1007/s00170-018-2146-z

ORIGINAL ARTICLE

Multiple-order permutation flow shop scheduling under process interruptions

Humyun Fuad Rahman[1] · Ruhul Sarker[2] · Daryl Essam[2]

Received: 26 December 2016 / Accepted: 7 May 2018 / Published online: 21 May 2018

Abstract
The permutation flow shop problem is a complex combinatorial optimization problem. Over the last few decades, a good number of algorithms have been proposed to solve static permutation flow shop problems. However, in practice, permutation flow shop problems are not static but rather are dynamic because the orders (where each order contains multiple jobs) arrive randomly for processing and the operation of any job may be interrupted due to resource problems. For any interruption, it is necessary to reschedule the existing jobs that are under process at different stages in the production system and also any orders that were previously accepted that are waiting for processing. In this paper, a memetic algorithm-based rescheduling approach has been proposed to deal with both single and multiple orders while considering random interruptions of resources. The experimental results have shown that the performance of the proposed approach is superior to traditional reactive approaches.

Keywords Manufacturing · Disruptions · Order acceptance and scheduling · Permutation flow shop scheduling · Genetic algorithm · Memetic algorithm

1 Introduction

The permutation flow shop scheduling problem (PFSP) is common in many manufacturing industries, such as steel and iron production, pharmaceutical, food processing and automobile industries. In a conventional PFSP, n jobs are processed on m machines, where each job has to follow the same processing order in all machines. In solving PFSPs, minimization of makespan is a popular objective. Makespan is the time difference between the start of the first operation in the first machine and the completion of the last operation in the last machine.

Over the past few decades, in most papers about PFSPs, it is commonly assumed that (i) a PFSP is static with a goal of scheduling a single order (with a set of jobs) once, on a set of machines, and (ii) the production environment is an ideal one with no interruption of product or process. However, in a real-world make-to-order production system, the shop floor deals with multiple orders, where the arrival pattern of orders is usually random, and the processing of jobs within these orders may be interrupted due to machine breakdown, machine maintenance and shortage of process-related resources. The multiple orders case, under ideal conditions, involves two decision problems that must be solved sequentially. Firstly, it must be decided if an arriving order should be accepted, while considering the available shop capacity and the specified due date, and then it is necessary to schedule the newly accepted orders with the existing orders scheduled previously (that are currently either under process or waiting in queue for processing). Note that to accommodate a newly accepted order effectively, some or all of the existing orders waiting in the queue may be rescheduled. In addition to the above two decisions, in order to minimize the effect of any interruption, it is necessary to revise the schedule [1]. Here, a real-time rescheduling approach is necessary to quickly recover from any interruption.

The static scheduling problems have limited practical applications because new orders arrive at random time intervals [2] and the possible interruptions in the production system [3]. In some systems, production capacity and resource availability may vary with time [4]. The static PFSPs disregard these variations. Although there are a few studies available for static

✉ Humyun Fuad Rahman
rahman@m-tech.aau.dk

[1] Department of Mechanical and Manufacturing Engineering, Aalborg University, Aalborg, Denmark

[2] School of Engineering and Information Technology, UNSW, Canberra, Australia

single-order PFSPs under disruption, studies with different disruption scenarios are absent from the literature. Besides, no attempt has been made to investigate multiple-order PFSPs under disruption. So a systematic approach is required to study different process interruptions in multiple-order PFSPs.

In this research, we have considered multiple-order PFSPs, where a production process may be interrupted at any time due to different disruption events. The disruption information is not known a priori. In multiple-order PFSPs, each order can be assumed to be a single-order PFSP, but there is no prior information about its arrival time, due dates and order composition. This problem involves three different decision strategies: (i) should we accept a new order? (ii) how can an accepted order be scheduled? and (iii) if there is a process interruption, how can the orders (under process and accepted) be rescheduled? To deal with the above research questions, we first developed a memetic algorithm (MA) that can determine the makespan of each order. This basically schedules the jobs of a single-order static PFSP. This depends on maximizing profit (or revenue or machine utilization) while not exceeding the existing resource capacity and also delivering the order according to a customer defined due date. We propose a simple heuristic to make this decision. To make the second decision, we have applied our proposed MA to schedule multiple orders (of new orders with all other orders waiting for processing) and their jobs. The objective in this problem is to minimize the completion time of all orders accepted for processing. The third decision is to reschedule the affected orders if the production system is disrupted. If an order under process is disrupted, some or all jobs of those orders and the accepted orders waiting in the queue are affected. These affected orders may be delayed in processing, and that may lead to exceeding the due date (tardiness) for all or a sub-set of those orders. In repairing the disrupted schedule, we reschedule the affected jobs and orders in order to maximize total profit while satisfying the current capacity and due date constraints.

As the above decisions can be taken on a real-time basis, we refer to these combined approaches as real-time (RT) strategies in this paper.

There is no standard benchmark available for multiple-order PFSPs with process interruptions. For experimental study with the proposed approach, eight 10-machine PFSP instances from the Taillard's benchmark [5] have been chosen, and then arrival times, due dates and disruption information were randomly generated for them.

This paper is organized as follows. Section 2 provides the background study of the research topics related to this research. In the next section, PFSPs with process interruptions are formulated. Section 4 describes the MA. In Section 5, an experimental study and the effectiveness of the proposed algorithm are presented. Section 6 presents conclusions.

2 Related work

This section provides an overview of PFSPs, with a special emphasis on order acceptance decisions and production disruptions. First, we present a brief review on static single- and multiple-order problems. Next, a literature on dynamic multiple-order problems in single- and multiple-machine environments has been discussed. Finally, an overview on studies relating to single-order problems with disruption is presented.

2.1 Static single- and multiple-order problems

The static single-order PFSPs have been widely studied in the literature. In 1954, Johnson proposed an algorithm for optimally solving two machines, as well as some special three-machine problems. That has been recognized as the most influential research in machine scheduling today [6]. It is well known that in general, more than two machine static PFSPs is *NP hard* (non-deterministic polynomial-time hard) [7]. The traditional optimization approaches, like branch and bound algorithm [8] and integer programming approach [9], can only solve small-sized PFSPs effectively. To solve bigger problems, researchers have focused on heuristics [10] and meta-heuristics [11–19]. In the literature, few studies consider multiple orders in both single- and multiple-machine production environments. Most of these studies focus on static multiple-order problems, where at the beginning of production, a production floor receives a pool of orders from which a certain number of orders are accepted, based on available capacity and due dates [20–24].

2.2 Dynamic multiple-order problems

The next level of complexity is dynamic order arrival in both single- and multi-machine environments. Only a few existing research works have dealt with dynamic multiple-order problems. In a single-machine environment, Wester et al. [25] investigated the relationship between three different order selection approaches: order selection based on information of the current production schedule with reschedule if needed, order selection based on estimation of workload and order selection based on the effect of the tardiness of previously accepted orders. Duenyas and Hopp [26] extended the problem to job shop (multi-machine) environments and presented an optimal control limit model to maximize expected profit with customers quoted due dates. Duenyas [27] extended that research for multiple customer classes. Nandi and Rogers [28] proposed to accept or reject an arriving order based on its potential profit. Later, they proposed another approach based on the total resources required by the order and its load on the busiest machine [29]. Moreira and Alves [30] developed an approach based on workload, total acceptance and due dates settings.

From these studies, it can be seen that no studies consider production disruptions.

2.3 Single-order problems with disruptions

Recently, a few studies considered disruptions in single-order production environments. Such interruptions can be classified into two categories [31–35]: (i) resource related: machine breakdowns, preventive maintenance, operator sickness, machine tool failure, raw material shortage, delay in material arrival and defective raw material and (ii) job related: job rush, change in due dates, early or late arrival of jobs, rework, scrap, change in job's priority or specifications and change in processing times. Although an interruption event can be predictive (or known in advance), such as machine preventive maintenance, most of them are stochastic in nature and are not known until they take place [36].

As regards to disruptions, all existing research has been carried out on either single- [37–43] or multiple-machine environments [43–58]. In single-machine scheduling, Adiri et al. [37] studied a single breakdown case with an objective of minimizing the number of tardy jobs, Liao and Chen [38] investigated a sequence-dependent set-up case with machine breakdowns and Hall and Potts [39] studied the new job arrival case, which scheduled them with the existing jobs and dealt with any disruption that happened. Yang [40] studied the similar rescheduling problem of new job arrival with compression time and used a cost function to minimize the effect of disruption. Wu et al. [41] studied the problem of rescheduling bi-criterion problems (minimize makespan and system impact) in single-machine environments when a system is suddenly interrupted with either a machine breakdown, a change in processing times of a job or an order rush (here each order contains a single job). To solve this, they proposed three bi-criterion heuristics. Unal et al. [42] extended Wu et al.'s [41] problem to consider sequence dependent setup. For sudden arrival of orders (each order contains a job), they proposed a polynomial time algorithm with makespan minimization as its objective. They also proposed two heuristics for minimizing total weighted completion time, which is strongly NP hard. Qi et al. [43] proposed a rescheduling approach for single and parallel two-machine environments that were subjected to random processing time variations and machine breakdowns. A few attempts have been made to solve scheduling problems with process interruptions such as new job arrival, machine unavailability and breakdowns, in single-order flow shop environments with makespan minimization as the objective. Allahverdi [44] studied static two-machine flow shops with sudden machine breakdowns. Rahman et al. [45] studied single-order PFSPs with machine unavailability and breakdowns with makespan minimization as the objective. Katragjini et al. [46] extended this study to study single-order PFSPs with three disruption events: machine breakdowns, new job arrival and job ready time variations. Ruiz et al. [47] proposed three preventive maintenance policies that were based on establishing reliability for static PFSPs. Perez-Gonzalez and Framinan [48] and Perez-Gonzalez et al. [49] studied process interruption that was created by the arrival of new jobs during production. Perez-Gonzalez and Framinan [48] studied the problem while assuming that the initial schedule of previous jobs was maintained when scheduling new jobs. Perez-Gonzalez et al. [49] studied a similar problem that allowed changes in the schedule of jobs already in the system when rescheduling with new jobs. They proposed a refreshing variable neighbourhood algorithm which outperformed other techniques. For static job shop environments, Fahmy et al. [50] proposed a reactive scheduling approach where the affected tasks were first replaced by dummy tasks, which were then rescheduled. But this approach is computationally expensive [51]. Hasan et al. [51] and Sarker et al. [52, 53] proposed a hybrid GA to solve static job shop scheduling problems under machine breakdown and preventive maintenance. Al-Hinai and ElMekkawy [54] proposed a two-stage hybrid GA for solving flexible job shop problems subject to machine breakdowns. Therefore, for both single- and multiple-machine scheduling problems with process interruptions, most of these studies considered makespan minimization as the objective and did not consider due dates for order acceptance decisions and profit maximization.

2.4 Summary

In summary, from the related works on order acceptance and process interruptions, it can be seen that the above-mentioned studies have restrictive assumptions. These assumptions are as follows: (1) the order acceptance/rejection decisions and the scheduling decisions were made separately and sequentially, (2) an uninterrupted production system was assumed and (3) production disruption is only limited to single-order production environments. These assumptions isolate the studies on make-to-order production systems from real-life manufacturing environments, and therefore, the present study aims at minimizing this gap.

3 Problem description and mathematical model

In this section, the multiple-order PFSPs are described and necessary assumptions are discussed.

3.1 Problem description

As discussed earlier, in multiple-order PFSPs with interruptions, schedulers have to deal with three important issues: (1)

which orders should be accepted or rejected, (2) how to schedule those orders and (3) whenever an interruption happens, how the accepted orders are rescheduled.

In case of disruption, an order is defined as an affected order, if one or more jobs of that accepted order is delayed after a disruption. The jobs of an affected order can also be classified as affected and unaffected.

Affected jobs (A): When a resource-related disruption starts, any job or job set of an affected order (which has already started processing by the machines), which is waiting for processing in the first machine, is considered as affected job. It is important to point out that even if the disruption starts in any other machine (except the first machine), the list of affected jobs of an affected order contains the jobs which have not started processing in the first machine. The reason for this is that at the beginning of every disruption, these jobs are the only part of an affected order that can be rescheduled. Besides, in multiple-order PFSPs, orders are interrelated. So any order, which is accepted and waiting in the queue, may also be affected. In that case, all jobs of that order are delayed and affected. For example, in Fig. 1, we consider a three-machine flow shop with two orders. Each order consists of five jobs. The job sequence for the 1st order is 4-2-1-5-3. Now assume that machine-3 breaks down when the first job (job-4) is processed by machine-3 (last machine). Meanwhile, jobs 1-5-3 are still waiting for machine-1. So jobs 1-5-3 are the affected jobs from the 1st order (identified with black rectangles). Also, assume that a 2nd order has arrived, been accepted and is waiting in the queue before the interruption starts. The sequence of that order is 5-2-4-1-3 (ash rectangles). Hence, that order is affected, as are all its jobs.

Unaffected jobs (U): A job that has already been processed by the first machine, before the disruption starts, is known as an unaffected job. Also, when the production interrupts, any job, which is involved with the disruption, is also included in the unaffected job list. As those jobs are already being processed by the system, they cannot be rescheduled or revised. So we call the jobs which are not affected jobs, the unaffected jobs. From the above example, the first two jobs (job 4–2) from the 1st order (identified by white rectangles) are unaffected.

For job-related disruptions, the concept of affected-unaffected jobs is similar to resource-related disruptions. The only difference is that in this case, the job that is resumed by the disruptions (such as in Fig. 1, job-4 from the 1st order) is also included in the affected job set. Since after a disruption event, this job can be revised or rescheduled. The detailed classifications of affected orders are described in the next section [31–34].

Each disruption event is caused by different reasons. Based on an examination of those events, different rescheduling actions can be taken, which are summarized in Table 1. With the right shifting strategy, after an interruption occurs and affected jobs have been delayed, the initial schedule is repaired, by pushing those jobs towards the right of the schedule. However, in real-time strategies, after every disruption, rescheduling approaches are used to improve the schedule. A job processing flow diagram for multiple-order PFSPs can be found in Appendix 1.

3.2 Assumptions

The assumptions made for multiple-order PFSPs are as follows: The processing time of each job on each machine is known; no pre-emption is allowed, i.e. a process once started cannot be stopped until it is finished; each machine can handle only one job at a time; the processing order of the operations of each job is predefined; work-in-process inventory, finished goods inventory; inter-machine transportation and set-up costs are negligible; each machine operates at its full efficiency; each order consists of multiple jobs; the arrival pattern of orders is stochastic; the customer sets the due date of each order; the scheduler has the right to accept or reject any new order; an accepted order cannot be rejected later (even if the order is tardy due to process interruptions); and there is financial benefit for early completion and delivery. However, a penalty or discount is also imposed for tardiness.

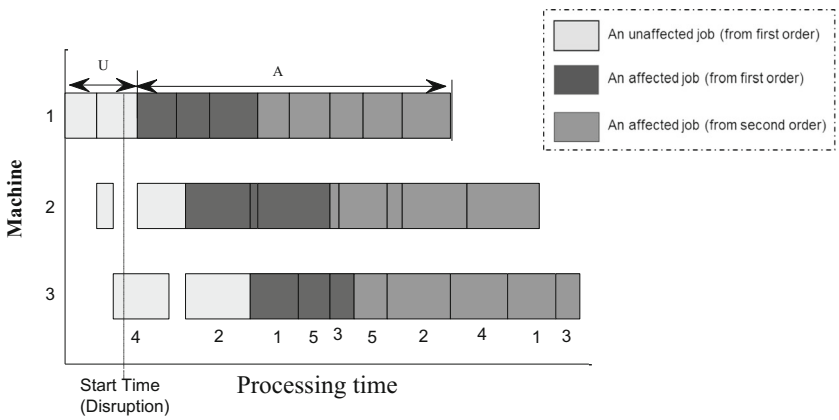

Fig. 1 Illustration of affected and unaffected jobs

Table 1 The details of interruption events and solution approaches

Interruption events	Causes	Right shifting strategy	Real-time strategy
Machine breakdowns [31–39, 43–45, 52–55]	Broken machine, operator illness, tool failure, raw material shortage	Right shift the affected jobs of an order and also the inter-related orders	Reschedule the affected jobs
Preventive maintenance [31–34, 45, 47, 55]	Overhaul machines	Consider as "dummy job" and right Shift the affected jobs of the order	Reschedule the affected jobs of the order, with respect to the unavailability constraints
Rework [31–34]	Scrap, impure raw materials	Start processing the scraped job of an order in the first machine as early as possible	Reschedule the affected jobs of that order
Change in processing times [31–34, 43]	Change in quality/specifications	Start processing the scraped job of an order in the first machine as early as possible	Reschedule the affected jobs of that order
Order cancellations [31–34]	Cancel the orders by customer	Process all other orders with the original schedule	Reschedule all the orders waiting to be processed in the first machine
Change in due dates [31–34, 46]	Due date changed by the customers	Process all other orders with the original schedule	Reschedule all the orders waiting to be processed in the first machine

3.3 Mathematical model

Table 2 presents the notations that are used in the development of the mathematical model and the proposed algorithm.

To propose a methodology for the above three research questions, we first need to generate schedules for static PFSPs.

3.3.1 Mathematical model of static PFSPs

Finding the $C_{max}(\pi^*)$ of a static PFSP is a preliminary step in developing methodologies for the three issues indicated in Section 3.1 (second paragraph). The major considerations of static PFSPs are (1) the capacity of each machine, (2) starting and finishing time of each operation in each machine, (3) technological constraints, (4) flow of operations, (5) preemption and (6) work-in-process inventory. Based on these considerations, Wagner [56] introduced an integer programming model for the static PFSP. In the next level, this model has been extended for developing the model of multiple-order PFSPs with interruptions.

In a PFSP, a set of n jobs must be processed in a set of m machines with the same order of processing. Then, the competition time of the last job n on the last machine m (i.e. makespan or maximum completion time) can be represented as

$$\text{Makespan}, C_{max}(\pi) = C(m,n) \tag{1}$$

In this case, the objective is to identify a permutation sequence, π^* so that

$$C_{max}(\pi^*) \leq C_{max}(\pi), \forall \pi \in \Pi \tag{2}$$

3.3.2 Mathematical model of multiple-order PFSPs with interruptions

In the following section is a description of all costs, the objective functions and the constraints of multiple-order PFSPs. In this regard, all decision variables are first derived.

Decision variables:

$$AZ_{ik}{}^{l} = \begin{cases} 1, \text{if the } i\text{th job of the } l^{\text{th}} \text{ order is scheduled in the } k^{\text{th}} \text{ position of the sequence} \\ 0, \text{otherwise} \end{cases}$$
$$\text{for } (i,k = 1,2,3,\ldots,n; l = 1,2,3,\ldots,L) \tag{3}$$

$AX_{kj}{}^{l}$= idle time of the jth machine before the start of the job of the lth order in the kth position in the sequence

$$\text{for } (k = 1,2,3,\ldots,n; j = 1,2,3,\ldots.,m; l = 1,2,3,\ldots,L) \tag{4}$$

$AY_{kj}{}^{l}$= waiting time for a job in between two machines, i.e. idle time of a job in the kth position in the sequence, after finishing processing on the jth machine while waiting for the $(j+1)$th machine to become free.

$$\text{for } (k = 1,2,3,\ldots,n; j = 1,2,3,\ldots.,m; l = 1,2,3,\ldots,L) \tag{5}$$

The relationship between the processing times of the ith job, scheduled on the kth position on the jth machine, is expressed by Eqs. (6) and (7)

$$p_{i(k+1),j} = \sum_{i=0}^{n} AZ_{i,k+1}{}^{l} p_{ij} \tag{6}$$

$$p_{i(k),j+1} = \sum_{i=0}^{n} AZ_{i,k}^{l} p_{ij+1} \tag{7}$$

$$By_{ij}^{l} = \begin{cases} 1, \text{if the } i^{\text{th}} \text{ job of the } l^{\text{th}} \text{ order is processed in } j^{\text{th}} \text{ machine} \\ 0, \text{otherwise} \end{cases} \quad \text{for}(i = 1,2,3,\ldots,n; j = 1,2,3,\ldots,m; l = 1,2,3,\ldots,L) \tag{8}$$

$$Bz_{iqj=1^{l}} \begin{cases} 1, \text{if job } i \text{ proceeds immediately after job } q \text{ of the } l^{\text{th}} \text{ accepted order is processed in the } 1^{\text{st}} \text{ machine} \\ 0, \text{otherwise} \end{cases} \quad \text{for}(i = 1,2,3,\ldots,n; j = 1,2,3,\ldots,m; l = 1,2,3,\ldots,L) \tag{9}$$

The makespan of the lth order is the maximum completion time of the nth job (last job in the sequence) of the lth order in the mth machine

$$C^{l}{}_{max} = C^{l}(m,n) \tag{10}$$

The completion time of the lth order can be expressed as

$$C^{l}{}_{com} = S^{l}{}_{j=1} + C^{l}{}_{max} \tag{11}$$

Objective function:

The total profit in one production shift (which is our objective) can be expressed as

$$\text{Maximize}, P = \sum_{l=1}^{L} P^{l} \tag{12}$$

where the profit or loss of an order l can be calculated as follows:

$$P^{l} = Sp^{l} + E_{l} \times C_{e}{}^{l} - T_{l} \times C_{t}{}^{l} - Op^{l}, l = 1,2,3,\ldots\ldots\ldots,L \tag{13}$$

Constraints A summary of all constraints is as follows:

Equation (14) ensures that each job follows the same order of machines (this constraint holds the characteristics of a PFSP)

$$\sum_{j=1}^{m} y_{ij}^{l} = m \tag{14}$$

Equation (15) ensures that each job is assigned to only one position in the sequence

$$\sum_{i=1}^{n} AZ_{ik}{}^{l} = 1 \quad \text{for}(i = 1,2,3,\ldots,n; l = 1,2,3,\ldots,L) \tag{15}$$

Equation (16) ensures that each position of a sequence is occupied by only one job

$$\sum_{k=1}^{n} AZ_{ik}{}^{l} \quad \text{for}(k = 1,2,3,\ldots,n; l = 1,2,3,\ldots,L) \tag{16}$$

Equation (17) ensures that once a job starts to process in the first machine, it cannot be rescheduled

$$\text{If } Bz_{iq\ j=1}{}^{l} = 1, Bz_{qi\ j=1}{}^{l} = 0 \text{ or } Bz_{iq\ j=1}{}^{l} = 0, Bz_{qi\ j=1}{}^{l} = 1 \text{ for } (i = 1,2,3,\ldots,n, q = 1,2,3,\ldots,n; l = 1,2,3,\ldots,L) \tag{17}$$

Equation (18) ensures that the first job of the first accepted order begins at 0

$$S^{l}{}_{j=1,i=1} = 0 \tag{18}$$

Equation (19) ensures that in this case, the earliness of an order is

$$E_{l} = d_{l} - C^{l}{}_{com,}, l = 1,2,3,\ldots\ldots\ldots,L \tag{19}$$

Equation (20) ensures that the tardiness of an order is

$$T_{l} =^{l}{}_{com} - d_{l}, l = 1,2,3,\ldots\ldots\ldots,L \quad d_{l} > 0 \tag{20}$$

Equation (21) ensures that a negative value of tardiness equals to the earliness of an order

$$E_{l} = -T_{l} \tag{21}$$

Equation (22) ensures that an order is accepted if it is completed within its due date

Table 2 Notations

Notation	Meaning
m	Number of machines on the shop floor
l	Index of orders
L	Number of orders that arrive for one production shift
H	A set of accepted orders ($H \in L$)
$n\ (l)$	The set of jobs of the lth order
i	Index of a job
j	Index of a machine
$p_{i,j}$	The processing time of each job i ($i \in n$) on each machine j ($j \in m$)
(i,j,k,l)	The ith job of the lth order is in the kth position in the sequence, and it has been processed on the jth machine
π	Permutation sequence of n jobs, $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$
π^*	Best permutation schedule found by the algorithm
Π	Set of all feasible job sequences
p_c	Crossover rate
p_m	Mutation rate
$C(i,j)$	The completion time of job i on machine j is $C(i,j)$
C_{max}	Makespan of a static PFSP
T_l	Tardiness of order l
E_l	Earliness of order l
S^l_j	Starting time of the first job of order l on machine j
C^l_{max}	The makespan of order l under static conditions
C^l_{com}	The completion time of order l
$C^l(i,j)$	The completion time of job i of order l on machine j
d_l	Due date of order l (also known as the delivery time)
P^l	Profit from the lth order
P	Total profit
Sp^l	Revenue from the lth order
Op^l	The opportunity loss cost of the lth order
C^l_t	The tardiness costs per unit time for the lth order
C^l_e	The earliness bonus per unit time for the lth order
BKS	Best known solution
BS	The solution generated by the solving technique
IR	Number of independent runs for each instance
c	Current clock time
S^l_j	Start time of the jth job of the lth order
N^p	Total number of potential orders
t_l	Is the current time of the clock time when the first job of the lth order starts processing in the first machine

$$T_l \leq 0, l = 1, 2, 3\ldots\ldots\ldots, \tag{22}$$

Equation (23) ensures that the lth accepted order is completed at the same time or before the $(l+1)$th accepted order

$$C^l{}_{com} \leq C^{l+1}{}_{com}, l = 1, 2, 3, \ldots\ldots\ldots\ldots, L \tag{23}$$

3.4 Upper and lower bound calculations

To ensure validity and to measure the quality of the solutions generated by the proposed approach, it has been compared with upper and lower bounds. In this section, these upper and lower bounds are derived. Assume that an order l arrives at time At^l and that it has been accepted by the flow shop. So the machines will be inactive for the $(At^l - C_{com}{}^{l-1})$ time period, as that is the time between the completion of job order $(l-1)$ and the arrival time of order l, when $(At^l - C_{com}{}^{l-1}) > 0$. The positive time difference, DT^l, can be expressed as follows:

$$DT^l = \begin{cases} At^l - Ccom^{l-1}, & if\ \left(At^l - Ccom^{l-1}\right) > 0 \\ 0 & \text{otherwise} \end{cases}, \forall l \geq 2 \tag{24}$$

Suppose that $C_{max}{}^l$ is the makespan of the static PFSP for order l and the number of jobs in the order is n^l. If l is an affected order for any reason, some or all jobs of the order may be delayed, in which case, the completion time may change to $C_{max}{}^d$. Where

$$Cmax^d = \begin{cases} Cmax^l + BD^{il}, & \text{if resouce related} \\ Cmax^{JR} & \text{else if job related} \end{cases}, C_{max}{}^d \neq C_{max}{}^l \tag{25}$$

$$C_{max}{}^l = C_{max}{}^d, \quad \text{Only if the disruption occurs within that order} \left(t_{fl} \geq t_d > t_{sh}\right) \tag{26}$$

That means the disruption starts on the ith machine at time t_d, which is between the start time of the first job of order l on that machine (t_{sl}) and the finishing time of the last job on the same machine (t_{fl}). BD^{il} is the duration of the disruption if the disruption relates to the machines. If the disruption relates to jobs, then the completion time, $C_{max}{}^{JR}$, is expressed as

$$Cmax^{JR} = C^l\left(m, n^l\right) \tag{27}$$

$$\text{Only for,} \quad t_{sl\alpha} = C^l\left(m, n^{l\beta}\right) \tag{28}$$

Expression (27) represents that the first job from an affected job set (number of jobs in the affected job set is $n^l - n^{l\beta}$) starts at time, $t_{sl\alpha}$, when the last job β from the unaffected job set (number of unaffected jobs is $n^{l\beta}$) from order l is completed on the last machine.

3.4.1 Upper and lower bounds

The upper bound (UB) is defined as the worst-case scenario to complete all accepted orders in a given working shift. To

calculate it, if the total number of accepted orders is L, then the UB for H accepted orders ($H \in L$) can be expressed as

$$\mathrm{UB} = \sum_{l=1}^{H} Cmax^{l} + \sum_{l=2}^{l} DT^{l} \tag{29}$$

Suppose that $Pt^{h}{}_{1}$ is the total processing time required for order l on machine-1. If $(At^{l} - Pt^{l-1}) < 0$ and $(Pt^{l-1} < Ct^{l-1})$ machine-1 starts to process order h as soon as it completes order $l-1$. In this case, after finishing order $l-1$, order l starts processing, and that will require $C_{max}{}^{l}$ units of time to complete all the processes. Also suppose that just after completing order $l-1$ on machine-1, order l arrives in the shop. If we ignore the effect from disruptions, then this is the best-case scenario for processing all the accepted H orders. Suppose that the time for completing order l on machine-1 is $CJt^{l}{}_{1}$. The reduction in processing time (RP^{l}) can be expressed as

$$RP^{l} = \begin{cases} C_{max}{}^{l} - Pt^{l}_{1}, & if\ \left(At^{l} - Pt^{l-1}\right) < 0, Pt^{l-1} < Ct^{l-1} \text{ and } At^{l} = CJt^{l}_{1}, \forall l \geq 2 \\ 0 & \text{Otherwise} \end{cases} \tag{30}$$

and processing all H accepted orders can be expressed as

$$\mathrm{LB} = \sum_{L=1}^{H} Cmax^{l} - \sum_{l=2}^{H} RP_{1}^{l-1} + \sum_{l=2}^{H} DT^{l} \tag{31}$$

So lower bound (LB) represents the best-case scenario of completing all H accepted orders.

4 Solution approach

In this section, the solution approach is presented. The detailed framework of the proposed approach is presented in Fig. 2. Following this figure, we first discuss the approach for order acceptance and rejection. Secondly, we describe the algorithm design that will be used both for generating an

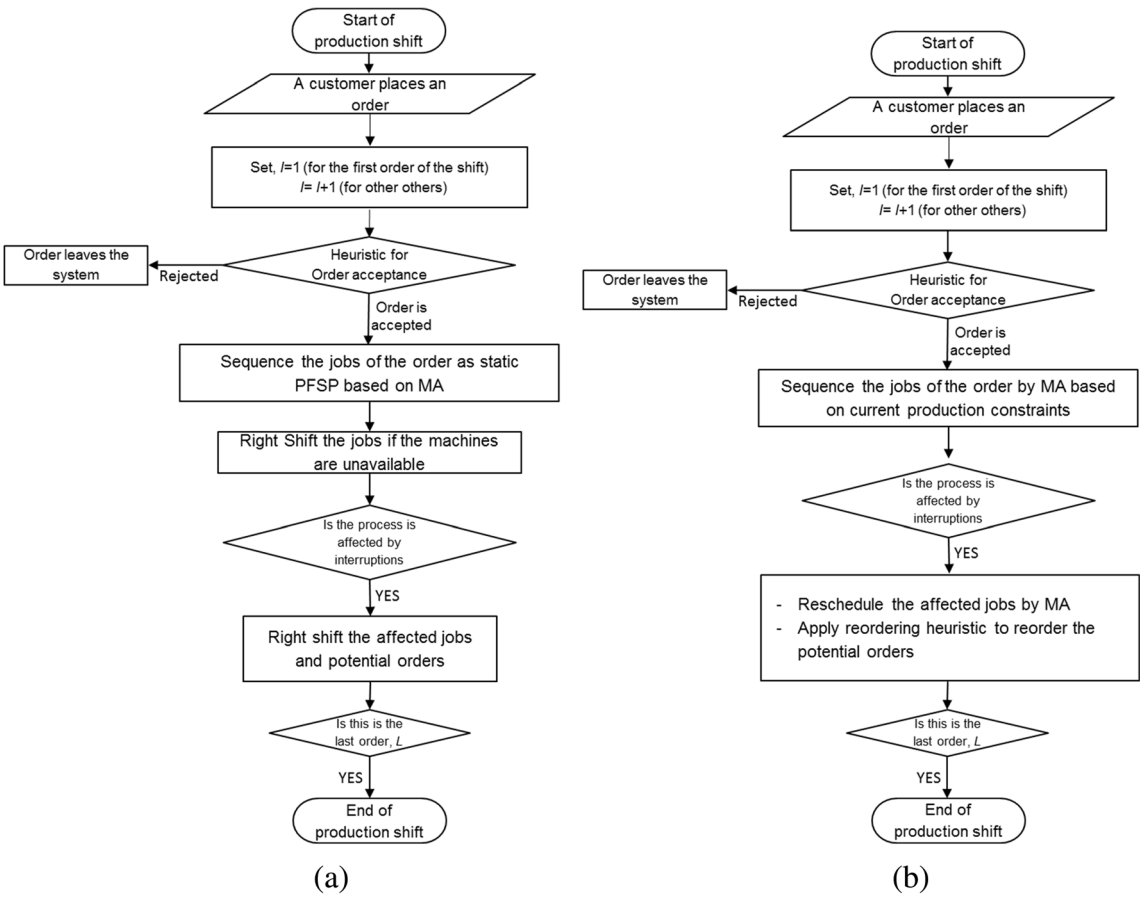

Fig. 2 Flowchart describing the proposed framework. **a** Right shifting strategy. **b** Real-time strategy

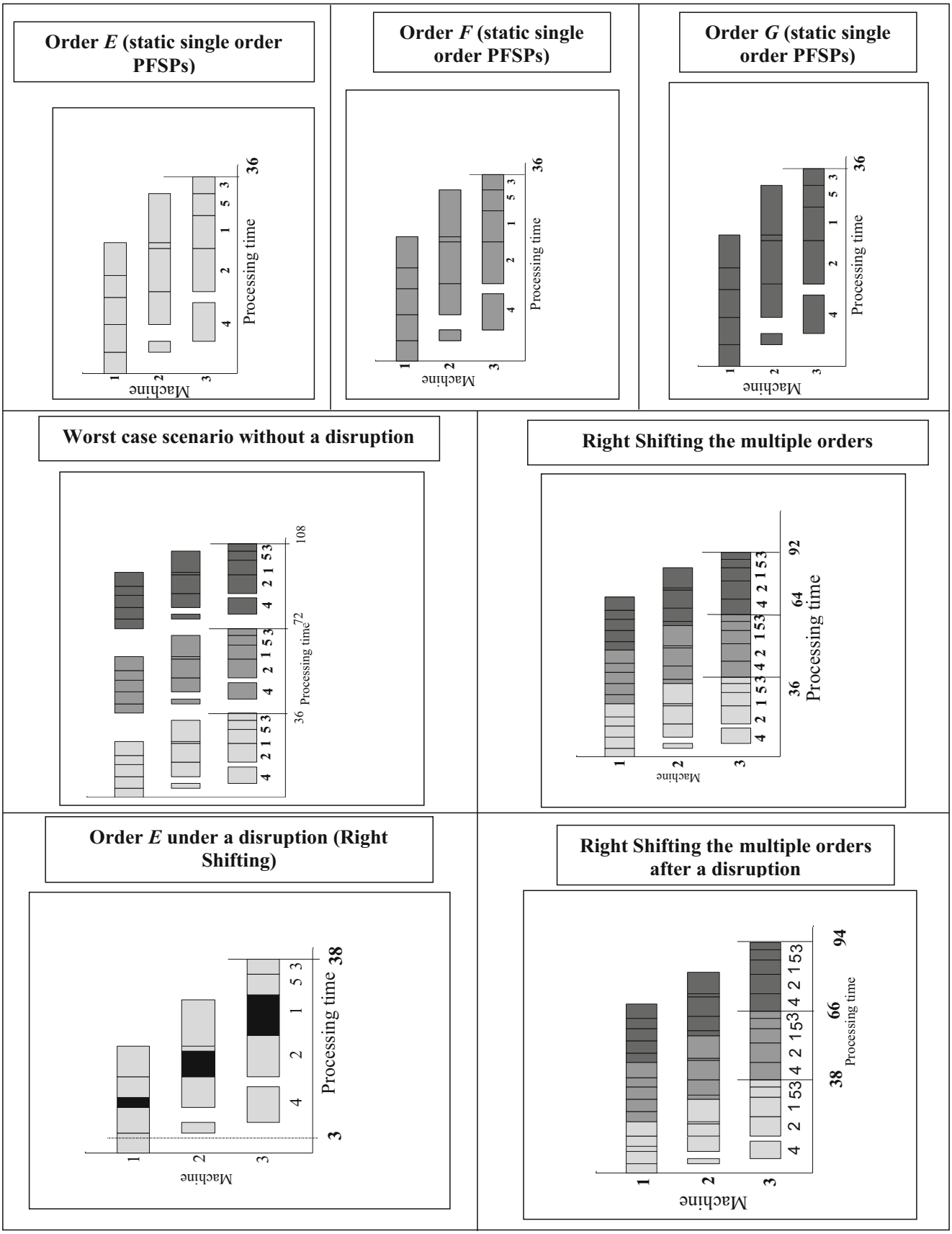

Fig. 3 Right shifting (RS) strategy with disruption

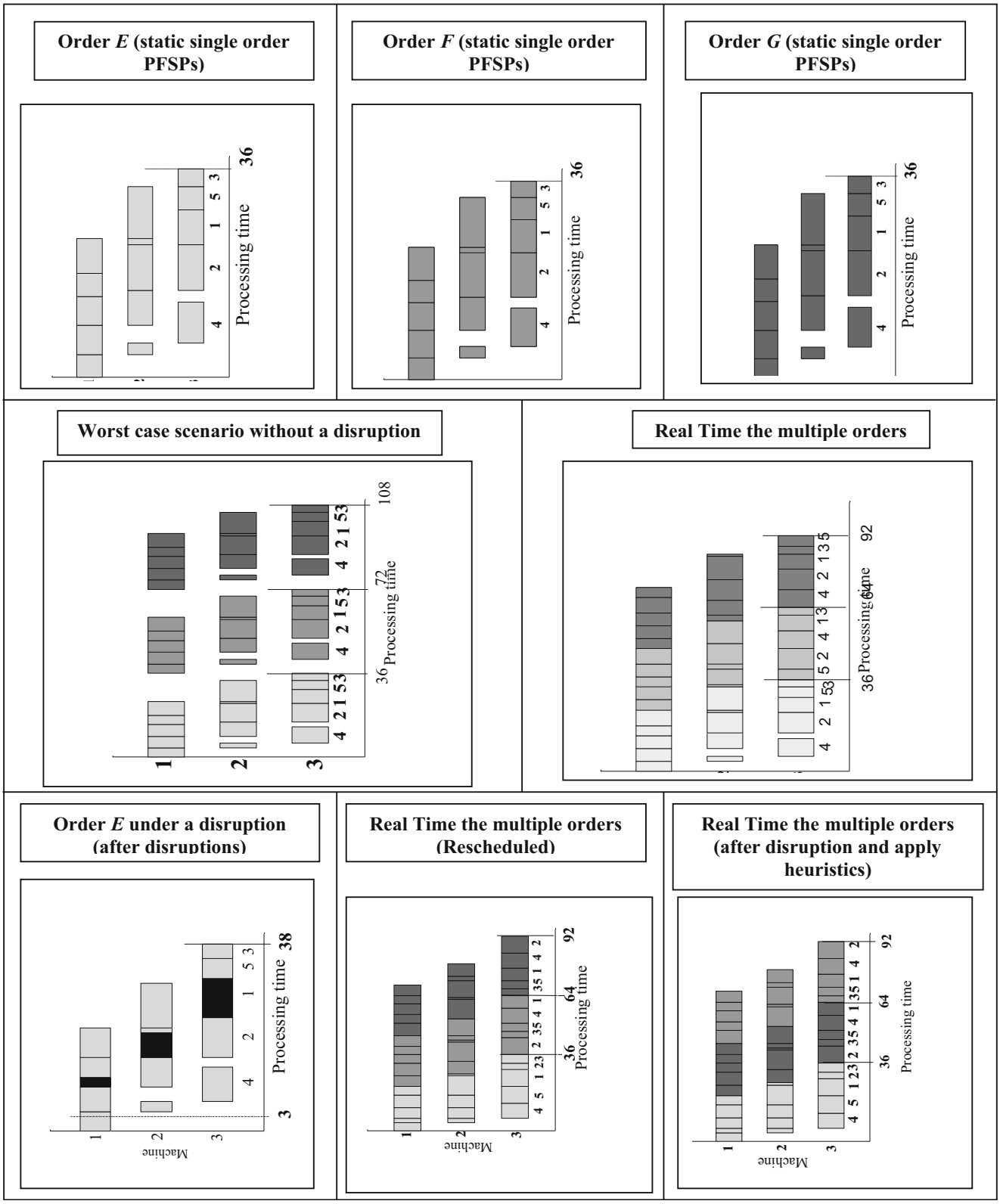

Fig. 4 Real-time (RT) strategy with disruption

initial schedule and for also rescheduling in the second and third decision problems. Our proposed algorithm is a genetic algorithm (GA)-based memetic algorithm (MA). Later, we discuss how this algorithm has been implemented to study both single- and multiple-order PFSPs with process disruptions.

4.1 Order acceptance/rejection

Whenever an order arrives to the production floor, it is important to make an order acceptance/rejection decision. This decision depends on the order arrival and start time, available production capacity, due date, completion time and the profitability of each order. An order should be accepted if it can be completed within the customer specified due date while generating a reasonable profit. If an order is rejected, an opportunity loss cost may be considered, which relates to loss of the company's goodwill. In the acceptance/rejection decision of multiple-order PSFP, we calculate the profit or loss for a certain length of time.

The heuristic for the order acceptance/rejection decision is described as follows. Note that the completion time is calculated using the scheduling algorithm presented in the next section.

1. The production shift starts and the first order arrives.
2. Set order number, $l = 1$
3. Repeat until ($l = L$)

A. Evaluate C^l_{com} of an order l.
B. If ($C^l_{com} \leq dl$), **Accept** the order and put it in the accepted order list.
C. Else **Reject** it.
D. Set, $l = l + 1$.

[End of step 2 loop]
[End of algorithm]

4.2 Proposed memetic algorithm

MAs are global meta-heuristic search algorithms that are based on Darwin's principle of natural evolution and Dawkin's notion of a meme that is individual learning [57]. MAs can be considered as the integration of a population based search algorithm (like GAs), with a constructive local search algorithm. Even though, GA is a widely accepted global search technique for solving complex combinatorial optimization problems. Usually, a GA integrated with a local search algorithm, or a GA-based MA, performs better than just the GA itself. MAs have a successful history for solving complex combinatorial optimization problems [58–61]. For this reason, a GA-based MA has been considered as the prime method for solving PFSPs in this research. GA starts with an initial population, which consists of a set of initial solutions. The population is then improved, iteratively, through alteration, while usually using three reproduction operators, known as selection, crossover and mutation. The process continues until some stopping criteria are met. Each solution is represented by a chromosome. The chromosome, search operators and parameters used for the GA to solve different PFSPs are described below.

4.2.1 Initial population and solution representation

In GA, chromosomes are usually represented by binary, integer or real numbers. For solving PFSPs, an integer representation of the job sequence is widely accepted. Traditionally, GA starts with a random initial population. However, for solving PFSPs, random initialization is not helpful as the solution becomes trapped very quickly in local optima. We have proposed using a mixed (both random and non-random) initialization. The first individual of the initial population is generated by the Nawaz, Enscore and Ham (NEH) algorithm [10]. Then, some individuals are generated from original modification of that individual, by randomly swapping two jobs from the NEH-based sequence. Next, a certain percentage of the initial population is generated by Johnson's algorithm. In that

Table 3 Comparisons between the algorithms (for single-order PFSPs)

Problem	Problem instances	NEH	HGA_RMA	PSO_{VNS}	$NEGA_{VNS}$	GA	MA	$MA_{Adaptive}$
20 × 5	Ta001–010	3.26	0.04	0.03	0.0	0.35	*0.0*	*0.0*
20 × 10	Ta011–020	4.6	0.02	0.02	0.01	1.14	*0.01*	*0.01*
20 × 20	Ta021–030	3.73	0.05	0.05	0.02	0.83	*0.02*	*0.02*
50 × 5	Ta031–040	0.73	0.0	0.0	0.0	0.17	0.01	0.01
50 × 10	Ta041–050	5.07	0.72	0.57	0.82	1.78	*0.54*	*0.52*
50 × 20	Ta051–060	6.66	0.99	1.36	1.08	3.2	*0.77*	*0.74*
100 × 5	Ta061–070	0.53	0.01	0.0	0.0	0.07	*0.0*	*0.0*
100 × 10	Ta071–080	2.21	0.16	0.18	0.14	0.97	0.15	*0.14*
100 × 20	Ta081–090	5.34	1.30	1.45	1.40	2.71	1.35	*1.29*
200 × 10	Ta091–100	1.26	0.14	0.18	0.16	0.65	0.2	0.16
200 × 20	Ta101–110	4.41	1.26	1.35	1.25	2.18	1.28	*1.24*
500 × 20	Ta111–120	2.07	0.69	–	0.71	1.51	0.75	0.72
Average		3.332	0.448	0.472	0.466	*1.29*	*0.423*	*0.404*

Table 4 Scenario I

Order number	Single-order makespan (static)	Arrival time	Due date, d_l	Right shifting				Real time (earliness bonus per unit time per order)				Real time (longest due date)			
				Completion time, $C^l_{com}(RS)$	Tardiness, $T_l = C^l_{com}(RS) - d_l$	Profit ($)	Status	Completion time, $C^l_{com}(RT)$	Tardiness, $T_l = C^l_{com}(RT) - d_l$	Profit ($)	Status	Completion time, $C^l_{com}(RT)$	Tardiness, $T_l = C^l_{com}(RT) - d_l$	Profit ($)	Status
1	1582	*0*	*4536*	1582	−2954	8900.8	A	1582	−2954	8900.8	A	1582	−2954	8900.8	A
2	1582	*50*	*5983*	2961	−3022	7132.2	A	2806	−3177	7302.7	A	2806	−3177	7302.7	A
3	1582	*59*	*7202*	4198	−3004	7896	A	3998	−3204	8296	A	4198	−3004	7896	A
4	1582	*76*	*8007*	5377	−2630	8379	A	5218	−2789	8649.3	A	5377	−2630	8379	A
5	1582	*101*	*7013*	6555	−458	2528.8	A	6396	−617	2783.2	A	6555	−458	2528.8	A
1 (break-down)	2851	*0*	*4536*	2851	−1685	*5790.2*	A	2837	−1699	*6767.3*	A	2837	−1699	*6767.3*	A
2	1582	*50*	*5983*	4181	−1802	*5352*	A	7656	1673	*1967.7*	A	7656	1673	*1967.7*	A
3	1582	*59*	*7202*	5470	−1732	*6218.3*	A	6478	−724	*3336*	A	5300	−1902	*5692*	A
4	1582	*76*	*8007*	6648	−1359	*495.2*	A	4073	−3934	*10,596*	A	4073	−3934	*10,596*	A
5	1582	*101*	*7013*	7826	813	*−316*	A	5300	−1713	*4536.8*	A	6478	−535	*2652*	A
6	1582	*404*	*7353*	9004	1651	*−110*	R	8716	1363	*−316*	R	8746	1393	*−316*	R
7	1582	*1062*	*6752*	9004	2252	*−185*	R	8716	1964	*−110*	R	8746	1994	*−110*	R
8	1582	*2409*	*8498*	9004	506	*7896.6*	R	8716	218	*−185*	R	8746	248	*−185*	R
9	1582	*5366*	*12,772*	9004	−3768	*1809.4*	A	8716	−4056	*8386.2*	A	8746	−4026	*8335.2*	A
10	1582	*5384*	*10,558*	10,182	−376	*5790.2*	A	9894	−664	*2212.6*	A	9924	−634	*2170.6*	A
Total profit ($)				32,740.9				*37,191.6*				*37,569.8*			

Table 5 Same job order arrival, machine breakdowns

PS (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	1.566	*2.0452*	0.7964	*0.5010*	64	*67*	66,847	*72,654*
50	0.371	*0.4073*	0.2410	*0.2182*	75	*78*	87,456	*93,541*
100	0.165	*0.1866*	0.1788	*0.1725*	65	*67*	74,565	*76,514*
200	0.114	*0.1150*	0.0638	*0.0626*	73	*76*	75,191	*78,901*

PS problem size

case, more than two machine PFSPs are divided into multiple two machine PFSPs and these problems are optimally solved by Johnson's algorithm. The remaining individuals are generated randomly.

4.2.2 Selection and population enhancement

In this research, we have applied the tournament selection method [62] to select parents for the purpose of crossover. The parents are directly replaced by their offspring. In a tightly constrained combinatorial optimization problem, many duplicate individuals are usually produced through the evaluation process and these solutions can converge to local optima. To avoid this, in each and every generation, duplicated individuals are replaced by random solutions. It has been observed that different sequences (permutations) may have the same fitness value [17]. So to identify duplicate individuals, we compare the sequence of jobs. Also, if the iteration process becomes stuck with the same fitness value for some generations, a restart scheme [14, 17, 18] is applied. In this case, if the fitness value does not improve for a certain number of generations, all individuals are divided into three segments, good 5%, middle 85% and bad 10%. The individuals from the middle segment are replaced by new individuals through shift mutation [14, 17, 63] of the top segment, where a random job from the job sequence is directly inserted into a random position in the job sequence. Each job between these two points is then moved to its next position. We also use an elitism strategy; hence, the best sequence from each generation is saved and is directly transferred to its next generation.

4.2.3 Crossover and mutation

In the algorithm, we have used similar job order crossover (SJOX) [17] and shift mutation [14, 17, 63] as the reproduction operators, as they produced more promising candidate solutions to most of their counter parts in our preliminary experimentation. In SJOX, from both parents, each position from the job sequence is examined individually. If both parents have common jobs in the same position, then those jobs are directly copied to both offspring. Then, each offspring inserts all jobs from one of the parents up to a random crossover point. That is to say, parent 1 inserts jobs directly to offspring 1 and parent 2 inserts jobs directly to offspring 2. Lastly, the missing jobs of each offspring are directly inserted from the other parent in the same relative job order of that parent. For example, parent 1 inserts the missing jobs to offspring 2 and parent 2 inserts the missing jobs to offspring 1.

4.2.4 Local search

To improve the performance of the algorithm, we have incorporated a local search method within the GA. In this algorithm, three local search techniques have been applied. They are insertion neighbourhood, gap filling and a job shifting process. A selected individual initially goes through the insertion neighbourhood process and then undergoes the gap filling process and finally the job shifting method. The details of the gap filling and the job shifting processes can be found in [18]. However, in brief, their steps are as follows:

Table 6 Mixed order, same order size, machine breakdowns

PS (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT		RS	RT	RS
20	0.145	*0.185*	0.0543	*0.0382*	17	*18*	87,401	*87,656*
50	0.070	*0.077*	0.0435	0.0445	10	*13*	65,114	*66,789*
100	0.020	*0.021*	0.0074	*0.0063*	9	*11*	45,657	*46,415*
200	0.027	*0.031*	0.0050	0.0061	14	14	74,103	*75,541*

PS problem size

- *Insertion neighbourhood search*: Every job combination has $(n-1)^2$ neighbours. In this process, a job j is extracted from its current position k in the job sequence and is then inserted in another position p ($p \neq k$). Hence, each job is moved from its current position in the job sequence and is inserted into all possible pairs of positions. If a new job sequence is better than the current best sequence, it then replaces the current best sequence. This process continues until all the neighbours have been evaluated. Insertion neighbourhood has shown good performance in solving PFSPs [13, 17, 64].
- *Gap filling*: In solving PFSPs, no inter-job gaps are permitted when scheduling the jobs in the first machine. However, because of the precedence constraints, there may be some inter-job gaps left on all of the other machines. The makespan can hence be minimized by reducing those gaps. In this technique, the total processing time of all jobs in the sequence is calculated and the jobs with the smallest total processing times are inserted in those gaps. If the makespan or completion time improves with such movement, then the new sequence is accepted. Or else, the job with the second best least total processing time is inserted. A maximum of three trials have been allowed for each gap. Note that if one of the adjacent jobs has one of the least total processing times, then we have inserted the job with the next best least total processing time in the schedule.
- *Job shifting*: If the jobs have (i) shorter processing time at the upper left-hand side (first or first couple of machines) of the sequence and (ii) longer processing time at the lower right-hand side (last or last couple of machines) of the job sequence, the quality of the sequence (makespan or completion time) can be improved by moving the jobs from the right-hand side to the left-hand side in the sequence (starting these jobs as early as possible). The process is as follows: compute the ratio of processing times (processing time in the last machine divided by the processing time in the first machine) for each job in the sequence. Next, if one of the three last jobs has one of the three largest ratios (limited up to three ratios), those jobs are then positioned as early as possible in the sequence.

4.2.5 An adaptive scheme

In this section, we describe the proposed adaptive method used in the MA. In GA, population diversity can be maintained by both by selection process and the genetic operators [65]. In MA, we control diversity by the second method.

Preliminary experiments show that in tightly constrained combinatorial optimization problems like PFSPs, a higher crossover rate intensifies the solutions, as more individuals exchange information with each other, and so create many duplicated individuals. This essentially converge the solutions to local optima. On the other hand, too much diversification reduces the convergence rate. To ensure good balance between intensification and diversification, we developed an adaptive crossover rate, p_c, and mutation rate, p_m, for MA, and the modified algorithm is known as adaptive MA ($MA_{Adaptive}$). In our adaptive strategy, we set the crossover rate low at the start of algorithm, but with the progress of the algorithm, the crossover rate increases.

We adaptively calculate the crossover rate as

$$p_c = \mathrm{Max}\left(\partial^c\mathrm{min}, {}^c\mathrm{min} + (\partial^c\mathrm{max} - \partial^c\mathrm{min}) \times (\mathrm{Gen}/\mathrm{Max\_Gen}) \right. \quad (32)$$

where ∂^cmin is the lowest limit of crossover rate, ∂^cmax is the highest limit of the crossover rate, Gen is the number of generation and Max_Gen is the maximum number of generation.

For the mutation rate, we set the mutation rate high at the start of algorithm, and with the progress of the algorithm, the crossover rate decreases.

The mutation rate is calculated as

$$p_m = \mathrm{Max}(\partial^m\mathrm{min}, {}^m\mathrm{max} - (\partial^m\mathrm{max} - \partial^m\mathrm{min}) \times (\mathrm{Gen}/\mathrm{Max\_Gen})) \quad (33)$$

where ∂^mmax is the highest limit of the mutation rate, and ∂^mmin is the lowest value of the mutation rate.

4.3 Proposed approach

Once a new order arrives in the flow shop, it may not be possible to immediately process the order due to machine unavailability. However, if the first machine is available when a new order arrives, the order can start processing without interrupting other orders being processed on other machines. Alternatively, if a new order arrives when the first machine is busy, the order has to wait in a queue for processing. Changes in production capacity that are due to disruption may also have some effects on the accepted orders in process and the orders waiting in the queue. Rescheduling the accepted orders can diminish these effects. In this research, we propose two new strategies to schedule or reschedule, namely the right shifting (RS) strategy and the real-time (RT) strategy. The RS strategy is a traditional approach where some or all jobs of each order are shifted to the right if the consecutive machine or machines are not available to start processing the jobs immediately. On the other hand, for the RT strategy, jobs of each order are re-optimized, based on the availability of machine/machines. As there is no state of the art algorithm with known benchmarks, for validation, we have compared RT with RS, upper bound and lower bound solutions.

Table 7 Mixed order, different order sizes, machine breakdowns

Problem Instances	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
1	0.292	0.144	0.0285	*0.023*	19	*21*	189,474	*190,471*
2	0.335	*0.335*	0.0071	*0.006*	25	*25*	188,743	*190,119*

4.3.1 RS strategy

In RS, each order is considered as one static PFSP, and the initial schedule of the jobs within each order was generated using our proposed algorithm. In the RS strategy, the first machine starts processing the first job of an accepted order if this machine is free, and otherwise, the job waits. After processing on any machine, one or multiple jobs may need to wait until a required machine is free. That means, some jobs must be shifted to the right due to waiting for machines. In this case, we do not change the sequence of jobs within an order. Here, the completion time of an order is usually greater than its static PFSP makespan. Similarly, after any disruption, the affected jobs of the affected orders can be shifted to the right. We call this strategy as the right shifting (RS) strategy [51, 53].

In the RS strategy, some or all of the jobs of an order which are affected/delayed and the jobs of all the interrelated orders are shifted to the right of the schedule.

Figure 3 shows an example of RS for multiple-order PFSP in Gantt chart form. It is a five-machine and three orders (say orders *E*, *F* and *G*) flow shop instance. Each order (contains of five jobs) has different arrival times and due dates (delivery times). In the figure, the static single-order *E* is represented by the Gantt chart in the top left hand corner. The job sequence of this order is 4-2-1-5-3, and the completion time (single-order makespan) is 36. The customer-specified due date for order *E* is 45. Assume that the inter-arrival time between order *E* and order *F* is two units and that order *F* has a due date of 70. The single-order makespan for order *F* is 36, and the job sequence for order *F* is also 4-2-1-5-3. Order *F* cannot start to process immediately in machine-1 as it is still processing the jobs of order *E*. In the worst-case scenario, order *F* starts to process immediately once order *E* has completed processing. In this case, the completion time for order *F* is 72 (= 36 + 36). However, if order *F* can be initiated immediately when machine-1 (1st machine) is free after processing order *E*, this completion time can be reduced. But after processing in machine-1, the first few jobs of order *F* complete processing on machine-1, and machine-2 is engaged in processing order *E*. To generate a feasible schedule, jobs from order *F* must be right shifted in machine-2. This simple shifting approach is known as the right shifting (RS) strategy. With the RS strategy, the completion time for order *F* is 64 (in Fig. 4 the Gantt chart in the middle right corner). Let, after 4 units of time from the arrival and starting of order *E*, for order *G* to arrive in the system with a due date of 100. In the worst case, order *G* has completion time 108 (=36 + 36 + 36) which exceeds its due date (delivery time). However, by applying the RS strategy, the completion time is reduced to 92. As the given due date is 100, order G can now be accepted.

Assume also that after 5 units of time from the start of order *E* (when the first job of order *E* is in process in machine-2) that the processing time of the third job (job-1) in the sequence is changed (disruption). So order *E* is a work-in-process affected order (WIPA). In Fig. 3, the Gantt chart in the lower left hand corner represents order *E* after disruption. After disruption, the changed completion time for order *E* is 38. In this case, the processing times of consecutive orders overlap (order *F* overlaps with order *E* and order *G* overlaps with order *F*). So after that disruption, the completion time for the other orders may be changed. The new completion time for order *F* is 66 and for order *G* is 94 (Gantt chart in bottom right corner). Basically, the RS strategy is a tighter upper bound than the worst-case scenario.

The algorithm for RS strategy with interruptions The algorithm for the right shifting (RS) strategy with process interruptions is described below.

L is the total number of orders placed by the customers in a single production shift. *c* is the current clock time, d_l the due date, sales revenue is Sp^l and the opportunity loss cost is Op_n of the *l*th order. Also, the start time of the *j*th job of the *l*th order is S^l_j, T_l is the tardiness cost per order and t_l is the current time of the clock time when the first job of the *l*th order starts processing in the first machine. The algorithm must calculate the start time t_l of the *n*th order, its completion time C^l_{com}, makespan C^l_{max}, its tardiness, T_n, profit P_n and total profit, *P*.

When the initial order arrives in the system, the algorithm starts.

[Start of algorithm]

1. Set order number, $l = 1$, clock time $t_l = 0$, and start time $S^l_1 = 0$
2. Repeat until $l = L$ (until the production shift ends)

a. update the clock, $t_l = c$, and $S^l_1 = c$
b. Use MA to generate a single order schedule, makespan, C^l_{max} for the *l*th order.
c. With respect to makespan, C^l_{max}:

I. Calculate completion time, $C^l_{com} = S^l_l + Cl_{max}$
II. Calculate tardiness, T_l and earliness, E_l
III. If $T_l \leq 0$, **Accept** the order
IV. Else, **Reject** the order
V. Calculate the profit/loss of the lth order, $P^l = S_p{}^l + E_l \times C_e{}^l - T_l \times C_t{}^l - O_p{}^l$ (using Eq. (13)

d. After any process interruptions at time, t_d

i. Identify the affected and unaffected jobs of the affected orders currently under process (WIPA)
ii. Right shift the affected jobs of that order and calculate the new completion time, C^l_{com}
iii. Identify the potential orders waiting in the queue.
iv. Right shift all the jobs of these orders and calculate the completion time, C^l_{com} of these orders
v. Set $l = l + 1$

[End of step 2 loop]
3. Calculate and save the total profit, P, using Eq. (12).
[End of algorithm]

4.3.2 RT strategy

In the real-time strategy, we revise the schedule (i.e. reschedule) as soon as one of the following events happens (to simulate real-time action): (1) arrival of a new order, (2) re-ordering the accepted orders, (3) the system is interrupted and (4) multiple orders are under disruption. The strategy with respect to these events is described below.

Arrival of a new order: Once an order arrives in the shop, a hypothetical schedule of that order, considering the constraints for production (available system capacity at that point in time and the orders currently under process), is made. On the basis of that schedule, an acceptance/rejection decision must be made on the grounds of the local view of whether or not the jobs of an order can be completed on time within the due date (and hence is profitable). Examples of this local decision are given below.

We have provided a simple example with Gantt charts to demonstrate the overall acceptance/rejection process. Here, we have considered a three machine flow shop with three orders (say orders E, F and G) where each order contains five jobs. These orders have different arrival and delivery times. In Fig. 4, the Gantt chart in the top left-hand corner represents the schedule of a static single-order E that can be generated using our algorithm. In order E, the sequence of jobs is 4-2-1-5-3 with a makespan of 36; we also assume that the due date is specified as 45 time units. So order E can be accepted and the process can be started on machine-1 immediately (at time t = 0), assuming that the machine is free. Suppose, at $t = 2$, order F arrives with a due date of 70. The sequence of jobs for this order is also 4-2-1-5-3 with a makespan of 36. Order F cannot be processed immediately on machine-1 as order E is still in process. In the worst-case scenario, order F can start processing, after order E has finished processing all its jobs. In which case, order F will be completed at $t = 72$ (=36 + 36). This is basically the maximum time required for completing these two orders (which can be interpreted as an upper bound; shown in the middle-left figure). However, the completion time of order F may be reduced if it can start processing as soon as machine-1 completes the processing of order E. When the first job of order F completes processing on machine-1, machine-2 is still engaged in processing order E. So order F must wait until machine-2 is free from order E and so on. In this process, we keep the same sequence of jobs for order F as was generated for it as a static PFSP. However, the completion time of order F can be further reduced, if it is rescheduled with the machine availability time windows with respect to order E. The new job sequence for order F can then be 2-3-5-4-1, and that gives a completion time of 64 (see middle-right figure). Since the completion time of order F (which is 64) is lower than its due date (which is 70), the order can be accepted. Finally, order G arrives at $t = 4$ with a due date of 100. Similar to order F, after rescheduling order G, the completion time of order G is obtained as 92. So order G should be accepted. In this paper, this process is referred to as real-time (RT) strategy. Note that, as of the middle-left figure, the worst-case completion time for order G is 108 (= 36 + 36 + 36).

Re-ordering the accepted orders: In the previous sub-section, we have considered an example with three orders (E, F and G) and made their acceptance/rejection decisions, in sequence of their arrivals, based on the local feasibility view. That means, after the arrival of G, we assumed that all orders up to F have been scheduled and we are examining the feasibility of accepting G. This feasibility does not mean that all orders (including G) are scheduled in the best possible way. For example, in the acceptance/rejection decision, we placed order G after order F. However, the performance may be improved by placing order G before order F. As the makespan, arrival time, start time and due date vary from order to order, an improved solution may be generated by rescheduling all the accepted orders together, instead of joining the new order at the bottom of the list of accepted orders. We now consider all the accepted orders waiting in the queue as potential orders. By rearranging the potential orders, the productivity of the system may be improved. This process will work even better where there is a defined benefit for early completion and/or penalty of late completion. To rearrange the potential orders, we have proposed a re-ordering heuristic.

System interruption: If a production system is disrupted, some or all jobs of the order (or orders) under process may be affected and delayed. These orders are WIPA. If the flow shop is fairly busy, any disruption can affect not only the WIPA but also other orders waiting in the queue, by delaying their completion times. In this case, it is necessary to

reschedule all orders under process and also re-sequence all orders waiting in the queue while considering the disruption recovery time window. In this case, after a disruption, the orders waiting in the queue can be considered as potential orders

For rescheduling the potential orders, we have proposed the following heuristics.

In Fig. 4, the Gantt chart in the lower left-hand corner shows order *E*, after disruption, with a revised schedule. Assume that at $t = 3$, when machine-1 was processing job-1, job-2 (3rd job of order *E*) was disrupted (processing time of that job had suddenly been changed). At the beginning of the production period, the processing times for job-1 were 5 (in machine-1), 8 (in machine-2) and 6 (in machine-3) respectively. But at $t = 3$, those time periods were changed to 2 (in machine-1), 5 (in machine-2) and 8 (in machine-3). Now, with the right shifting approach, the revised completion time for order *E* becomes 38. According to our definition here, the first job in the sequence (which is job number 4) is unaffected and the rest of the jobs (jobs 2-1-5-3) are affected. However, by rescheduling only the affected jobs, it is possible to reduce the completion time to 36 with a new job sequence of 4-5-1-2-3.

Multiple orders under disruption In the multiple orders case, the disruption with order *E* may change the completion time for other orders. Orders *F* and *G* are the potential orders, as they arrived, and were accepted and scheduled before the start of the disruption. As discussed in the previous section, these potential orders may be reordered and the jobs within each order can be rescheduled, based on the system capacity with respect to the time window. By rescheduling the jobs for orders *F* and *G* (with order sequence *E*-*F*-*G*), the new completion times for orders *F* and *G* are 64 and 92 respectively (bottom-middle figure). After reordering the potential orders, which is *E*-*G*-*F*, the revised completion times for orders *G* and *F* are 64 and 92 respectively (bottom-right figure).

Heuristic for re-ordering the potential orders *L* is the total number of orders and N^p is the total number of potential orders (N^p€*L*) at any time *t* (either for multiple-order arrival or after a disruption) and P^t is the total profits earned from these potential orders with the current ordering sequence. P^t can be calculated as follows:

$$p^t = \sum_{l=1}^{N^P} P^l \tag{34}$$

The potential orders are sorted in ascending order of earliness benefit per unit time per order. Besides the cost criteria, these orders are also sorted according to their due dates, which can be considered as an alternative approach. That is, orders are sorted from the latest to the earliest due dates.

[Start of algorithm]

1. Set order number, $l = 1$.
2. Repeat until ($l = N^p$)
   a. Calculate the profit/loss of the *l*th order, P_l, using Eq. (13).
   b. $l = l + 1$

[End of step 2 loop]

3. Calculate the total profit, P^t from all the potential orders with the current ordering sequence, using Eq. (32).
4. With respect to the decision criteria (earliness profit per order/longest due date), sort all potential orders.
5. Set order number, $l = 1$.
6. Reschedule all the orders in the current positions.

a. Repeat until ($l > N^p$)
b. Schedule/reschedule the *l*th order in the current position by MA.
c. Calculate the profit of the *l*th order, P^l, using Eq. (13).
d. $l = l + 1$

[End of step 6 loop]

7. Calculate the total profit for all potential orders, *Pn* (for the new ordering sequence) using Eq. (32).
8. If ($P^t < P^n$), accept the new ordering sequence and save P^n.
9. Else, maintain the initial sequence of potential orders and save P^t.

[End of algorithm]

Algorithm for multiple orders with process interruption In this section, the real-time (RT) strategy for multiple-order PFSPs under process interruptions is presented. Assume that *c* is the current clock time and t_d is the start time of a disruption. The RT algorithm must calculate the completion time (makespan) of the *l*th order C^l_{com} (C^l_{max}) and its tardiness, T_l or earliness E_l. It must also calculate the current clock time, t_l, when the first job of the *l*th order will start to be processed in the first machine and the profit/loss P^l earned from that order. Finally, the algorithm needs to determine the total profit/loss, *P*, earned from a production shift.

At the beginning of the production shift, suppose the 1st order arrives and the algorithm starts.

[Start of algorithm]

1. Set order number, $l = 1$, clock time $t_1 = 0$, and start time $S^l_1 = 0$

Repeat until $l = N$ (until the production shift ends)

a) Update the clock, $t_l = c$, and $S^l_l = c$
b) Considering all machine availability constraints and the previous assigned schedules, use MA to generate a schedule (makespan, Cl_{max}) for the lth order.
c) With respect to makespan, Cl_{max}:

i. Calculate completion time, $Cl_{com} = S^l_l + Cl_{max}$
ii. Calculate tardiness, T_l and earliness, E_l
iii. If $T_n \leq 0$, **Accept** the order
iv. Else **Reject** the order
v. Calculate the profit/loss of the lth order, Pl, using Eq. (13).

d. If the accepted order is waiting in the queue (potential order for multiple order)

- Set N^p to be the number of accepted orders waiting in the queue
- If N^p>1, apply the re-ordering heuristics. Then, set $N^p = 0$

[End step 2.d loop]
e. After any process interruptions at time, t_d

I. Identify the affected and unaffected jobs of an affected order currently under process
II. Reschedule these affected jobs that are under process by the MA and calculate the profit, Pl for the revised order
III. If the accepted order is waiting in the queue (potential orders for after a disruption)
IV. At time t_d, count the number of potential orders.

- Set N^p to be the number of accepted orders waiting in the queue
- $N^p = N^p + 1$ (count any other potential order)

[End step 2.e.iv loop]
v. If $N^p > 1$, apply the re-ordering heuristic. Then, set $N^p = 0$
f. Set $l = l + 1$
[End of step 2 loop]
3. Calculate and save the total profit, P, using Eq. (12).
[End of algorithm]

5 Experimental results and analysis

In this section, we present the experimental results for multiple-order PFSPs under different disruptions and analyse the performance of the proposed approach. All algorithms have been coded using C++ and were run on a Personal Computer with a 2.80-GHz Core i7 CPU, 4 GB RAM and Windows 7.

In our earlier work [18], we proposed a MA to solve static single-order PFSPs. The algorithm was tested against Taillard's Benchmark [5], and it was compared against other competitive algorithms proposed for solving single-order PFSPs. The NEH algorithm [10] is a simple and effective constructive heuristic for solving single-order PFSPs. HGA_RMA [17] and $NEGA_{VNS}$ [14] are two powerful GAs for static single-order PFSPs. PSO_{VNS} is a particle swarm optimization approach for the same problem. $NEGA_{VNS}$ is a hybrid GA for solving PFSPs. Table 3 shows the average percentage of deviations (APD) (Eq. 33) of each of these algorithms.

The average percentage of deviation is given by

$$\mathrm{APD} = \sum_{i=1}^{R} \left(\frac{BS - BKS}{BKS} \times 100 \right) / IR \tag{35}$$

Table 3 presents the comparative results for solving single-order PFSPs obtained by MA, adaptive MA ($MA_{Adaptive}$) and the state of the art algorithms mentioned above. The results tabulated in the table show the APD between the algorithms and the best known makespan. The first column shows the size of the problem (number of jobs × number of machines). The problem instances are shown in the next column, although all consist of a set of 10 different problem instances set. The final five columns show the average APD (average of 10 independent runs) for each competitive algorithm (note that only a single run of NEH [10] is listed because it is a deterministic algorithm). As shown in Table 3, in most problems, our algorithm outperforms over the other algorithms and also that MA improves the overall average (as reported in the bottom row) of APD from the known best makespan. The point of performing this computational study is that the job schedule generated by MA helps to reduce the completion time (or makespan) of an order. For this reason, the production system may require less time to produce an order. This increases the opportunity to accept more orders in real-time multiple-order PFSPs. So our proposed MA has been extended to solve practical real-time multiple-order PFSPs.

Based on our earlier study [18], the MA parameters have been selected as follows. The population size is set to 100. In the initial population, one individual is generated using the NEH (Nawaz, Enscore and Ham) algorithm [10], 40% of individuals are produced from the modified NEH algorithm, 10% from the modified Johnson algorithm [6] and the remaining individuals are generated randomly. The tournament pool size is set to 5. The probability of crossover is 90%, and the probability of mutation is 60%. With respect to traditional GAs, the crossover rate and mutation rate are high, but in tightly constrained combinatorial problems, these rates help to maintain population diversity [55, 66]. A restart mechanism has been used, if the best fitness does not change in 10 successive generations. The local search techniques are applied to 30 individuals in every generation. The algorithm terminates after 150 generations. The algorithm has been independently run 10 times and the results have been analysed. The

computational times of the proposed algorithms have been presented in Appendix 2.

For $MA_{Adaptive}$, the lowest value of crossover probability is set to $\partial^c \min = 60\%$ and the highest value of crossover rate is set to $\partial^c \max = 90\%$. For mutation, the lowest value is set to $\partial^m \min = 5\%$ and the highest value is set to $\partial^m \max = 50\%$. The values of other parameters are the same as the MA's parameters.

The problem under study considers two dynamic issues in PFSPs: (i) multiple orders arrive on a continuous timeline and (ii) unknown process disruption. In the literature, there are no benchmark problems on multiple-order PFSPs under disruptions that can be used to test the performance of the proposed algorithms. For this purpose, test problems were generated randomly in the following manner: a set of jobs was selected with processing times to form an order, and an arrival time and a due date of each order was generated. For this purpose, we selected a group of eight static problem instances (Ta011, Ta012, Ta041, Ta042, Ta071, Ta072, Ta091, and Ta092) from Taillard's Benchmark [5]. These instances are 10-machine problems, which are a representative number of many practical flow shops. In this research, the arrival times were generated by a Poisson distribution and the due date of each order was generated following exponential distributions, as suggested in [51, 53]. We have compared the solutions with the right shifting method, as well as with the upper bound (UB) and lower bound (LB) solutions derived in this paper. Here, we assume that if there is no order to process, then the production shop may be left idle. In reordering (/re-sequencing) a set of orders, our objective is to maximize the profit obtained from the orders. To find the overall profit, all revenue and cost figures were generated randomly using a uniform distribution where the sales revenue was within the range of (1000, 4000), the tardiness cost and earliness benefit per order was within (1, 10) and the opportunity lost cost was within (100, 600). All these cost/revenue units are in $ (dollar). For simplicity, we have considered that the tardiness cost was the same as the earliness benefit per order. Sample revenue and cost figures are provided in Table 10 (in Appendix 3).

To test the performance of the proposed approach, three different production scenarios, based on the eight test problems discussed earlier, have been considered as follows:

- Scenario 1: the orders are the same in their characteristics, for example, dealing with any one type of problem.
- Scenario 2: different orders, but every order has the same fixed number of jobs.
- Scenario 3: different orders with any number of jobs.

To demonstrate the performance of the proposed algorithm, we have experimented with 10 orders in a production shift for scenario 1, and 25 orders for scenarios 2 and 3.

Table 4 presents a sample comparison between the RS and RT strategies for scenario 1, where each order contains 20 jobs that must be processed on 10 machines (problem instance Ta011). The first column represents the order number. The second column represents the single-order makespan or completion time (static) for each order. Note that after the machine breakdown, the affected jobs are right shifted and the single-order makespan for the 1st order was changed to 2851. The following two columns show randomly generated arrival times and due dates for the orders. The completion time is the time to complete each order with the current production capacity. The tardiness shows each order's earliness or tardiness. If the tardiness value for an order is negative (or positive earliness value), the order is accepted as it can be completed before the due date. In addition, a zero value of tardiness (or earliness) means the order can be finished and delivered on its due date (just in time).

However, a positive value of tardiness (negative value of earliness) means a delay in completing an order and so the order is rejected. The profit or loss gained from each order is also shown, where a positive value means profit and a negative value means loss. In the status column, 'A' means accepted and 'R' means rejected. If the tardiness value for an order is negative, the order is accepted, and a positive value of tardiness means a delay in completing a job. For the convenience of readers, all of the related cost figures used for the current scenario is provided in Table 9 (in Appendix 3). Table 4 shows another scenario, in it processing starts immediately after the arrival of the 1st order as the flow shop is idle. For the 1st order, the completion time is equal to the makespan of the static PFSP version of this problem instance for both the RS and RT strategies. The following order arrives when the 1st order is still in process on machine-1. The total time to complete all jobs from the 1st order on machine-1 is 1081 and the 2nd order arrives at time = 50. So there is overlap between these two orders. So the first job of the 2nd order starts to process on machine-1 at $t = 1081$. If the 2nd order is scheduled with the RS strategy, it completes at time = 2961, which takes 2806 time units with the RT strategy. The 3rd, 4th and 5th orders arrive at a time when the 1st order is still in process on machine-1. So these three orders must also wait until machine-1 is free for them. Considering the available system capacity, the acceptance/rejection decision, using both the RS and RT strategies, suggests to accept these orders (3rd, 4th and 5th). As just explained, the first five rows in Table 4 present the expected completion times and the profit/loss of five orders with no interruption. Now suppose that machine-1 breaks at $t = 127$ while processing the 1st order and the expected duration of disruption is 1252 time units (Table 4).

Meanwhile, four other accepted orders (2nd to 5th) are waiting in the queue. According to our definition, the 1st order is now a WIPA order and the remaining four orders are potential orders. For ease of explanation, we copy the content of all

cells of the first four columns from rows 1–5 to rows 6–10. After the machine breakdown, according to the RS strategy, some of the jobs of the 1st order, and all of the other waiting orders, must be shifted to the right of the schedule. If no other interruption takes place, the total profit for these five orders is $17,539 (sum of the profits from rows 6–10). However, with the RT strategy, the affected jobs from the 1st order must be rescheduled while considering the changed system capacity and the required due dates. The re-ordering of the potential orders will be allowed if it improves the performance of the system and if that occurs, the jobs of these orders will also then be rescheduled. In the re-ordering process, we consider two rules: (i) earliness bonus per unit time per order and (ii) longest due date. Based on the first rule, the suggested sequence of potential orders is 4-5-3-2 (reflected in the completion time column), which provides a total profit of $27,203.8. Note that the total profit is higher than the same of the initial sequence of orders (as shown in the first five rows). On the other hand, based on the second rule, the sequence of waiting orders is 4-3-5-2 with a profit of $27,675.

With both the RS and RT strategies, the 6th to 8th orders are rejected as they are not feasible to schedule, and so the opportunity loss costs have been deducted from the objective value. Finally, with the currently available capacity, the 9th and 10th orders are feasible to accept with both strategies. From the table, it is clear that for this production shift the total profit earned by RS is $32,740.9 (sum of profit/loss from all orders shown in rows 6–15). In contrast, the total profit by RT with rule-1 is $37,191.6 and with rule-2 is $37,569.8. This means that the RT strategy is superior to the RS strategy.

Table 5 shows a summary of the comparisons between the RS and RT (with rule-2) strategies under different machine breakdown scenarios, where the same types of orders randomly arrive at a shop. For each problem size, under the same type of disruption (here machine breakdown), we have considered two problem instances. Preliminary experiments show that RT based on rule-2 is better than RT based on rule-1. So the results of RT based on rule-2 have been compared with the RS approach. The table starts with a column that represents the number of jobs in each order. The following columns, headed with U_D, presents the percentage deviations from the worst-case profits (i.e. UB) for RS and RT. The deviation from the LB is shown in the next two columns. The profit values are expected to be between UB and LB, and a larger deviation from UB and a small deviation from LB usually provide confidence that reasonable solutions have been obtained. Note that UB and LB are calculated based on accepted orders per production shift. If the number of accepted orders for RS and RT are different, the corresponding UB and LB will also be different. In all cases, RT outperforms the RS strategy. The next two columns show the number of accepted orders by the RS or RT strategies. Based on one random machine breakdown in every 10 orders, RT produces compact schedules that allow it to accept either equal or more orders than the RS strategy**.** RS usually takes longer to complete the orders under a disruption situation and that reduces the possibility of accepting more orders than the RT strategy. The final column represents the total profit earned from the production shifts; it clearly shows that RT provides higher profit than RS under machine breakdown scenarios.

Tables 6 and 7 present the simulation results for the RS and RT strategies for scenarios 2 and 3 respectively. The representations of these tables are the same as Table 5 that was described above. For each problem size, Table 6 illustrates the comparisons between RS and RT with respect to single problem instances. On the other hand, Table 7 shows the simulation results for two independent problem instances under the same types of disruption (machine breakdown).

Both of these tables also demonstrate the superior performance of RT over RS. The detailed results for each disruption scenario are presented in Tables 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, and 24 (in Appendix 4).

6 Conclusion

In this paper, we have considered a practical PFSP with real-time multiple-order acceptance, combined with a scheduling problem, under different real-time process disruptions. In this problem, multiple decision strategies need to be considered, which involve, whether a new order is accepted or not, and if accepted, how the jobs of that order are scheduled, and whether there is any advantage from rearranging the orders waiting in the queue. Also, once the process is interrupted, what decision strategies should be taken to compensate for its effects? In real-life situations in manufacturing environments, order arrival and process interruptions are dynamic. So these decision strategies have to be taken on a real-time basis.

This research differs from other works, since the majority of the research published to-date focuses on the static order acceptance problem, and no work focuses on simultaneously considering dynamic multiple-order acceptance and process interruptions. The key advantage of the present work over existing studies include the following: higher financial return, better production capacity utilization, higher customer satisfaction and an effective approach that is suitable for practical decision making in a real-time manner. To solve this problem, we have proposed an integrated approach that includes two heuristics and a memetic algorithm. To evaluate the performance of the proposed approach, we have generated a set of benchmark instances. Numerical experiments on these instances demonstrate that the proposed real-time approach enhances the performance of flow shop businesses by maximizing the overall profit gain from production and improves the satisfaction level of customers, as compared to the traditional right shifting technique.

The research carried out in this paper opens up the opportunities to do interesting research in future on several problems of interest. First, this study mainly considers that one machine is assigned at each production stage and so considering multiple machines at each stage (i.e. hybrid permutation flow shop) can be considered in the future. This research also considers negligible set-up times for each job, which can be extended to study sequence dependent setup times. Secondly, research may be carried out on how to extend this method of solving real-time scheduling problems to solve other shop floor environments. Third, the proposed algorithms can be extended to study integrated decision making processes with out-sourcing or capacity expansion decisions. Fourth, besides considering in-house production floor scheduling decisions, the proposed decision process can be linked with supply chains, where order acceptance and production scheduling decisions are integrated with supply chain decisions. Fifth, the algorithms proposed can be revised to consider the concept of flow shops with limited buffer or bottleneck conditions. Finally, the proposed approach can be extended to study other PFSPs variants, like flexible flow shop, limited buffer, blocking or re-entrant flow shop scheduling problems. In the future, we also intend to focus on such problems.

Acknowledgments This research is partially supported by ARC DP170102416 awarded to R. Sarker and D. Essam.

Appendix 1
Job processing flow diagram for multiple-order PFSP with process interruptions

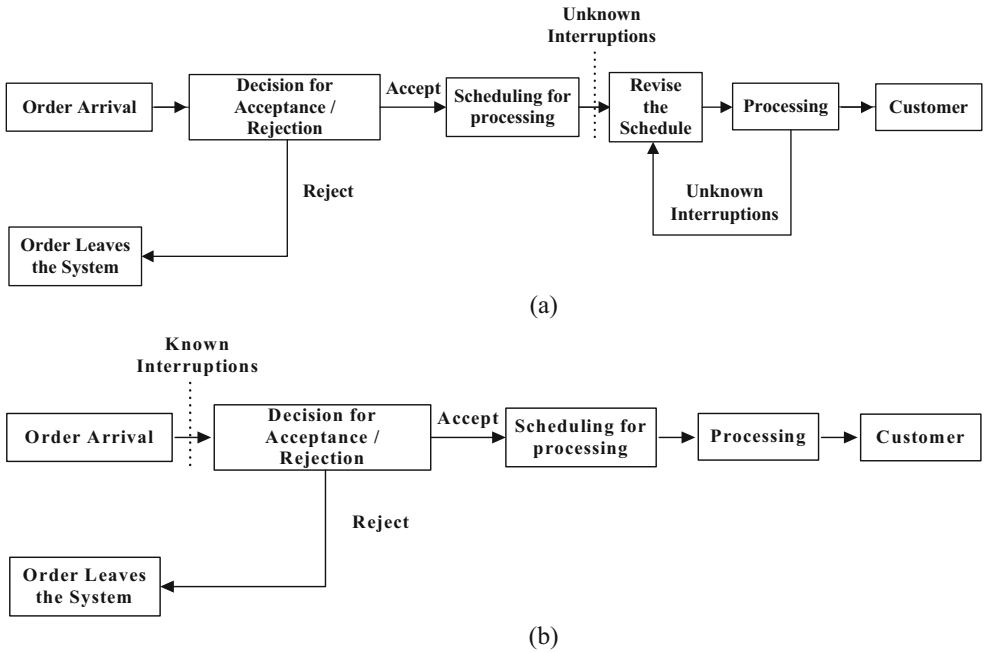

Figure 5 represents the production flow diagram for multiple-order PFSPs with different process interruptions. The production system first receives the orders from a customer. Depending on the feasibility and profitability of manufacturing that order, the management takes the decision to accept or reject the new order. If that order is accepted, it is scheduled with respect to the current orders in the system, available shop capacity, production disruptions, and the due dates of all of the accepted jobs waiting for processing. As a result, some of the orders waiting in the queue may be rescheduled. If the production capacity is available, an order may be processed instantly after being accepted. Alternatively, the accepted order waits in the queue. While the accepted orders are being processed by the machines, known (in Fig. 1a) or unknown (in Fig. 1b) disruptions may halt the production. This disturbance may in turn delay the processing of one or more orders. To compensate for the effect of this disturbance, the schedule of affected orders needs to be revised. After the production, each job order is delivered to its customer. Otherwise, if an order is rejected, then it leaves the system.

Appendix 2

The average computational time for the proposed MA is shown in Appendix Table 8. A comparison of different techniques, with respect to computational time, is rather difficult as CPU time in computers is not the one and only indicator of computational power (for example programming skills, language used for coding, CPU architectures with dual core processors) [14]. To make a fair comparison, we convert the CPU time of our proposed algorithm using Eq. (36), as found in [67]

$$\text{Scaled CPU time(s)} = \frac{2.8\ \text{GHz}}{\text{Given CPU speed (GHz)}} \times \text{given CPU time (s)} \quad (36)$$

Recall that the proposed MA algorithm was coded in the C++ language and was run with an Intel Core i7, 2.80 GHz with 4 GB RAM. In this section, the computational time has been scaled to a Pentium IV platform. The average running time for two of the competitive algorithms has been reported in Appendix Table 8 [14]; they both used a Pentium IV at 2.4 GHz. Ruiz et al. [17] did not report the average computational time for each problem instance [14].

Appendix 3

Table 8 Time requirements by each algorithm

Problem	Problem instance group	Average computational time in Pentium IV platform (s)			
		PSO_{VNS}	$NEGA_{VNS}$	MA	$MA_{Adaptive}$
20 × 5	Ta001–010	15.7	2.2	2	2
20 × 10	Ta011–020	30.7	12.2	12	12
20 × 20	Ta021–030	80.8	29.2	30	30
50 × 5	Ta031–040	3.3	8.2	6	6
50 × 10	Ta041–050	93.1	32.3	40	40
50 × 20	Ta051–060	196.1	55.0	65	65
100 × 5	Ta061–070	61.4	30.8	40	40
100 × 10	Ta071–080	246.2	58.7	70	70
100 × 20	Ta081–090	362.6	122.7	142	142
200 × 10	Ta091–100	223.2	134.5	190	190
200 × 20	Ta101–110	511.8	271.7	380	380
500 × 20	Ta111–120	–	523.4	750	750

Appendix 4

Table 9 All relevant costs and sales revenue

Order no.	Sales revenue ($)	Earliness bonus per order ($)	Tardiness cost per order ($)	Opportunity loss cost ($)
1	3879	1.7	1.7	534
2	3808	1.1	1.1	542
3	1888	2.0	2.0	161
4	3908	1.7	1.7	370
5	1796	1.6	1.6	391
6	1016	1.4	1.4	316
7	3884	1.1	1.1	110
8	2270	1.8	1.8	185
9	1491	1.7	1.7	404
10	1283	1.4	1.4	548

Table 10 Same order arrival, preventive maintenance

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	1.86875	2.21875	1.51375	1.4675	65	70	74,561	75,615
50	1.57	2.17	2.19375	1.78125	72	77	76,512	78,406
100	1.40375	1.49125	2.25875	1.90375	80	84	67,812	70,471
200	1.3075	1.6875	1.7825	2.01375	71	71	78,114	80,490

Table 11 Mixed order, same order size, preventive maintenance

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	2.17125	1.7325	1.305833	0.850833	14	15	70,216	71,311
50	2.49375	1.47375	1.123333	0.9575	11	15	66,109	69,914
100	1.905	1.91875	1.349167	1.1225	14	12	74,242	78,843
200	1.3175	1.5475	1.594167	1.581667	11	14	77,020	83,069

Table 12 Mix order, different order sizes, preventive maintenance

Problem instances	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
1	0.23	0.14625	0.19625	0.2225	19	21	180,371	182,274
2	0.23	0.17	0.235	0.24625	19	19	185,140	186,783

Table 13 Same order arrival, rework

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	1.7475	1.68375	2.4625	2.42625	60	70	69,184	73,521
50	1.79625	1.8125	1.28375	1.28125	74	77	78,360	83,354
100	1.41125	1.61	1.25375	2.34375	70	84	71,863	82,948
200	2.21125	1.7175	2.3425	2.21625	66	71	76,072	77,548

Table 14 Mixed order, same order size, rework

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	2.03875	1.28625	0.934167	1.04	12	16	80,373	77,675
50	1.89375	1.3525	1.598333	1.471667	16	13	74,688	65,463
100	1.9525	1.69125	1.4025	1.03	11	12	78,587	65,894
200	1.65625	1.94875	0.984167	0.920833	15	11	72,271	77,744

Table 15 Mix order, different order sizes, rework

Problem instances	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
1	0.22625	0.245	0.1475	0.14	19	17	185,438	179,228
2	0.18375	0.205	0.15125	0.17375	17	18	181,131	181,356

Table 16 Same order arrival, job Cancellationsc

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	1.43125	1.8375	2.14875	2.495	69	70	70,784	80,136
50	2.205	1.73875	2.08375	1.945	62	77	67,235	82,451
100	2.25625	1.685	2.3525	1.84875	62	84	84,002	77,162
200	1.83875	1.9175	2.03125	2.26375	75	71	83,740	75,798

Table 17 Mixed order, same order size, job Cancellationsc

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	2.14375	1.96625	1.385833	1.4125	16	13	71,427	69,282
50	1.5225	1.9775	0.838333	1.131667	14	14	74,767	75,420
100	1.8825	1.54	1.299167	0.9025	12	12	66,671	68,989
200	2.00875	2.0975	1.085	1.545833	14	16	67,267	77,797

Table 18 Mix order, different order sizes, job Cancellationsc

Problem instances	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
1	0.22625	0.245	0.1475	0.14	19	17	185,438	179,228
2	0.18375	0.205	0.15125	0.17375	17	18	181,131	182,356

Table 19 Same order arrival, change in processing times

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	2.05	1.73875	1.415	2.2425	61	70	71,175	65,062
50	2.29375	1.8925	1.72375	2.1825	68	77	69,979	65,778
100	1.275	1.55125	1.98875	2.14625	69	84	76,667	81,068
200	1.80375	2.045	1.33125	1.98625	63	71	69,880	78,243

Table 20 Mixed order, same order size, change in processing times

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	2.14375	1.96625	1.385833	1.4125	16	17	71,427	74,282
50	1.5225	1.9775	0.838333	1.131667	14	14	76,767	81,420
100	1.8825	1.54	1.299167	0.9025	15	16	66,671	68,989
200	2.00875	2.0975	1.085	1.545833	14	14	67,267	77,797

Table 21 Mix order, different order sizes, change in processing times

Problem instances	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
1	0.185	0.205	0.20375	0.18	18	15	174,329	175,310
2	0.18875	0.1825	0.22125	0.15	17	18	175,558	176,566

Table 22 Same order arrival, change in due dates

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	1.86125	2.1775	2.47375	2.49625	64	70	80,398	69,103
50	2.3125	2.405	1.89125	1.8	70	77	68,285	83,790
100	1.975	2.28875	2.325	1.855	70	84	66,140	75,795
200	1.40125	1.6575	2.31875	1.86625	62	71	77,637	82,419

Table 23 Mixed order, same order size, change in due dates

Problem size (jobs)	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
20	1.52125	1.57875	1.349167	1.526667	16	12	71,154	72,184
50	2.22125	1.99875	0.8425	1.315833	15	15	73,128	73,685
100	1.675	2.4775	1.665833	0.996667	16	11	68,614	80,841
200	2.015	1.42375	1.130833	0.959167	15	12	72,376	76,980

Table 24 Mix order, different order sizes, change in due dates

Problem instances	U_D		L_D		Accepted number of orders		Total profits ($)	
	RS	RT	RS	RT	RS	RT	RS	RT
1	0.23125	0.24375	0.2325	0.13625	20	18	177,023	178,679
2	0.1375	0.19375	0.18875	0.17125	20	20	172,830	175,235

References

1. Abumaizar RJ, Svestka JA (1997) Rescheduling job shops under random disruptions. Int J Prod Res 35:2065–2082
2. Pinedo M (2012) Scheduling: theory, algorithms, and systems, 4th edn. Springer, New York
3. Aggoune R, Portmann M-C (2006) Flow shop scheduling problem with limited machine availability: a heuristic approach. Int J Prod Econ 99:4–15
4. Schmidt G (2000) Scheduling with limited machine availability. Eur J Oper Res 121:1–15
5. Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64:278–285
6. Johnson SM (1954) Optimal two- and three-stage production schedules with setup times included. Nav Res Logist Q 1:61–68
7. Garey MR, Johnson DS, Sethi R (1976) The complexity of flowshop and jobshop scheduling. Math Oper Res 1:117–129
8. Bansal S (1977) Minimizing the sum of completion times of n jobs over m machines in a flowshop—a branch and bound approach. AIIE T 9:306–311
9. Selen WJ, Hott DD (1986) A mixed-integer goal-programming formulation of the standard flow-shop scheduling problem. J Oper Res Soc 37:1121–1128
10. Nawaz M, Enscore EE, Ham I (1983) A heuristic algorithm for the M-machine, N-job flowshop sequencing problem. OMEGA Int J Manag Sci 11:91–95
11. Grabowski J, Wodecki M (2004) A very fast tabu search algorithm for the permutation flow shop problem with makespan criterion. Comput Oper Res 31:1891–1909
12. Tasgetiren MF, Liang YC, Sevkli M, Gencyilmaz G (2007) A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. Eur J Oper Res 177:1930–1947
13. Osman IH, Potts CN (1989) Simulated annealing for permutation flowshop scheduling. OMEGA Int J Manag Sci 17:551–557
14. Zobolas GI, Tarantilis CD, Ioannou G (2009) Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm. Comput Oper Res 36:1249–1267
15. Rajendran C, Ziegler H (2004) Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. Eur J Oper Res 155:426–438
16. Ruiz R, Maroto C (2005) A comprehensive review and evaluation of permutation flowshop heuristics. Eur J Oper Res 165:479–494
17. Ruiz R, Maroto C, Alcaraz J (2006) Two new robust genetic algorithms for the flowshop scheduling problem. OMEGA Int J Manag Sci 34:461–476
18. Rahman HF, Sarker RA, Essam DL (2013) A memetic algorithm for permutation flow shop problems, in 2013 IEEE Congress on Evolutionary Computation (CEC), pp. 1618–1625
19. Dasgupta P, Das S (2015) A discrete inter-species cuckoo search for flowshop scheduling problems. Comput Oper Res 60:111–120
20. Slotnick SA (2011) Order acceptance and scheduling: a taxonomy and review. Eur J Oper Res 212:1–11
21. Slotnick SA, Morton TE (1996) Selecting jobs for a heavily loaded shop with lateness penalties. Comput Oper Res 23:131–140
22. Lewis HF, Slotnick SA (2002) Multi-period job selection: planning work loads to maximize profit. Comput Oper Res 29:1081–1098
23. Slotnick SA, Morton TE (2007) Order acceptance with weighted tardiness. Comput Oper Res 34:3029–3042
24. Rom WO, Slotnick SA (2009) Order acceptance using genetic algorithms. Comput Oper Res 36:1758–1767
25. Wester F, Wijngaard J, ZIJM WRM (1992) Order acceptance strategies in a production-to-order environment with setup times and due-dates. Int J Prod Res 30:1313–1326
26. Duenyas I, Hopp WJ (1995) Quoting customer lead times. Manag Sci 41:43–57
27. Duenyas I (1995) Single facility due date setting with multiple customer classes. Manag Sci 41:608–619
28. Nandi A, Rogers P (2004) Using simulation to make order acceptance/rejection decisions. SIMULATION 80:131–142
29. Rogers P, Nandi A (2007) Judicious order acceptance and order release in make-to-order manufacturing systems. Prod Plan Control 18:610–625
30. Moreira MRA, Alves RAFS (2009) A methodology for planning and controlling workload in a job-shop: a four-way decision-making problem. Int J Prod Res 47:2805–2821
31. Ouelhadj D, Petrovic S (2009) A survey of dynamic scheduling in manufacturing systems. J Sched 12:417–431
32. Stoop PPM, Wiers VCS (1996) The complexity of scheduling in practice. Int J Oper Prod Manag 16:37–53
33. Suresh V, Chaudhuri D (1993) Dynamic scheduling-a survey of research. Int J Prod Econ 32:53–63
34. Vieira GE, Herrmann JW, Lin E (2003) Rescheduling manufacturing systems: a framework of strategies, policies, and methods. J Sched 6:39–62
35. Tang LX, Zhao Y, Liu JY (2014) An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production. IEEE Trans Evol Comput 18:209–225
36. McKay KN, Buzacott JA, Safayeni FR (1989) The scheduler's knowledge of uncertainty: the missing link, Knowledge Based Production Management Systems, pp. 171–189
37. Adiri I, Frostig E, Kan AHGR (1991) Scheduling on a single machine with a single breakdown to minimize stochastically the number of tardy jobs. Nav Res Logist 38:261–271
38. Liao CJ, Chen WJ (2004) Scheduling under machine breakdown in a continuous process industry. Comput Oper Res 31:415–428
39. Hall NG, Potts CN (2004) Rescheduling for new orders. Oper Res 52:440–453
40. Yang B (2007) Single machine rescheduling with new jobs arrivals and processing time compression. Int J Adv Manuf Technol 34: 378–384
41. Wu SD, Storer RH, Pei-Chann C (1993) One-machine rescheduling heuristics with efficiency and stability as criteria. Comput Oper Res 20:1–14
42. Tamer Unal A, Uzsoy R, Kiran AS (1997) Rescheduling on a single machine with part-type dependent setup times and deadlines. Ann Oper Res 70:93–113
43. Qi X, Bard JF, Yu G (2006) Disruption management for machine scheduling: the case of SPT schedules. Int J Prod Econ 103:166–184
44. Allahverdi A (1996) Two-machine proportionate flowshop scheduling with breakdowns to minimize maximum lateness. Comput Oper Res 23:909–916
45. Rahman HF, Sarker RA, Essam DL, Guijuan C (2014) A memetic algorithm for solving permutation flow shop problems with known and unknown machine breakdowns, in 2014 IEEE Congress on Evolutionary Computation (CEC), pp. 42–49
46. Katragjini K, Vallada E, Ruiz R (2013) Flow shop rescheduling under different types of disruption. Int J Prod Res 51:780–797
47. Ruiz R, Garcia-Diaz JC, Maroto C (2007) Considering scheduling and preventive maintenance in the flowshop sequencing problem. Comput Oper Res 34:3314–3330

48. Perez-Gonzalez P, Framinan JM (2009) Scheduling permutation flowshops with initial availability constraint: analysis of solutions and constructive heuristics. Comput Oper Res 36:2866–2876
49. González PP, Torres JMF, Parente JMM, Pineda A (2009) Comparison of heuristics for rescheduling in permutation flowshops. In XIII Congreso de Ingeniería de Organización, pp. 1498–1506
50. Fahmy SA, Balakrishnan S, ElMekkawy TY (2009) A generic deadlock-free reactive scheduling approach. Int J Prod Res 47: 5657–5676
51. Hasan SMK, Sarker R, Essam D (2011) Genetic algorithm for job-shop scheduling with machine unavailability and breakdowns. Int J Prod Res 49:4999–5015
52. Sarker R, Essam D, Hasan S, Karim A (2015) Managing risk in production scheduling under uncertain disruption, Artificial Intelligence Engineering Design, Analysis Manufacturing, pp. 1–11
53. Sarker R, Omar M, Hasan SMK, Essam D (2013) Hybrid evolutionary algorithm for job scheduling under machine maintenance. Appl Soft Comput 13:1440–1447
54. Al-Hinai N, ElMekkawy TY (2011) Robust and stable flexible job shop scheduling with random machine breakdowns using a hybrid genetic algorithm. Int J Prod Econ 132:279–291
55. Hasan SK (2009) Evolutionary algorithms for solving job-shop scheduling problems in the presence of process interruptions, University of New South Wales
56. Wagner HM (1959) An integer linear-programming model for machine scheduling. Nav Res Logist Q 6:131–140
57. Ong Y-S, Lim M-H, Neri F, Ishibuchi H (2009) Special issue on emerging trends in soft computing: memetic algorithms. Soft Comput—A Fusion Foundations, Methodologies Applications 13: 739–740
58. Akkan C (1997) Finite-capacity scheduling-based planning for revenue-based capacity management. Eur J Oper Res 100:170–179
59. Burke EK, Smith A (1999) A memetic algorithm to schedule planned maintenance for the national grid. Journal of Experimental Algorithmics (JEA) 4:1–13
60. Merz P, Freisleben B (2001) Memetic algorithms for the traveling salesman problem. Complex Syst 13:297–346
61. Tang J, Lim MH, Ong YS, Er MJ (2005) Solving large scale combinatorial optimization using PMA-SLS. In Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, pp. 621–628
62. Goldberg DE, Holland JH (1988) Genetic algorithms and machine learning. Mach Learn 3:95–99
63. Murata T, Ishibuchi H, Tanaka H (1996) Genetic algorithms for flowshop scheduling problems. Comput Ind Eng 30:1061–1071
64. Taillard E (1990) Some efficient heuristic methods for the flow-shop sequencing problem. Eur J Oper Res 47:65–74
65. Zhu KQ, Liu Z (2004) Empirical study of population diversity in permutation-based genetic algorithm. In Genetic and Evolutionary Computation–GECCO 2004, pp. 420–421
66. Hasan SK, Sarker R, Essam D, Cornforth D (2009) Memetic algorithms for solving job-shop scheduling problems. Memet Comput 1:69–83
67. Yuan X, Wang L, Yuan Y, Zhang Y, Cao B, Yang B (2008) A modified differential evolution approach for dynamic economic dispatch with valve-point effects. Energy Convers Manag 49: 3447–3453