Evolutionary Multi-objective Blocking Lot-streaming Flow Shop Scheduling with Machine Breakdowns

Yuyan Han, Dunwei Gong, *Member IEEE*, Yaochu Jin, *Fellow IEEE*, and Quanke Pan

***Abstract*—In various flow shop scheduling problems, it is very common that a machine suffers from breakdowns. Under these situations, a robust and stable sub-optimal scheduling solution is of much more practical interest than a global optimal solution that is sensitive to environmental changes. However, blocking lot-streaming flow shop scheduling problems with machine breakdowns have not yet been well studied up to date. This paper presents, for the first time, a multi-objective formulation of the above problem including robustness and stability criteria. Based on this formulation, an evolutionary multi-objective robust scheduling algorithm (REMO, for short) is suggested, in which solutions obtained by a variant of single-objective heuristic algorithm are incorporated in population initialization and two novel crossover operators are proposed to take advantage of non-dominated solutions. In addition, a rescheduling strategy based on the local search is introduced to further reduce the influence resulting from machine breakdowns.The proposed algorithm is applied to 22 test sets, and compared with the state-of-the-art algorithms without machine breakdowns. Our empirical results demonstrate that the proposed algorithm can effectively tackle blocking lot-streaming flow shop scheduling problems in the presence of machine breakdowns by obtaining scheduling strategies that are robust and stable.**

***Index Terms*—Machine breakdown, scheduling, robustness and stability criteria, genetic algorithms, rescheduling strategy.**

I. Introduction

VARIOUS real-world problems can be formulated as flow shop scheduling problems, among which lot-streaming flow shop (LSFS) problems are very typical ones. In these problems, a job is split into several sublots, with each being transferred to the downstream machine after it is completed on the current one [1]. Dependent on whether an intermediate buffer exists or not, LSFS scheduling problems can be classified into two categories: one with infinite buffers and the other with finite buffers. The former does not cause job blocking since it has enough intermediate buffers to store those completed jobs, whereas the latter only maintains a limited capacity of in-process inventories. LSFS scheduling problems with no intermediate buffers can be seen as a special case of the latter. In the literature, flow shop scheduling problems with no intermediate buffers are known as blocking flow shop (BFS) scheduling problems [2]. Correspondingly, LSFS scheduling problems with no intermediate buffers are called blocking LSFS problems (BLSFS), where a sublot is kept blocked on the current machine until the downstream one is available.

BLSFS scheduling problems are commonly seen in real-world applications, such as in poultry industry [3], serial manufacturing processes [4], and iron and steel industry [5]. However, they are very difficult to solve effectively due to the large number of constraints and high complexity. Apart from the above-mentioned difficulties, real-world BLSFS scheduling problems often suffer from various disruptions and unforeseen events. Generally, uncertainties in BLSFS scheduling problems can be classified into two categories [6]: resource-related and job-related factors. The former mainly refers to machine breakdowns and material shortage, whereas the latter includes the arrival of new jobs, and changes in process time. Although various efforts have been made on solving LSFS scheduling problems with single or multiple objectives, most of them do not take uncertainty into account. It is, however, extremely important to guarantee that an optimal schedule is relatively insensitive to unforeseen machine breakdowns. Therefore, it is high time that efforts be dedicated to the multi-objective BLSFS scheduling problem with machine breakdowns. This study has the following twofold novelties.

In case that the considered scheduling problem is subject to uncertainties, robustness and stability are two common objectives to reduce the effects of the uncertainties. This study formulates, for the first time, a multi-objective BLSFS scheduling problem that takes robustness and stability to such uncertainties as machine breakdowns into account. The formulation of the above scheduling problem can better reflect real-world applications, and thus, is of more practical significance, compared to those in previous work.

Following that, an evolutionary multi-objective robust scheduling algorithm is then proposed to solve the formulated multi-objective scheduling problem. The proposed evolutionary algorithm has the following trifold features. The first is that the population is initialized with solutions obtained by a variant of a single-objective heuristic, vNEH. The second is that two novel crossover operators are designed to take advantage of non-dominated solutions. Third, a rescheduling

Y. Han is with School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou 221116, China, and School of Computer Science, Liaocheng University, China. e-mail: yuyanhan1023@163.com

D. Gong is with School of Information Science and Technology, Qingdao University of Science and Technology, Qingdao, Shandong, 266061, China, and School of Information and Electrical Engineering, China University of Mining and Technology, Xuzhou 221116, e-mail: dwgong@vip.163.com

Y. Jin is with School of Management Science and Engineering, Dalian University of Technology, Dalian, China; 116023 and the Department of Computer Science, University of Surrey, Guildford, Surrey, GU2 7XH, United Kingdom. e-mail: yaochu.jin@surrey.ac.uk

Q. Pan is with State Key Laboratory of Digital Manufacturing Equipment and Technology, Huazhong University of Science &Technology, Wuhan, China. e-mail: panquanke@gmail.com

strategy based on the local search is presented to further reduce the negative influence resulted from machine breakdowns. The performances of the proposed initialization, crossover, and rescheduling strategies are empirically evaluated, respectively. The experimental results demonstrate that the proposed strategies can effectively tackle the multi-objective BLSFS scheduling problem in the presence of machine breakdowns by obtaining robust and stable scheduling strategies.

The rest of this paper is organized as follows. Section II provides a brief overview of related works. Section III formulates the BLSFS scheduling problem with the makespan and the tardiness time being the two objectives, and its non-deterministic version including the robustness and stability criteria. The proposed optimization algorithm is described in Section IV. Empirical results are presented and discussed in Section V. Finally, Section VI concludes the paper.

II. Literature review

Generally, methods for solving a flow shop scheduling problems in the presence of uncertainties can be divided into the two categories. The first category converts the problem with uncertainties into a deterministic one by constructing a robust mathematical model or surrogate measures using discrete or continuous scenario sets with intervals with known and specific terminals, and then solves it using existing algorithms [7]. The second category modifies an existing schedule strategy in response to the uncertainties, i.e. the rescheduling methods[8].

In the real-world, the processing time of a job may be highly uncertain. Up to date, there are three types of methods to tackle the uncertainty of processing time in the flow shop scheduling. The first is to model the uncertain processing time using a gamma distribution, with gamma being the expected processing time[9]. The second is to describe the uncertain processing time using an interval number, with the actual processing time being any value in the range of the interval[10]. The third method introduces a fuzzy number to capture the uncertainty in processing time. Accordingly, the objectives of makespan and (or) the total weighted completion time are (is) also a fuzzy number(s)[11].

In addition uncertainty in processing time, machine breakdown has also been considered in the single-machine scheduling problems. Both scenario-based robustness and slack-based surrogate measures are also employed to represent the uncertainties of an optimization problem. Liu et al. [12] proposed a surrogate measure based on the robustness and stability objectives, and adopted a two-stage multi-population genetic algorithm to optimize the above criteria. Their computational results show that the proposed method successfully reduced the sensitivity of the obtained schedule strategy to machine breakdown. Chaari et al. [13] and Goren et al. [7] developed a scenario-based and a slack-based methods, respectively, to measure robustness of a scheduling strategy. However, the robustness measures focus only on the structure of a scheduling strategy, and neglect the uncertainties involved in a scheduling problem. To overcome the above drawbacks, Xiong et al. [14] developed two new surrogate measures by taking the location of the float time and machine breakdown as well as the probability of the machine breakdown into account.

Rescheduling strategies have also been adopted to deal with uncertainties in the flow shop scheduling, which can be further categorized into two approaches. The first approach includes either a right-shift heuristic that shifts the remaining operation schedules forwards in case of a machine breakdown and/or a partial or a complete rescheduling. In partial rescheduling, rescheduling is conducted only on the operation schedule(s) in failure, whereas, in complete rescheduling, a completely new schedule strategy will be generated. Compared with partial rescheduling, complete rescheduling can theoretically obtain a new optimal solution, although in practice, such optimal solutions are hardly achievable, while complete rescheduling requires prohibitive computation time. Moreover, complete rescheduling often results in instability, which means that there is a lack of continuity in a detailed schedule, often causing additional production cost [15].

The second approach includes dynamic, robust and predictive-reactive rescheduling methods. Rahman et al. [16] proposed a proactive-reactive method based on a two-step procedure. In the first step, a robust optimization approach is adopted to generate an initial robust solution proactively against the uncertainty in processing time. In the second step, a reactive approach is applied to yield the best modified sequence so as to deal with unexpected event(s). As flow shop scheduling problems with machine breakdown remain a challenge, Wang et al. [17] developed a decomposition-based method to decompose all machines into several clusters, using a neighboring K-means clustering without predefining the number of clusters. To reduce the possibility of frequent machine breakdowns, Liao and Chen [18] designed a heuristic to maximize makespan at the cost of increasing either the total setup or the total idle time, based on the assumption that long idle time between machines might significantly reduce the rate of machine breakdowns.

Although the above-mentioned methods have been successfully applied to solve the flow shop scheduling problems in the presence of uncertainties such as changes in processing time and machine breakdowns, no multi-objective approach to BLSFS with machine breakdowns has been reported. The multi-objective formulation of BLSFS with machine breakdown suggested in this work is closer to the real world, and thus, is of more practical significance, compared to the problem formulations found in the literature.

III. Problem formulation

The BLSFS scheduling problem with and without machine breakdown will be formally defined in this section. Related notations are given as follows.

m the total number of machines, with each being indexed by $i = 1, 2, ..., m$;

n the total number of jobs, with each being indexed by $j = 1, 2, ..., n$;

$\pi(j)$ the j-th job of sequence π ;

$w_{\pi(j)}$ the total number of sublots belonging to job $\pi(j)$;

e the e-th sublot, with each being indexed by $i = 1, 2, ..., w_{\pi(j)}$;

$p_{\pi(j),i}$ the processing time of job *j* on machine *i*;
$\pi(j), i, e$ the *e*-th sublot of the *j*-th job on the *i*-th machine;
$C_{\pi(j),i,e}$ completion time of sublot *e* belonging to job *j* on machine *i*;
$S_{\pi(j),i,e}$ the start time of sublot *e* belonging to job *j* on machine *i*;
d_j the due date of job *j*;
BT_i the random breakdown time of machine *i*;
RT_i the random repair time of machine *i*;
$f^{'}$ the makespan criterion;
$f^{''}$ the tardiness time criterion;
f_1 the robustness criterion related to makespan;
f_2 the robustness criterion related to the tardiness time;
f_3 the stability criterion.

A. Formulation of the BLSFS scheduling problem without machine breakdown

In BLSFS, blocking, in this paper, refers to the situation where no buffers exist for any adjacent machines, which is considered as an additional constraint to incorporate the process of calculating the objectives. Consider a BLSFS scheduling problem with *n* jobs and *m* machines, in which these jobs are represented by $\pi(1), \pi(2), ..., \pi(n)$, and the sequence formed by these jobs is denoted as $\pi = \{\pi(1), \pi(2), ..., \pi(n)\}$. Each job is processed on each machine in the same order and can be split into several sublots of an equal size, i.e., each sub-lot of a job has different processing times on different machines and has the same processing time on the same machines. In addition, we assume the following:

- A job can be processed on machine *i* only after all sublots of its previous job have been completed on this machine;
- At any time, each machine can process at most one sublot, and each sublot can be processed on at most one machine;
- All sublots of the same job should be continuously processed and no buffers exist for any machine;
- Both the setup time of each job and the transportation time of each sublot are included in their processing time.

For a BLSFS scheduling problem, each sublot of a job can be processed by at most one machine at the same time, although there may be a number of repeatable sublots. Additionally, the blocking constraint between adjacent sublots on a machine is considered. As a result, it is unpractical to regard all the sublots of a job as a whole, and impossible to achieve the values of the objectives using only one equation. Thus, for the BLSFS scheduling problem, each sublot must be addressed separately in the problem formulation.

According to the above assumptions, the BLSFS scheduling problem without machine breakdown can be formulated as follows.

$$\begin{cases} S_{\pi(1),1,1} = 0 \\ C_{\pi(1),1,1} = S_{\pi(1),1,1} + p_{\pi(1),1} \end{cases} \tag{1}$$

$$\begin{cases} S_{\pi(1),i,1} = C_{\pi(1),i-1,1} \\ C_{\pi(1),i,1} = S_{\pi(1),i,1} + p_{\pi(1),i} \end{cases} \quad i = 2, 3, ...m \tag{2}$$

$$\begin{cases} S_{\pi(j),1,1} = \max\{C_{\pi(j-1),1,w_{\pi(j-1)}}, S_{\pi(j-1),2,w_{\pi(j-1)}}\} \\ C_{\pi(j),1,1} = S_{\pi(j),1,1} + p_{\pi(j),1} \end{cases} \quad j = 2, 3, ..., n \tag{3}$$

$$\begin{cases} S_{\pi(j),i,1} = \max\{C_{\pi(j),i-1,1}, S_{\pi(j-1),i+1,w_{\pi(j-1)}}\} \\ C_{\pi(j),i,1} = S_{\pi(j),i,1} + p_{\pi(j),i} \end{cases} \quad i = 2, 3, ..., m-1 \quad j = 2, 3, ..., n \tag{4}$$

$$\begin{cases} S_{\pi(j),m,1} = \max\{C_{\pi(j),m-1,1}, C_{\pi(j-1),m,w_{\pi(j-1)}}\} \\ C_{\pi(j),m,1} = S_{\pi(j),m,1} + p_{\pi(j),m} \end{cases} \quad j = 1, 2, 3, ..., n \tag{5}$$

$$\begin{cases} S_{\pi(j),1,e} = \max\{C_{\pi(j),1,e-1}, S_{\pi(j),2,e-1}\} \\ C_{\pi(j),1,e} = S_{\pi(j),1,e} + p_{\pi(j),1} \end{cases} \quad e = 2, 3, ..., w_{\pi(j)} \quad j = 1, 2, 3, ..., n \tag{6}$$

$$\begin{cases} S_{\pi(j),i,e} = \max\{C_{\pi(j),i-1,e}, S_{\pi(j),i+1,e-1}\} \\ C_{\pi(j),i,e} = S_{\pi(j),i,e} + p_{\pi(j),i} \end{cases} \quad e = 2, 3, ..., w_{\pi(j)} \quad i = 2, 3, ..., m-1 \quad j = 1, 2, 3, ..., n \tag{7}$$

$$\begin{cases} S_{\pi(j),m,e} = \max\{C_{\pi(j),m-1,e}, C_{\pi(j),m,e-1}\} \\ C_{\pi(j),m,e} = S_{\pi(j),m,e} + p_{\pi(j),m} \end{cases} \quad e = 2, 3, ..., w_{\pi(j)} \quad j = 1, 2, 3, ..., n \tag{8}$$

Equation (1) is utilized to calculate the completion time of the first sublot of the first job on the first machine, which is different from Equation (2), where the start time of $\pi(1), 1, 1$ is equal to zero. The calculations for the first sublots of the following jobs in the sequence on the first machine is described in Equation (3). When calculating the start time, we take into account the completion time of the last sublot of the previous job on the current machine and the start time of the last sublot of the previous job on the downstream machine. The related time of the first sublots of the following jobs on the (m-1) machines is calculated in Equation (4), which includes the completion time of the first sublot of the current job on the previous machine and the start time of the last sublot of the previous job on the downstream machine. Equation (5) defines the completion time of the first sublot of a job on the last machine. Equations (6)-(8) calculate the start and complete time of the rest sublots of a job on different machines, which guarantees that the sublots of the same job are continuously processed. During the recursive calculations, the completion time of the jobs on a machine is calculated sequentially from the first to the last job.

Given that the start time of the first sublot of the first job on the first machine is equal to zero, i.e., $S_{\pi(1),1,1} = 0$, the two objectives of the BLSFS scheduling problem without machine breakdown can be calculated as follows: To clearly understand a BLSFS scheduling problem, an example for the decoding procedure is given below. The job sequence based representation is adopted in the proposed algorithm, which is easy to decode into a schedule. Assume that there are two

jobs and two machines, where both jobs 1 and 2 contain 2 sublots. The processing time of each sublot is: $p_{\pi(1),1} = 1$, $p_{\pi(1),2} = 2$, $p_{\pi(2),1} = 4$, $p_{\pi(2),2} = 2$.

Let $\pi = (1, 2)$ and $S_{\pi(1),1,1} = 0$. The objective value, i.e., makespan, is calculated as follows:

(a) $S_{\pi(1),1,1} = 0, C_{\pi(1),1,1} = S_{\pi(1),1,1} + p_{\pi(1),1} = 1$;

(b) $S_{\pi(1),2,1} = C_{\pi(1),1,1} = 1, C_{\pi(1),2,1} = S_{\pi(1),2,1} + p_{\pi(1),2} = 3$;

(c) $S_{\pi(1),1,2} = \max(C_{\pi(1),1,1}, S_{\pi(1),2,1}) = 1, C_{\pi(1),1,2} = S_{\pi(1),1,2} + p_{\pi(1),1} = 2$;

(d) $S_{\pi(1),2,2} = \max(C_{\pi(1),1,2}, C_{\pi(1),2,1}) = 3, C_{\pi(1),2,2} = S_{\pi(1),2,2} + p_{\pi(1),2} = 5$;

(e) $S_{\pi(2),1,1} = \max(C_{\pi(1),1,2}, S_{\pi(1),2,2}) = 3, C_{\pi(2),1,1} = S_{\pi(2),1,1} + p_{\pi(2),1} = 7$;

(f) $S_{\pi(2),2,1} = \max(C_{\pi(2),1,1}, C_{\pi(1),2,2}) = 5, C_{\pi(2),2,1} = S_{\pi(2),2,1} + p_{\pi(2),2} = 7$;

(g) $S_{\pi(2),1,2} = \max(C_{\pi(2),1,1}, S_{\pi(2),2,1}) = 7, C_{\pi(2),1,2} = S_{\pi(1),1,2} + p_{\pi(2),1} = 11$;

(h) $S_{\pi(2),2,2} = \max(C_{\pi(2),1,2}, C_{\pi(2),2,1}) = 11, C_{\pi(2),2,2} = S_{\pi(2),2,2} + p_{\pi(2),2} = 13$;

The above makespan of permutation π is $C_{\pi(2),2,2} = 13$.

$$\min f' = C_{\pi(n),m,w_{\pi(n)}} \tag{9}$$

$$\min f'' = \sum_{j=1}^{n} \max\{0, C_{\pi(j),m,w_{\pi(j)}} - d_j\} \tag{10}$$

B. Formulation of the BLSFS scheduling problem with machine breakdown

In the following, the situation in which a machine breaks down is investigated. For simplicity, the following additional assumptions are made for the BLSFS scheduling problems with machine breakdowns.

- Any machine can suffer from breakdowns during the production process, and the number of each machine's breakdown is equal to β;
- When a machine breaks down, the job being processed on the machine will be stopped;
- Each sublot that suffers from a disruption is required to be reworked after the repair is finished;
- Except for machine breakdowns, no other factors that disturb a job's processing are considered.

To characterize a machine breakdown, three aspects need to be taken into account [19], i.e., which machine breaks down, when the machine breaks down (i.e., machine breakdown time), and when the broken-down machine will be operational again (i.e., machine repair time). In real-world scheduling, it is very difficult, if not impossible, to build an exact probability model of the machine breakdown. In this study, a uniform distribution function is adopted to simulate the machine breakdown time and repair time under the assumption that the probability of machine breakdown is equal at any time.

The expected value of the machine breakdown time is generated using the following discrete uniform distribution.

$$E(BT_i) = rand()\%T_i + p_{\pi(j),i} \times i \qquad \begin{matrix} i = 1, 2, ..., m \\ j = 1, 2, ..., n \end{matrix} \tag{11}$$

$$E(T_i) = \sum_{j=1}^{n} p_{\pi(j),i} \times w_{\pi(j)} \qquad i = 1, 2, ..., m \tag{12}$$

In Eq.(11), *rand*() is a function that generates a random integer within the range of [0,MAXINT], where MAXINT is the maximum integer that can be generated by a computer. T_i represents the total processing time on the i-th machine for all jobs. $rand()\%T_i$ denotes the remainder when $rand()$ is divided by T_i. Equation (11) indicates that the machine breakdown time ranges from $p_{\pi(j),i}$ to makespan, and the probability of machine breakdown observes a uniform distribution.

The expected value of the repair time is simulated using the following discrete uniform distribution.

$$E(RT_i) = rand()\%31 + 1 \qquad i = 1, 2, ..., m \tag{13}$$

Equation (13) is employed to specify the repair time. For the scheduling problem considered in this paper, we assume that the repair operation can be completed within the processing time of a job on a machine. For the BLSFS scheduling problem, the processing time of a job is generated using Equation (13) as suggested in [20], where 31 is the upper bound of the processing time of a job.

Assuming that a job can be resumed when a broken-down machine is repaired, the expected completion time of the job is related not only to the processing time and the job sequence, but also to the machine breakdown time and the repair time. So the expected start time of this job is not $S_{\pi(j),i,e}$ as described in Equations (2-8); instead, it is the sum of the breakdown and the repair time, i.e., $E(S_{\pi(j),i,e}) = \max\{E(RT_i) + E(BT_i), C_{\pi(j),i-1,e}, S_{\pi(j),i,e-1}\}$.

Fig. 1 illustrates the process of BLSFS without (with) machine breakdown with an instance having two jobs and three machines. Suppose that $\pi(1)$ and $\pi(2)$ each contain 2 sub-lots. When there exist no machine breakdowns, the completion time is equal to $t1$, as shown in Fig.1(a). If there is one machine breakdown, e.g., machine M1, at this time, $\pi(2), 1, 1$ requires to be reworked after the repair is finished, which delays the start time of $\pi(2), 1, 2$ and increases the completion time from $t1$ to $t2$, see in Fig. 1(b). By contrast, when two machines, e.g., machine M1 and M3, break down, the completion time increases to $t3$. The example in Fig. 1 indicates that the number of machine breakdowns will heavily influence the completion time.

Most existing research has focused on optimizing one particular performance measure, for instance, either makespan, or total flow time, earliness time, or tardiness time[21], when the flow shop problem is supposed to be deterministic. In case the scheduling problem is subject to uncertainties, robustness and stability are two common objectives to reduce the effects of the uncertainty [7], [16], which has been demonstrated to be effective [22]. The robustness measure is to minimize the difference between the objective functions before and after a machine breakdown, and the stability measure aims to reduce

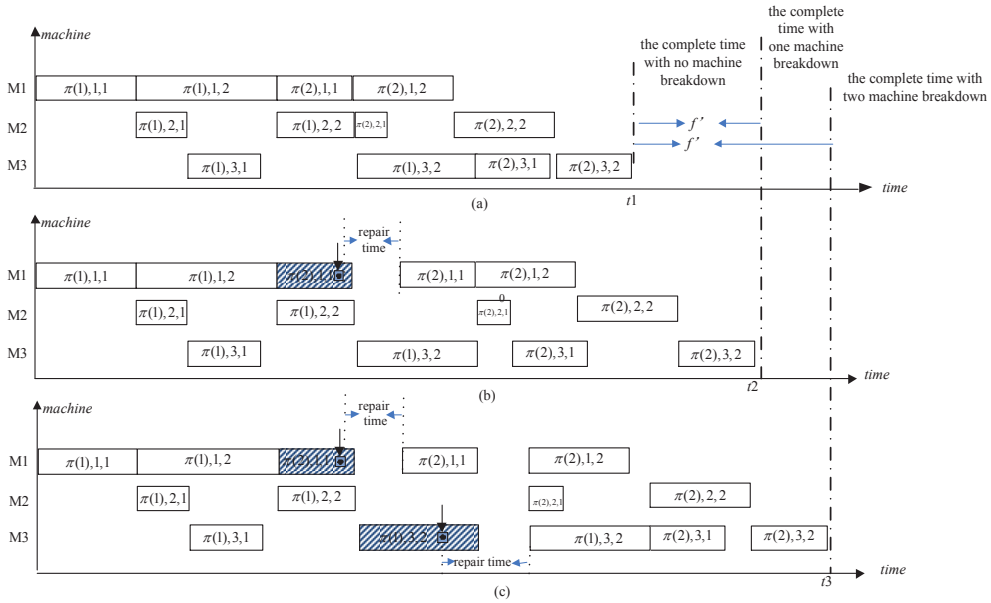

Fig. 1. Gantt of BLSFS without (with) machine breakdown

the discrepancy between the modified schedule strategy and the initial one.

In this study, for handling machine breakdowns, robustness based on makespan and the tardiness time and stability are also taken into account in addition to makespan and tardiness time when formulating the scheduling problem. Specifically, the robustness objectives based on makespan and the tardiness time, as well as the stability objective can be represented as follows.

$$\min f_1 = E(C_{\pi'(n),m,w_{\pi'(n)}}) - E(C_{\pi(n),m,w_{\pi(n)}}) \tag{14}$$

$$\min f_2(\pi) = \frac{1}{n}(\sum_{j=1}^{n} \max\{0, C_{\pi'(j),m,w_{\pi(j)}} - d_j\} - \sum_{j=1}^{n} \max\{0, C_{\pi(j),m,w_{\pi(j)}} - d_j\}) \tag{15}$$

$$\min f_3 = \frac{\sum_{i=1}^{m}\sum_{j=1}^{n} |E(C_{\pi'(j),i,w_{\pi'(j)}}) - E(C_{\pi(j),i,w_{\pi(j)}})|}{n \times m} \tag{16}$$

where π refers to the original job sequence, and π' denotes the modified sequence after implementing the rescheduling strategy, which will be described in detail in Section 4.3. f_1 and f_2 are the robustness objective based on makespan and the tardiness time, respectively. As shown in Fig.1, a small f_1 indicates a small difference between the makespan objectives before and after the machine breakdowns. Similarly, the smaller f_2 is, the less the customers will be influenced by the machine breakdowns. In addition, a small f_3 means a small modification in the scheduling, typically implying a smaller cost resulted from the machine breakdown.

IV. The Proposed Algorithm

The proposed evolutionary optimization algorithm is presented in Algorithm 1, where the maximal elapsed CPU time is regarded as the stopping criterion considered in the proposed algorithm. The parameter, *pc*, is the crossover probability, *r* is a random number in the range of [0,1] and $rand()$ is a function that generates a random integer in the range of [0, MAXINT]. The proposed algorithm is composed of the following three stages. In the first stage, an initial population is generated using solutions obtained by a slightly modified version of the single-objective heuristic algorithm proposed in [23] that minimizes either the makespan or the tardiness time for BLSFS problems without considering machine breakdown, referring to line 1. In the second stage, the population is evolved using an evolutionary multi-objective algorithm to simultaneously optimize the robustness and stability criteria of the BLSFS scheduling problem in the presence of machine breakdowns, seeing lines 3-11. The multi-objective evolutionary algorithm is characterized by two new crossover operators in addition to one insertion operator and a swap operator. Finally, a rescheduling strategy is adopted to further reduce the influence resulting from machine breakdowns, showing in lines 12-14. The details of the algorithm will be elaborated in following sections.

A. Population Initialization

The heuristic algorithm proposed by [23], NEH for short, is employed to generate an initial sequence (solution). The main idea is that jobs with a large total processing time (i.e., the sum of the processing time on all machines) should be scheduled as early as possible. Based on the above sequence, a new sequence is produced by performing $n(n+1)/2-1$ insertion operators to optimize one objective. As NEH is meant for single-objective optimization, a variant of NEH, termed vNEH

Algorithm 1 The framework of the proposed algorithm

Input: Initialize the parameters used in the proposed algorithm

Output: the non-dominated solution set

Initialize the population using a variant of NEH.
while the stopping criterion is not met **do**
  if $r < pc$ **then**
    if $rand()\%2$ **then**
      Implement the improved similar job order crossover operator;
    else
      Implement the improved similar block order crossover.
    end if
  else
    Implement the mutation operator.
  end if
  if machine breakdowns occur **then**
    Carry out the rescheduling strategy.
  end if
  Execute the selection and archive updating operators
end while

has been suggested to generate solutions for initializing the population. To promote the diversity of the initial population, random solution(s) within the neighborhood of the solutions obtained by vNEH will also be generated. The details of vNEH are presented in Algorithm 2.

Note again that vNEH solved the BLSFS scheduling problem by minimizing either the makespan or the tardiness time without considering machine breakdowns. Thus, the initial population will contain at least one solution that minimizes the makespan and one solution that minimizes the tardiness time, which can be used to calculate the robustness and stability criteria described in Eqs.(14-16). This initial population will be further evolved in the second stage, where machine breakdowns will be considered.

Algorithm 2 vNEH heuristic

Input: the number of jobs, *n*

Output: num initial solutions

let $\pi = \phi$,$\pi^* = \phi$,and $k = 0$;
Step 1: Generate a seed sequence, $\pi = \{\pi(1), \pi(2), ..., \pi(n)\}$, by sorting jobs according to their total processing time in a descending order;
Step 2: Pick the first two jobs of π, form two subsequences,$\pi(1), \pi(2)$ and $\pi(2), \pi(1)$, evaluate the performance of the subsequences, and select the one with the local minimal value of f' as the current sequence, π^*;
Step 3: let $k = 3$. carry out the insertion operator below:
while $k < n$ **do**
Pick the *k*-th job of π , obtain k subsequences by inserting it into *k* possible positions of the current sequence, π^*, and select the subsequence with the local minimal f' value as the current sequence;
$k = k + 1$;
end while
*/*There is no following statements for NEH.*/*
Insert $\pi(n)$ into the current sequence π^* at n possible positions, simultaneously evaluate two objectives, f' and f''. Denote the n complete sequences as $\mathcal{TS}$;
Set $\mathcal{C} = \emptyset$ and $\mathcal{TS}' \leftarrow \mathcal{TS}$;
while $|\mathcal{C}| < num$ **do**
  Seek non-dominated solutions in $\mathcal{TS}' \rightarrow \mathcal{D}$ based on the Pareto dominance relation;
  Set $k = |\mathcal{D}|$.
  if $k \geq (num - |\mathcal{C}|)$ **then**
    Randomly select $num - |\mathcal{C}|$ non-dominated solution(s) from $\mathcal{D} \rightarrow \mathcal{E}$.
    $\mathcal{C} = \mathcal{C} \bigcup \mathcal{E}$.
  else
    Set $\mathcal{C} = \mathcal{C} \bigcup \mathcal{D}$ and $\mathcal{TS}' = \mathcal{TS}' \backslash \mathcal{D}$.
  end if
end while
Output num initial solutions.

B. Crossover and mutation operators

Crossover and mutation operators play an important role in evolutionary algorithms, which generate new candidate solutions for selection. Such operators become more critical for search performance in scheduling than in continuous optimization. Consequently, a large body of research in evolutionary scheduling has been dedicated to designing problem-specific crossover and mutation operators to enhance the performance of evolutionary algorithms. However, not many variation operators have been proposed for solving multi-objective scheduling problems, where valuable information contained in the non-dominated solutions can be taken advantage of.

In the following, we present two new crossover operators together with two mutation operators.

1) Crossover operators: A variety of crossover operators have been developed, such as similar job order crossover (SJOX), similar block order crossover (SBOX)[24], order crossover (OX) [25], one-point order crossover (OP), two-point order crossover (TP) [26], and generalized position crossover (GPX) [27]. Previous studies have shown that SJOX and SBOX outperform the others. In addition, [28] proposed a new crossover operator based on SJOX, called artificial chromosome SJOX (ACJOX), which can produce more promising candidate solutions than SJOX [28].

Empirical results have shown that both SBOX and ACJOX operators are superior to most of their counterparts [28]. However, a common weakness of the SBOX and ACJOX operators is that they use two parents only to generate offspring individuals. In addition, SBOX does not take identical gene blocks in the parents at different loci. As they both were designed for single-objective optimization, neither SBOX nor ACJOX takes non-dominated solutions into account. Due to the above reasons, two enhanced crossover operators based on SBOX and ACJOX, respectively, are proposed for the multi-objective BLSFS scheduling problem, which are named improved SBOX (ISBOX) and improved SJOX (ISJOX), respectively.

ISBOX consists of the following steps. First, a temporary

set composed of a number of gene blocks (containing at least two genes) that appear frequently in the chromosome of all non-dominated solutions achieved so far. Then, two parents are randomly selected from the parent population. Then, randomly select one of the parents and identify if there are common gene blocks between the parent and the temporary set. If yes, put the identical gene blocks into its offspring at the same locus (loci) as in the parent. The genes of the offspring in the rest loci are filled up based on the OP crossover using the two parents. The second offspring individual is generated in a similar way using the parent and temporary set.

It is worth noting that ISBOX considers the common gene blocks between the temporary set and the parents at any loci, avoiding loss of promising gene blocks in the parents. It should also be pointed out that there exists a specific situation where no common gene block between the temporary set and the parents is identified. In this situation, ISBOX becomes the traditional OP crossover operator.

In the following, we present an example to illustrate how ISBOX works. Suppose there are 5 non-dominated solutions in the current archive, denoted by $\pi_i, i = 1, 2, 3, 4, 5$, with each containing 7 jobs. Their expressions are given as follows.

$$\pi_1 = \{2, 6, 3, 1, 4, 5, 7\};$$
$$\pi_2 = \{3, 5, 1, 6, 7, 2, 4\};$$
$$\pi_3 = \{4, 5, 2, 1, 7, 3, 6\};$$
$$\pi_4 = \{4, 1, 7, 2, 6, 3, 5\};$$
$$\pi_5 = \{4, 5, 2, 6, 3, 7, 1\};$$

To perform the ISBOX crossover, a temporary set is generated using the following three steps.

Step 1: Count the times that job *j* ($j = 1, 2, ..., 7$) appears immediately after job *i* ($i = 1, 2, ..., 7$) in all non-dominated solutions. In this example, the times that jobs 1, 2, 3, 4, 5, 6, and 7 appears immediately after jobs 1 are 0, 0, 0, 1, 0, 1 and 2, respectively. Similarly, the times that jobs 1, 2, 3, 4, 5, 6, and 7 appear immediately after job *i* ($i = 2, ..., 7$) can be found as follows:

job 2: 1, 0, 0, 1, 0, 3, and 0;
job 3: 1, 0, 0, 0, 2, 1, and 1;
job 4: 1, 0, 0, 0, 3, 0, and 0;
job 5: 1, 2, 0, 0, 0, 0, and 1;
job 6: 0, 0, 3, 0, 0, 0, and 1;
job 7: 1, 2, 1, 0, 0, 0, and 0.

Step 2: Put gene blocks *i,j* that job *j* appears immediately after job *i* with the highest into the temporary set.

Step 3: Repeat Step 2 to obtain a complete temporary set, {{1,7}, {2,6}, {3,5}, {4,5}, {5,2}, {6,3}, {7,2}}.

Following this, randomly select two parents from the population, e.g., $parent1 = \{1, 2, 6, 4, 5, 3, 7\}$ and $parent2 = \{5, 2, 7, 1, 4, 3, 6\}$. Find out the common gene blocks between $parent1$ and the temporary set, i.e., {2,6} and {4,5} and put them into $offspring1$ at loci 2, 3, 4, and 5.

To obtain the genes for the unfilled loci of $offspring1$, randomly choose a crossover point (in this example between gene 4 and gene 5) and perform OP crossover on $parents1$ and 2. This results in $offspring1$ to be {1,2,6,4,5,3,6}. Note however that the obtained offspring is infeasible, as it contains job 6 twice and job 7 is not included. Therefore, this offspring is repaired by replacing job 6 on locus 7 with job 7 (note that job 6 at locus 2 is part of a gene block and should not be changed). Because job 7 appears before job 3 in parent2, we exchange the positions of jobs 3 and 7 to preserve the gene block in $parent2$. The above steps result in $offspring1$ {1,2,6,4,5,7,3}. Similarly, $offspring2$ {5,2,7,1,4,3,6} will be generated. The whole process of generating $offspring1$ and 2 is illustrated in Fig. 2.

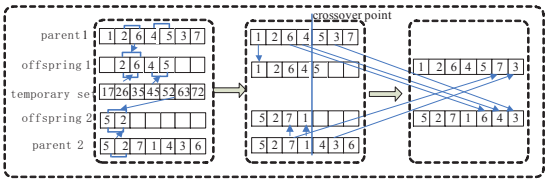

Fig. 2. Process of performing ISBOX

The second new crossover operator, ISOJX, can be described by the following steps. First, generate a temporary individual based on all non-dominated solutions in the current archive. Then randomly select two parents from the population and compare the genes of each parent with those of the temporary individual at the same locus. If they are the same, put the gene in the same locus of its offspring. Similar to ISBOX, the genes in the rest loci of the offspring are generated by performing OP between the two parents. The second offspring is created in the same way.

In the following, we use the same example used above to exemplify the main steps of ISOJX.

First, a temporary individual is generated according to the following three steps.

Step 1: For all non-dominated solutions, count the times that job i appears at position *k* ($k = 1, 2, ..., 7$). For example, the times that job 1 appears at loci 1, 2, 3, 4, 5, 6, and 7 is 0, 1, 1, 2, 0, 0, and 1, respectively. Similarly, the times that jobs 2, 3, 4, 5, 6 and 7 appear at loci 1, 2, 3, 4, 5, 6, and 7 can be given as follows:

job 2: 1, 0, 2, 1, 0, 1, and 0;
job 3: 1, 0, 1, 0, 1, 2, and 0;
job 4: 3, 0, 0, 0, 1, 0, and 1;
job 5: 0, 3, 0, 0, 0, 1, and 1;
job 6: 0, 1, 0, 2, 1, 0, and 1;
job 7: 0, 0, 1, 0, 2, 1, and 1.

Step 2: Put the job that appears most frequently at locus k into the temporary individual at the same locus.

Step 3: Repeat Step 2 until the temporary individual is a complete sequence, i.e., {4,5,2,1,7,3,6}. Following this, randomly select two parents from the population, e.g., $parent1 = \{1, 2, 6, 4, 5, 3, 7\}$ and $parent2 = \{5, 2, 7, 1, 4, 3, 6\}$. By comparing temporary individual and $parent1$, we find that the gene at locus 6 is that same, i.e., job 3. Thus, assign job 3 to locus 6 of $offspring1$.

Finally, the genes at the rest of the loci of $offspring1$ are obtained by perform a OP on $parent1$ and $parent2$ by

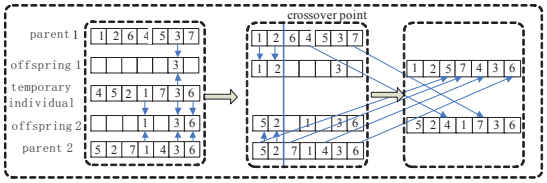

Fig. 3. Process of performing ISJOX

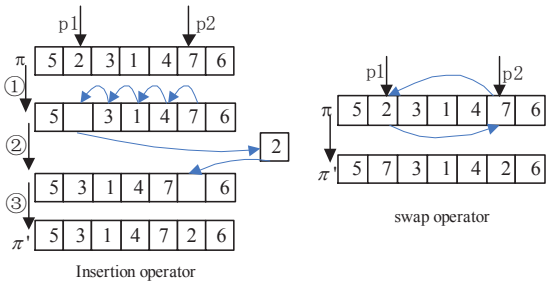

Fig. 4. Process of performing mutation operators

randomly choosing a crossover point, say between gene 2 and gene 3, resulting $offspring1$ to be {1,2,7,1,4,3,6}. Again, $offspring1$ is an infeasible solution and needs to be repaired. Since job 1 appears twice, and job 5 is absent, one of job 1 in the offspring should be substituted with job 5. Because job 5 appears before job 7 in $parent2$, job 1 at locus 4 is replaced by job 5 and then the order of jobs 5 and 7 are exchanged to preserve the gene block in $parent2$. Thus, $offspring1$ is obtained to be {1,2,5,7,4,3,6}, $offspring2$ can be generated similarly, which is {5,2,4,1,7,3,6}. The whole process of ISOJX or generating $offspring1$ and 2 is depicted in Fig. 3.

2) Mutation operators: Although various mutation operators have been used to further change the offspring individuals generated by crossover, it has been shown in [1] that two mutation methods, insertion and swap, are very efficient. Thus, in this study, we randomly select one of them as the mutation operator in the proposed evolutionary algorithm. In the following, a process how to utilize the insertion and swap operators suggested in [1] will be briefly introduced.

The sequence to be mutated is denoted as π and the mutated sequence π'. Suppose that the loci to perform the insertion operator are $p1$ and $p2$. They are randomly generated, and let $p1 < p2$. Then, put all genes of π at loci from $p1$+1 to $p2$ into π' at loci from $p1$ to $p2$-1, and then put the gene of π at locus $p1$ into π' at locus $p2$. Finally, put gene(s) of π at the other locus (loci) into π' at the same locus (loci). An example is provided on the left panel of Fig. 4.

In the swap operator, genes of π at loci $p1$ and $p2$ are put into π' at loci $p2$ and $p1$, respectively, and gene(s) of π at the other locus (loci) are put into π' at the same locus (loci). An example of the swap operator is given on the right panel of Fig. 4. The crossover operators and mutation operators will be performed on each individual with a probability of *pc*, and 1-*pc*, respectively. It is noted that if a crossover operator is to be performed, one of the above two crossover operators is randomly selected. Similarly, one of the above insertion and swap operators is randomly adopted when a mutation operator is to be employed.

C. The rescheduling strategy

Rescheduling refers to locally adjusting the current schedule strategy, and is preferable because of its potential in saving computational time and preserving the stability of the scheduled plan. Given that an unforeseen machine breakdown may affect not only the original objectives, but also the scheduling strategy, in this study, a reference local search based on the insertion operator proposed in [29] is adopted as the rescheduling strategy to maintain the stability of the original schedule strategy and further reduce the disturbance to the original objectives.

When a machine breaks down, at least one job is suspended if it is being processed on that machine. At this moment, the original sequence is divided into the following two subsequences. The first consists of the finished job(s) whose relative position(s) will not be changed. The second subsequence contains the unscheduled job(s), which forms a reference sequence, denoted by π^r. If an interrupted job is $\pi(d)$, the reference sequence will be $\pi^r = \{\pi^r(1), \pi^r(2), ..., \pi^r(i), ..., \pi^r(n-d)\}$, where $\pi^r(i)$ is equal to $\pi(d+i)$, *i*=1,2,...,*n-d*. Denote the sequence after rescheduling as π', the rescheduling process based on the local search can be described by Algorithm 3.

Algorithm 3 The rescheduling strategy

Input: the number of jobs, *n*; the number of machines, *m*; the position of the interrupted job, *d*; the temporary set φ φ=ϕ.

Output: a number of good solutions

1: Let *i*=1;
2: **while** $i < n - d$ **do**
3: Pick job, $\pi^r(i)$, from the sequence,π, and obtain a subsequence, π' ;
4: *i*=*i*+1;
5: Insert job $\pi^r(i)$ into position *k* (*k*=*d*+1,*d*+2,...,*n*)of sequence,π', respectively, and obtain $n - d + 1$ complete sequence π.
6: Select the sequences using the minimal robustness and stability criteria based on the Pareto dominance relation, and put them into the temporary set.
7: **end while**
8: Update the temporary set φ by deleting the dominated solutions.

In Algorithm 3, first, we obtain a reference subsequence π' containing $n - d$ jobs. Second, we take a job from π^r (as shown in line 3), insert it into $n - d + 1$ positions of the sequence π', and evaluate the $n - d + 1$ complete sequences,

referring to lines 5-6. Third, we repeatedly pick a job from π^r and implement the insertion operators of lines 5-6 until π^r is empty, as described in lines 2-7. By this way, we obtain some solutions and put them into the temporary set, φ . Thus, last, we update the temporary set by deleting the dominated solutions using Pareto dominance relation. The above method can locally adjust an obtained job sequence to seek for a new sequence having good robustness and stability. Note that there are *n-d* iterations for Lines 5, each having a computation complexity of $O(n-d)$. Thus the total computation complexity of the rescheduling strategy is $O((n-d)^2)$.

In Algorithm 1, we adopt the proposed crossover operator to generate new solutions. At least one job will be suspended if the new solution is processed on a machine which breaks down. The reader is referred to Subsection IV.B for detailed descriptions of the crossover operators. In the following, we give an example of the rescheduling strategy.

Let the current solution is $\pi = (3,1,4,6,2,5)$. Assume job 4 is being processed on machine 2, which breaks down. At this moment, the original sequence π is divided into the following two subsequences, i.e, $\pi'' = (3,1,4)$ and $\pi^r = (\pi^r(1), \pi^r(2), \pi^r(3)) = (6,2,5)$.

Let $\pi' = \pi/\pi^r(1)$=(3,1,4,2,5). Insert $\pi^r(1)$ into π' at the fourth, fifth and sixth position, respectively, which results in three subsequences, (3,1,4,6,2,5), (3,1,4,2,6,5) and (3,1,4,2,5,6). Evaluate the subsequences in terms of f_1 and f_2. Select the solutions according to line 6 of Algorithm 3, and put them into a temporary set φ.

Let $\pi' = \pi/\pi^r(1)$=(3,1,4,6,5). Insert $\pi^r(2)$ into π' at the fourth, fifth and sixth position, respectively, resulting in three subsequences, (3,1,4,2,6,5), (3,1,4,6,2,5) and (3,1,4,6,5,2). Evaluate the subsequences in terms of f_1 and f_2. Select the solutions according to line 6 of Algorithm 3, and put them into a temporary set φ.

Similarly, let $\pi' = \pi/\pi^r(1)$=(3,1,4,6,2). Insert $\pi^r(3)$ into π' at the fourth, fifth and sixth position, respectively, which results in subsequences, (3,1,4,2,6,5), (3,1,4,6,2,5) and (3,1,4,6,5,2). Evaluate the subsequences in terms of f_1 and f_2. Select the solutions according to line 6 of Algorithm 3, and put them into temporary set φ.

Finally, update φ by deleting the dominated solutions.

D. Selection and Archive Updating

An external archive of a limited size is employed to store the non-dominated solutions found so far, in order to avoid losing good solutions. During the search, these solutions are iteratively updated by deleting not only dominated solutions but also non-dominated solutions with a small value of the crowding distance. The crowding distance is calculated as in [30].

Before, the parent population is merged with the external archive and temporary set, φ , to form a combined population. All individuals are sorted into a number of fronts based on the non-dominance relationship and a crowding distance is calculated as in [30]. Finally, *PS*(the size of the parent population) individuals are selected based on the front number and the crowding distance of solutions. For more details of the selection procedure, please refer to [30].

V. EMPIRICAL COMPARISONS

We have deveoped improved NSGA-II algorithm (INSGA-II) to solve multi-objective LSFS scheduling problems [1], where an estimation of distribution algorithm (EDA) is adopted to replace the traditional crossover operator, and a restarting strategy is employed to increase the diversity of the population. In [31] a Pareto block-based EDA (PBEDA) for multi-objective permutation flow shop scheduling problems was proposed. In PBEDA, a bi-variate probabilistic model is utilized to generate blocks, and the non-dominated sorting technique is employed to filter solutions. For the same problem, Li and Ma [32] proposed a novel multiobjective memetic search algorithm (MMSA), in which a global search and local search strategies are used to find the promising solutions. Recently, Wang and Tang [33] developed a machine-learning based multi-objective memetic algorithm combined with multi-objective local search (MOMA) to solve the above optimization problem.

The above state-of-the-art multi-objective algorithms, i.e., INSGA-II[1], PBEDA[31], MMSA[32], and MOMA[33], were selected to compare with our proposed algorithm. As indicated in Section 2, little work has been reported on solving a multi-objective LSFS scheduling problem with machine breakdowns. As a result, it is hard to directly compare the proposed algorithm with those of solving a BLSFS scheduling problem without machine breakdowns. To evaluate the performances of the rescheduling strategy proposed in this paper in robustness and stability, the strategy is incorporated into the four compared algorithms, and compared these algorithms with their counterparts without the strategy. On the premise of the same evolutionary strategies, if these algorithms with the proposed rescheduling strategy obtain solutions better than those without the strategy, the proposed rescheduling strategy will be beneficial to improving an algorithm in tacking a BLSFS scheduling problem with machine breakdowns.

The standard test instances of the lot-streaming flow shop scheduling problem used in the experiments were originally proposed in [20]. To more rigorously evaluate the proposed algorithm, we have added more test instances. The test set is composed of 220 instances, which are divided into 22 subsets, with each subset consisting of ten instances of the same size. For each subset, the size of instances is changed from 30 jobs and 5 machines to 500 jobs and 20 machines. Each instance is independently executed five replications. The parameter settings of these instances, including processing time, due date, and the number of sublots of each job, are given by a series of discrete uniform distributions, and listed in Table I.

In the experiments, all the algorithms are written in Visual C++ 6.0, and the same library functions are employed to make fair comparisons. All algorithms are implemented on a PC with Pentium (R) Dual 2.79 GHz and 1.96 G memory, whose operation system is Microsoft Windows 7 X64. For the termination criterion of these algorithms, the same maximum elapsed CPU time of $30 \times n \times m$ milliseconds is employed, where*n*represents the number of jobs, and *m* refers to the number of machines.

All performance comparisons are conducted using the Hy-

TABLE I
PARAMETER SETTINGS

parameter	notation	value
number of jobs	n	$\{30, 50, 70, 90, 110, 200, 500\}$
number of machines	m	$\{5, 10, 15, 20\}$
due date of job j	d_j	$randIn()$ mod$(15m + 1) + 15n$
number of sublots of job j	$w_{\pi(j)}$	$randIn()$ mod $6 + 1$
processing time of j on i	$p_{\pi(j),i}$	$randIn()$ mod $31 + 1$
population size	PS	20
the number of initial solutions	num	10
crossover probability	pc	0.9
external archive size	EA	100
stopping condition	CPU time	$30 \times n \times m$ millisecond
number of machine breakdowns	β	$\{5, 10, 15\}$

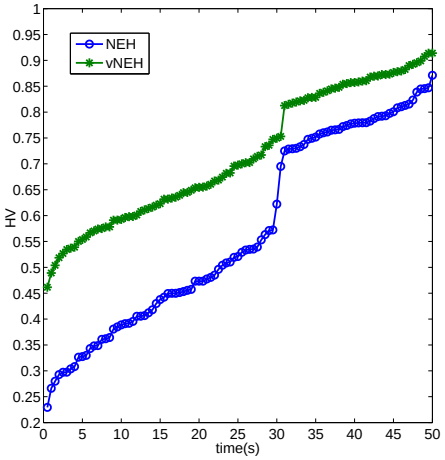

Fig. 5. Change of the HV indicator over tine.

pervolume (HV for short) indicator [34]. HV is able to account for both convergence and diversity of the non-dominated solutions obtained by an algorithm. Here, (1, 1) is chosen as the reference point and a larger HV indicates better performance. In the experiment, we randomly sampled 100 machine breakdown cases whose repair times are generated using Equations (11) and (13), respectively. For each instance, the corresponding objective values of every solution are calculated, and the averages of 100 cases are considered as the objective value of every solution.

1) Performance of the initialization strategy: We first evaluate the performance of the vNEH and NEH strategies by comparing the convergence profiles of the two methods, given the same genetic operators, rescheduling method and parameter settings for $\beta = 15$. We run two algorithms with a CPU time of 50 seconds on the aforementioned PC. The convergence profile, denoted by the change of the average HV indicator for all instances, where the non-dominated sets obtained by the above compared methods are employed as the reference set, are plotted in Fig. 5.

From Fig. 5, we can see that the population initialized with the solutions generated by vNEH converges faster than that using NEH. This indicates that vNEH is more efficient in seeding promising solutions for the multi-objective evolutionary search.

2) Performance of the proposed crossover operator: First, to demonstrate the effectiveness of the proposed crossover operators, a sensitivity analysis of parameter pc is conducted. Without loss of generality, pc is changed from 0 to 1.0 at a step size of 0.1. The instances and the settings of the other parameters remain the same as those in Section V. The corresponding results with respect to the HV indicators are plotted in Table II.

The following observations can be made from the results in Table II. First, the proposed algorithm without the crossover operators ($pc = 0$) performs very poorly. Second, the proposed algorithm performs slightly poorly when pc =1 than when $pc = 0.9$. The reason might be that if there are no mutation operators, offspring are generated by using the crossover operators only, resulting in getting stuck in local optima of the scheduling problems. Third, for most instances, the proposed algorithm performs better as pc increases, and it achieves the best results for $pc = 0.9$. From these experimental results, we set the value of pc to 0.9.

Second, we evaluate the performance of the proposed crossover operators, i.e., ISBOX and ISJOX by comparing them with the SBOX and SJOX, given the same genetic operators and parameter settings. Table III presents the comparative results in terms of the HV indicator, where the non-dominated solutions obtained by the above compared methods are employed as the reference set. "ISBJ" denotes the results when the improved crossover operators are applied as whereas "SBJ" refers to the results when SBOX and SJOX are applied. The better results of the two methods are highlighted.

TABLE III
PERFORMANCES OF THE PROPOSED AND THE COMPARED CROSSOVER OPERATORS

test sets	SBJ	ISBJ	test sets	SBJ	ISBJ
30×5	7.3146E-01	**8.8405E-01**	70×20	4.5157E-01	**5.0313E-01**
30×10	7.0632E-01	7.0335E-01	90×5	4.9255E-01	**5.7237E-01**
30×15	7.2125E-01	**7.9860E-01**	90×10	4.0169E-01	**4.5026E-01**
30×20	7.8174E-01	7.8455E-01	90×15	3.4274E-01	**4.0287E-01**
50×5	6.8835E-01	**7.1105E-01**	90×20	2.1018E-01	**3.7098E-01**
50×10	6.0502E-01	**6.9224E-01**	110×5	3.2463E-01	**3.9050E-01**
50×15	5.5691E-01	**6.4334E-01**	110×10	3.2762E-01	**4.2316E-01**
50×20	4.9692E-01	**5.5325E-01**	110×15	3.0455E-01	**4.0574E-01**
70×5	6.7090E-01	**7.3814E-01**	110×20	3.3092E-01	**4.1405E-01**
70×10	4.5879E-01	**5.5978E-01**	220×20	2.3592E-01	**3.7691E-01**
70×15	5.0036E-01	**6.0227E-01**	500×20	2.4561E-01	**3.6749E-01**

From Table III, we can see that the improved crossover achieved better performance in 20 out of 22 test sets in terms of the HV indicator. From the above results, we can conclude that the proposed improved crossover operators can generate more promising candidate solutions than the original ones due to their ability in taking advantage of the information in the non-dominated solutions.

3) Comparison of the overall performance: The five algorithms namely, INSGA, PBEDA, MMSA, MOMA and REMO, are compared in three different machine breakdown situations, where the number of machine breakdowns (β) is set to 5, 10, and 15, respectively. The experimental results are listed in Tables IV-VI, respectively, where a row represents results obtained by different algorithms when solving an instance of the BLSFS scheduling problem with machine breakdowns. The row in the bottom, denoted by 'mean', presents the

TABLE II
INFLUENCES OF pc ON HV INDICATOR

test sets	0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
30×5	3.249E-01	3.949E-01	4.937E-01	5.121E-01	5.653E-01	6.660E-01	6.435E-01	6.826E-01	7.318E-01	**8.840E-01**	6.422E-01
30×10	3.348E-01	3.110E-01	4.109E-01	4.324E-01	4.776E-01	5.201E-01	6.778E-01	**7.525E-01**	6.435E-01	7.227E-01	7.447E-01
30×15	2.590E-01	2.829E-01	3.741E-01	4.073E-01	4.538E-01	5.766E-01	6.261E-01	7.139E-01	**7.716E-01**	7.715E-01	7.602E-01
30×20	2.410E-01	2.246E-01	3.253E-01	4.229E-01	4.677E-01	5.009E-01	6.447E-01	7.504E-01	7.607E-01	**7.855E-01**	7.452E-01
50×5	2.080E-01	3.578E-01	3.285E-01	4.284E-01	4.386E-01	5.818E-01	5.162E-01	6.639E-01	**7.430E-01**	7.093E-01	6.766E-01
50×10	2.271E-01	2.530E-01	2.989E-01	3.369E-01	3.516E-01	4.142E-01	5.078E-01	5.236E-01	5.923E-01	**6.187E-01**	5.268E-01
50×15	2.019E-01	2.255E-01	3.561E-01	4.094E-01	4.351E-01	4.033E-01	4.547E-01	**6.295E-01**	5.954E-01	6.124E-01	5.798E-01
50×20	1.051E-01	2.144E-01	2.731E-01	2.528E-01	3.570E-01	3.624E-01	4.012E-01	4.490E-01	4.897E-01	**5.218E-01**	4.251E-01
70×5	2.194E-01	2.212E-01	2.622E-01	2.450E-01	3.018E-01	4.052E-01	4.100E-01	5.494E-01	6.257E-01	**7.050E-01**	6.473E-01
70×10	1.260E-01	1.909E-01	2.593E-01	3.024E-01	3.398E-01	3.409E-01	3.925E-01	4.233E-01	4.386E-01	**5.059E-01**	4.138E-01
70×15	1.189E-01	1.243E-01	2.221E-01	2.523E-01	2.332E-01	3.310E-01	3.578E-01	4.338E-01	4.548E-01	**5.012E-01**	4.292E-01
70×20	1.673E-01	1.650E-01	2.079E-01	3.149E-01	3.053E-01	3.563E-01	4.368E-01	4.744E-01	**4.903E-01**	4.823E-01	3.796E-01
90×5	1.513E-01	1.730E-01	2.397E-01	2.790E-01	2.520E-01	3.067E-01	3.800E-01	3.660E-01	4.191E-01	**5.079E-01**	4.243E-01
90×10	1.406E-01	2.615E-01	2.138E-01	2.865E-01	3.056E-01	3.471E-01	3.510E-01	4.059E-01	**4.549E-01**	4.450E-01	4.181E-01
90×15	6.397E-02	1.496E-01	1.695E-01	1.909E-01	1.756E-01	2.552E-01	2.909E-01	3.357E-01	3.602E-01	**3.855E-01**	2.610E-01
90×20	4.315E-02	9.081E-02	2.011E-01	1.963E-01	2.019E-01	2.428E-01	2.129E-01	2.772E-01	2.768E-01	**3.258E-01**	2.248E-01
110×5	1.011E-01	1.312E-01	1.334E-01	1.511E-01	1.850E-01	1.645E-01	2.091E-01	1.952E-01	2.392E-01	**2.650E-01**	2.021E-01
110×10	1.217E-01	2.108E-01	2.087E-01	2.217E-01	2.822E-01	2.028E-01	3.441E-01	4.168E-01	**4.500E-01**	4.191E-01	2.289E-01
110×15	2.558E-02	1.333E-01	1.496E-01	1.599E-01	2.214E-01	2.604E-01	2.728E-01	2.240E-01	3.421E-01	**3.861E-01**	3.019E-01
110×20	8.776E-02	1.015E-01	1.118E-01	1.178E-01	1.949E-01	1.518E-01	2.287E-01	2.833E-01	2.788E-01	**3.903E-01**	1.920E-01
220×20	1.013E-01	1.052E-01	1.077E-01	2.087E-01	1.509E-01	2.652E-01	2.709E-01	3.146E-01	3.219E-01	**3.283E-01**	1.932E-01
500×20	9.765E-02	1.383E-01	1.392E-01	2.646E-01	2.107E-01	2.800E-01	2.992E-01	3.071E-01	2.947E-01	**3.076E-01**	1.897E-01

average values of HV over 22 test sets. The larger the mean value is, the better the algorithm. To make a fair comparison, all compared algorithms adopt the same maximum elapsed CPU time of $30\times n\times m$ milliseconds as a termination criterion. In addition, for convenience, the five compared algorithms without rescheduling strategy are denoted as INSGAn, PBEDAn, MMSAn, MOMAn and REMOn, respectively.

TABLE IV
PERFORMANCE COMPARISON OF INSGA, PBEDA, MMSA, MOMA AND REMO ($\beta = 5$)

test sets	INSGA	PBEDA	MMSA	MOMA	REMO
30×5	8.9701E-01	8.9461E-01	**9.0818E-01**	9.0785E-01	8.9887E-01
30×10	8.6003E-01	8.6224E-01	8.8220E-01	8.6143E-01	**8.8873E-01**
30×15	8.7850E-01	8.6409E-01	8.8118E-01	**8.9351E-01**	8.8290E-01
30×20	7.0205E-01	7.1274E-01	7.6611E-01	**7.6806E-01**	7.6005E-01
50×5	6.8165E-01	6.7988E-01	7.0285E-01	7.0484E-01	**7.5295E-01**
50×10	6.4752E-01	6.5613E-01	7.1867E-01	7.2729E-01	**7.7073E-01**
50×15	6.3552E-01	6.4437E-01	6.7297E-01	6.8686E-01	**6.8867E-01**
50×20	4.2538E-01	4.3549E-01	4.4282E-01	4.5536E-01	**4.6954E-01**
70×5	5.1585E-01	5.1617E-01	5.4426E-01	5.7596E-01	**5.8060E-01**
70×10	3.8740E-01	3.9230E-01	3.9426E-01	4.0508E-01	**4.2862E-01**
70×15	4.0448E-01	4.2849E-01	4.2272E-01	4.3377E-01	**4.7400E-01**
70×20	4.5122E-01	4.9225E-01	5.1200E-01	5.2938E-01	**5.3878E-01**
90×5	3.6501E-01	4.1372E-01	4.2914E-01	4.2424E-01	**4.5380E-01**
90×10	2.2531E-01	2.3395E-01	3.2536E-01	3.2894E-01	**3.9616E-01**
90×15	2.7432E-01	2.6230E-01	3.1146E-01	3.1616E-01	**3.8184E-01**
90×20	3.4490E-01	3.2067E-01	3.6371E-01	3.8549E-01	**4.1800E-01**
110×5	4.6041E-01	4.6079E-01	5.0907E-01	5.1541E-01	**5.8325E-01**
110×10	2.8000E-01	2.5542E-01	3.4902E-01	3.1888E-01	**3.8616E-01**
110×15	2.1953E-01	2.2602E-01	2.5977E-01	2.6455E-01	**3.0587E-01**
110×20	3.2005E-01	3.9935E-01	3.9554E-01	4.0550E-01	**4.5677E-01**
220×20	2.9971E-01	3.0095E-01	3.4134E-01	3.5220E-01	**3.7656E-01**
500×20	2.7026E-01	2.8974E-01	2.8334E-01	2.9937E-01	**3.1075E-01**
mean	4.7937E-01	4.8826E-01	5.1891E-01	5.2546E-01	**5.5471E-01**

From Tables VII, we can conclude that INSGAn, PBEDAn, MMSAn, MOMAn and REMOn perform worse than INSGA, PBEDA, MMSA, MOMA and REMO. The reason is that the proposed rescheduling strategy can reduce the negative influence resulting from machine breakdowns, and reduce the difference between the completion time before and after machine breakdowns. In addition, the mean HV produced by the proposed algorithm, REMO, is 5.5471E-01, 5.6294E-01 and 6.0117E-01 for β= 5, 10, and 15, respectively, which is larger than any mean HV obtained by the compared algorithms.

TABLE V
PERFORMANCE COMPARISON OF INSGA, PBEDA, MMSA, MOMA AND REMO ($\beta = 10$)

test sets	INSGA	PBEDA	MMSA	MOMA	REMO
30×5	8.6868E-01	8.5082E-01	8.6026E-01	**8.7931E-01**	8.6670E-01
30×10	7.3526E-01	7.8088E-01	7.8275E-01	7.8134E-01	**7.8374E-01**
30×15	7.9681E-01	7.7604E-01	7.9908E-01	**8.0821E-01**	8.0223E-01
30×20	7.3458E-01	7.4529E-01	7.4855E-01	7.6637E-01	**7.9014E-01**
50×5	5.4198E-01	6.0660E-01	6.9019E-01	6.8755E-01	**6.9378E-01**
50×10	6.1779E-01	6.1618E-01	7.0900E-01	7.1427E-01	**7.7364E-01**
50×15	5.2846E-01	5.3108E-01	5.3552E-01	5.4030E-01	**5.4424E-01**
50×20	5.7922E-01	5.7760E-01	**6.0537E-01**	6.0225E-01	5.9993E-01
70×5	5.0915E-01	5.2482E-01	5.6660E-01	5.7866E-01	**5.9664E-01**
70×10	3.9856E-01	3.9172E-01	4.2521E-01	**4.4225E-01**	4.3885E-01
70×15	4.0228E-01	3.8406E-01	3.9261E-01	4.0298E-01	**4.1501E-01**
70×20	3.6449E-01	3.6668E-01	3.9069E-01	4.0578E-01	**4.1280E-01**
90×5	4.1146E-01	4.4236E-01	4.7458E-01	4.8862E-01	**4.9137E-01**
90×10	5.4875E-01	5.5171E-01	5.6804E-01	5.6408E-01	**5.8987E-01**
90×15	5.4121E-01	5.5991E-01	5.5842E-01	5.7337E-01	**6.0092E-01**
90×20	4.3272E-01	4.3028E-01	4.2148E-01	4.5139E-01	**4.8996E-01**
110×5	4.0936E-01	4.1674E-01	4.3047E-01	4.2976E-01	**4.9288E-01**
110×10	2.5212E-01	2.5326E-01	2.8119E-01	2.6133E-01	**3.0340E-01**
110×15	3.6100E-01	3.4772E-01	3.9799E-01	4.0094E-01	**4.3973E-01**
110×20	2.2945E-01	2.2383E-01	2.1604E-01	2.3781E-01	**2.8491E-01**
220×20	3.3227E-01	3.2128E-01	3.3768E-01	3.4026E-01	**4.0316E-01**
500×20	4.6797E-01	4.5506E-01	5.1076E-01	5.1328E-01	**5.7081E-01**
mean	5.0289E-01	5.0700E-01	5.3193E-01	5.3955E-01	**5.6294E-01**

Hence we can conclude that the proposed algorithm is effective. Overall, REMO outperforms the compared algorithms for the problem under consideration. Finally, as the number of machine breakdowns increases, the advantage of the proposed algorithm over the compared ones becomes more obvious.

We have studied the overall performance of all the compared algorithms in terms of HV from the following two aspects:

(1) All the algorithms are compared in scenarios of different machine breakdowns, i.e., the number of machine breakdowns is 5, 10, and 15, respectively. The experimental results listed in Tables IV-VI indicate that the proposed algorithm outperforms its counterparts at a considerable margin.

(2) All the algorithms with and without the rescheduling strategy are investigated to evaluate influence of the rescheduling strategy, and the experimental results are listed in Table VII. We can conclude from Table VII that the proposed rescheduling strategy can alleviate the negative influences

TABLE VII
PERFORMANCE COMPARISON OF INSGA, PBEDA, MMSA, MOMA AND REMO WITH AND WITHOUT RESCHEDULING STRATEGY($\beta = 15$)

test sets	INSGA	PBEDA	MMSA	MOMA	REMO	INSGAn	PBEDAn	MMSAn	MOMAn	REMOn
30×5	**8.498E-01**	**8.569E-01**	**8.225E-01**	**8.641E-01**	**8.518E-01**	2.066E-01	2.673E-01	3.955E-01	4.850E-01	4.513E-01
30×10	**8.458E-01**	**8.548E-01**	**8.621E-01**	**8.715E-01**	**8.994E-01**	2.682E-01	2.716E-01	3.264E-01	5.086E-01	4.962E-01
30×15	**7.942E-01**	**8.055E-01**	**8.372E-01**	**8.190E-01**	**8.772E-01**	1.616E-01	2.438E-01	2.662E-01	3.620E-01	4.531E-01
30×20	**8.099E-01**	**8.016E-01**	**8.163E-01**	**8.188E-01**	**8.076E-01**	1.330E-01	2.069E-01	2.293E-01	4.733E-01	5.723E-01
50×5	**7.903E-01**	**7.919E-01**	**7.962E-01**	**8.068E-01**	**8.016E-01**	1.315E-01	2.101E-01	3.275E-01	4.806E-01	5.393E-01
50×10	**7.483E-01**	**7.689E-01**	**7.827E-01**	**7.922E-01**	**5.099E-01**	1.519E-01	1.917E-01	2.973E-01	4.733E-01	4.494E-01
50×15	**7.342E-01**	**7.449E-01**	**7.613E-01**	**7.693E-01**	**7.983E-01**	1.566E-01	2.063E-01	3.379E-01	4.532E-01	6.222E-01
50×20	**5.790E-01**	**5.881E-01**	**5.854E-01**	**5.976E-01**	**6.080E-01**	1.644E-01	2.525E-01	2.614E-01	4.301E-01	5.347E-01
70×5	**6.463E-01**	**6.653E-01**	**6.839E-01**	**6.870E-01**	**7.016E-01**	1.762E-01	2.645E-01	2.569E-01	3.487E-01	5.291E-01
70×10	**5.106E-01**	**5.124E-01**	**5.190E-01**	**5.212E-01**	**5.590E-01**	1.852E-01	1.436E-01	2.296E-01	3.081E-01	4.423E-01
70×15	**4.728E-01**	**4.949E-01**	**5.010E-01**	**5.022E-01**	**5.451E-01**	1.678E-01	1.615E-01	3.050E-01	2.927E-01	3.912E-01
70×20	**3.589E-01**	**3.926E-01**	**4.069E-01**	**4.190E-01**	**4.665E-01**	1.559E-01	1.922E-01	2.851E-01	3.849E-01	4.121E-01
90×5	**3.995E-01**	**4.071E-01**	**4.183E-01**	**4.225E-01**	**4.969E-01**	1.671E-01	2.221E-01	3.030E-01	4.055E-01	3.608E-01
90×10	**5.013E-01**	**5.209E-01**	**5.337E-01**	**5.762E-01**	**5.933E-01**	1.112E-01	2.013E-01	2.556E-01	3.298E-01	4.054E-01
90×15	**4.137E-01**	**4.153E-01**	**4.114E-01**	**4.452E-01**	**4.858E-01**	1.033E-01	1.335E-01	2.283E-01	3.037E-01	4.555E-01
90×20	**4.116E-01**	**4.293E-01**	**4.293E-01**	**4.334E-01**	**4.629E-01**	1.793E-01	1.352E-01	2.633E-01	2.799E-01	3.633E-01
110×5	**5.144E-01**	**5.361E-01**	**5.441E-01**	**5.511E-01**	**5.729E-01**	1.531E-01	3.193E-01	1.802E-01	3.482E-01	4.193E-01
110×10	**4.925E-01**	**5.038E-01**	**5.038E-01**	**5.167E-01**	**5.890E-01**	9.720E-02	1.802E-01	2.778E-01	3.044E-01	3.780E-01
110×15	**2.916E-01**	**3.106E-01**	**3.966E-01**	**3.872E-01**	**4.183E-01**	9.467E-02	1.950E-01	1.941E-01	2.131E-01	2.378E-01
110×20	**2.479E-01**	**2.869E-01**	**2.811E-01**	**2.910E-01**	**3.018E-01**	1.041E-01	1.305E-01	1.161E-01	1.880E-01	2.119E-01
220×20	**3.191E-01**	**3.380E-01**	**3.954E-01**	**4.018E-01**	**4.369E-01**	1.223E-01	1.950E-01	2.010E-01	2.974E-01	3.302E-01
500×20	**3.792E-01**	**3.806E-01**	**4.407E-01**	**4.159E-01**	**4.419E-01**	1.525E-01	2.170E-01	2.481E-01	3.007E-01	3.775E-01
mean	**5.505E-01**	**5.639E-01**	**5.786E-01**	**5.868E-01**	**6.012E-01**	1.520E-01	2.064E-01	2.630E-01	3.623E-01	4.288E-01

TABLE VI
PERFORMANCE COMPARISON OF INSGA, PBEDA, MMSA, MOMA AND REMO ($\beta = 15$)

test sets	INSGA	PBEDA	MMSA	MOMA	REMO
30×5	8.4975E-01	8.5694E-01	8.2252E-01	**8.6408E-01**	8.5185E-01
30×10	8.4583E-01	8.5475E-01	8.6209E-01	8.7151E-01	**8.9938E-01**
30×15	7.9424E-01	8.0552E-01	8.3716E-01	8.1900E-01	**8.7723E-01**
30×20	8.0987E-01	8.0161E-01	8.1630E-01	**8.1880E-01**	8.0757E-01
50×5	7.9030E-01	7.9186E-01	7.9620E-01	**8.0678E-01**	8.0162E-01
50×10	7.4829E-01	7.6892E-01	7.8267E-01	**7.9215E-01**	5.0987E-01
50×15	7.3416E-01	7.4486E-01	7.6131E-01	7.6930E-01	**7.9827E-01**
50×20	5.7895E-01	5.8812E-01	5.8544E-01	5.9762E-01	**6.0804E-01**
70×5	6.4630E-01	6.6532E-01	6.8389E-01	6.8700E-01	**7.0164E-01**
70×10	5.1065E-01	5.1239E-01	5.1905E-01	5.2119E-01	**5.5902E-01**
70×15	4.7277E-01	4.9490E-01	5.0102E-01	5.0222E-01	**5.4513E-01**
70×20	3.5892E-01	3.9264E-01	4.0692E-01	4.1897E-01	**4.6651E-01**
90×5	3.9949E-01	4.0709E-01	4.1830E-01	4.2246E-01	**4.9685E-01**
90×10	5.0127E-01	5.2092E-01	5.3369E-01	5.7624E-01	**5.9329E-01**
90×15	4.1369E-01	4.1528E-01	4.1138E-01	4.4522E-01	**4.8583E-01**
90×20	4.1161E-01	4.2925E-01	4.2928E-01	4.3345E-01	**4.6289E-01**
110×5	5.1442E-01	5.3607E-01	5.4413E-01	5.5106E-01	**5.7286E-01**
110×10	4.9247E-01	5.0377E-01	5.0376E-01	5.1674E-01	**5.8897E-01**
110×15	2.9163E-01	3.1063E-01	3.9664E-01	3.8718E-01	**4.1833E-01**
110×20	2.4785E-01	2.8689E-01	2.8115E-01	2.9103E-01	**3.0182E-01**
220×20	3.1907E-01	3.3798E-01	3.9539E-01	4.0178E-01	**4.3693E-01**
500×20	3.7917E-01	3.8063E-01	4.4070E-01	4.1591E-01	**4.4195E-01**
mean	5.5049E-01	5.6392E-01	5.7859E-01	5.8680E-01	**6.0117E-01**

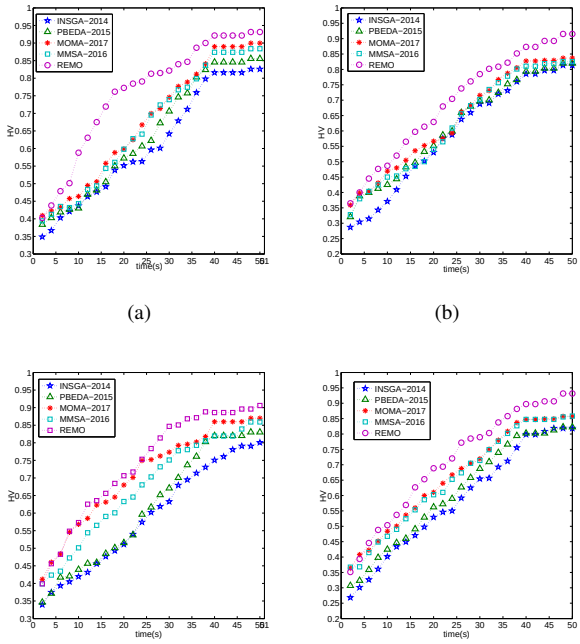

Fig. 6. Changes of HV w.r.t. evolution

resulting from machine breakdowns, and reduce the difference in completion time before and after machine breakdowns.

4) Analysis of convergence curves: In this subsection, we compare the convergence profiles of five algorithms, including INSGA, PBEDA, MMSA, MOMA and REMO, when β is equal to 15. We run these algorithms with a CPU time of 50 seconds on the aforementioned PC. Several typical convergence curves, denoted by the change of HV over time, on instances Ta52, Ta74, Ta96 and Ta158 obtained by INSGA, PBEDA, MMSA, MOMA and REMO, respectively, are plotted in Fig. 6. From Fig. 6., we can see that the convergence of the proposed algorithm is the fastest among the five algorithms in the four compared instances. It is attributed to the fact that, the proposed algoirthm explicitly take advantage of information from non-dominated solutions in generating, whereas the others do not.

To summarize, the proposed algorithm has much better or competitive performance compared with four representative existing algorithms for solving multi-objective BLSFS scheduling problem with machine breakdowns, which may be attributed to the fact that the proposed crossover operators can make full use of information in the non-dominated solutions and that the proposed rescheduling strategy is able to reduce the difference in the objectives before and after machine breakdowns.

VI. Conclusions

In this study, the BLSFS scheduling problem with machine breakdowns is investigated, first by formulating it as a multi-objective optimization problem, and then solved using a multi-objective evolutionary algorithm whose population is initialized with solutions obtained by a single-objective heuristic algorithm. Two new crossover operators are developed that can make use of non-dominated solutions, which are combined with two mutation operators to achieve a balance between exploration and exploitation. Last but not the least, a rescheduling strategy is proposed that can further reduce the influence of machine breakdowns.

The performance of the proposed algorithm is evaluated on 22 test sets of the BLSFS scheduling problem, and compared with four state-of-the-art algorithms proposed for solving multi-objetive problems without machine breakdowns. The experimental results demonstrate the superiority of the proposed algorithm in terms of robustness and stability of the approximated non-dominated solutions. The outperforming of the proposed algorithm may be attributed to the new crossover operators as well as the rescheduling strategy.

There are several opportunities for future research on BLSFS scheduling problems with machine breakdowns. First, we assume in this study that all machines have the same number of machine breakdowns, which may be unrealistic. Thus, more practical machine breakdowns can be considered in future. Second, in this study, one of the two mutation operators is randomly selected to generate offspring. It might be desirable to develop a self-adaptive mechanism for selecting one of the mutation operators to improve the exploration capability. Third, the computational complexity of the rescheduling strategy may further be reduced. Finally, other types of uncertainties, such as non-deterministic processing times, operator illness and due-date changes can also be considered.

Acknowledgments

This work was supported in part by the National Natural Science Foundation of China with grant no. 61375067, 61473299, 71533001, 61374043, 61403155 and 71533001, the Joint Research Fund for Overseas Chinese, Hong Kong and Macao Scholars of the National Natural Science Foundation of China with grant no. 61428302, Program for New Century Excellent Talents in University with grant No. NCET-13-0106, Specialized Research Fund for the Doctoral Program of Higher Education with grant no. 20130042110035, and Gansu Province Basic Research Innovation Group Project with grant no. 1506RJIA031.

References

[1] Y.-Y. Han, D.-w. Gong, X.-Y. Sun, and Q.-K. Pan, "An improved nsga-ii algorithm for multi-objective lot-streaming flow shop scheduling problem," *International Journal of Production Research*, vol. 52, no. 8, pp. 2211–2231, 2014.

[2] Q.-k. Pan, L. Wang, H.-y. Sang, J.-q. Li, and M. Liu, "A high performing memetic algorithm for the flowshop scheduling problem with blocking," *Automation Science and Engineering, IEEE Transactions on*, vol. 10, no. 3, pp. 741–756, 2013.

[3] A. Boonmee and K. Sethanan, "A glnpso for multi-level capacitated lot-sizing and scheduling problem in the poultry industry," *European Journal of Operational Research*, vol. 250, no. 2, pp. 652–665, 2016.

[4] C. Gahm, F. Denz, M. Dirr, and A. Tuma, "Energy-efficient scheduling in manufacturing companies: A review and research framework," *European Journal of Operational Research*, vol. 248, no. 3, pp. 744–757, 2016.

[5] Q.-K. Pan, L. Wang, K. Mao, J.-H. Zhao, and M. Zhang, "An effective artificial bee colony algorithm for a real-world hybrid flowshop problem in steelmaking process," *Automation Science and Engineering, IEEE Transactions on*, vol. 10, no. 2, pp. 307–322, 2013.

[6] M. Gholami, M. Zandieh, and A. Alem-Tabriz, "Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns," *The International Journal of Advanced Manufacturing Technology*, vol. 42, no. 1-2, pp. 189–201, 2009.

[7] S. Goren and I. Sabuncuoglu, "Robustness and stability measures for scheduling: single-machine environment," *IIE Transactions*, vol. 40, no. 1, pp. 66–83, 2008.

[8] K. Katragjini, E. Vallada, and R. Ruiz, "Flow shop rescheduling under different types of disruption," *International Journal of Production Research*, vol. 51, no. 3, pp. 780–797, 2013.

[9] K. Wang and S. Choi, "A holonic approach to flexible flow shop scheduling under stochastic processing times," *Computers & Operations Research*, vol. 43, pp. 157–168, 2014.

[10] Y. N. Sotskov, T.-C. Lai, and F. Werner, "Measures of problem uncertainty for scheduling with interval processing times," *OR spectrum*, vol. 35, no. 3, pp. 659–689, 2013.

[11] W.-C. Yeh, P.-J. Lai, W.-C. Lee, and M.-C. Chuang, "Parallel-machine scheduling to minimize makespan with fuzzy processing times and learning effects," *Information Sciences*, vol. 269, pp. 142–158, 2014.

[12] L. Liu, H.-y. Gu, and Y.-g. Xi, "Robust and stable scheduling of a single machine with random machine breakdowns," *The International Journal of Advanced Manufacturing Technology*, vol. 31, no. 7-8, pp. 645–654, 2007.

[13] T. Chaari, S. Chaabane, T. Loukil, and D. Trentesaux, "A genetic algorithm for robust hybrid flow shop scheduling," *International Journal of Computer Integrated Manufacturing*, vol. 24, no. 9, pp. 821–833, 2011.

[14] J. Xiong, L.-n. Xing, and Y.-w. Chen, "Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns," *International Journal of Production Economics*, vol. 141, no. 1, pp. 112–126, 2013.

[15] C.-H. Liu and D.-H. Huang, "Reduction of power consumption and carbon footprints by applying multi-objective optimisation via genetic algorithms," *International Journal of Production Research*, vol. 52, no. 2, pp. 337–352, 2014.

[16] D. Rahmani and M. Heydari, "Robust and stable flow shop scheduling with unexpected arrivals of new jobs and uncertain processing times," *Journal of Manufacturing Systems*, vol. 33, no. 1, pp. 84–92, 2014.

[17] K. Wang and S. Choi, "A decomposition-based approach to flexible flow shop scheduling under machine breakdown," *International Journal of Production Research*, vol. 50, no. 1, pp. 215–234, 2012.

[18] C. Liao and W. Chen, "Scheduling under machine breakdown in a continuous process industry," *Computers & Operations Research*, vol. 31, no. 3, pp. 415–428, 2004.

[19] K. Wang, Y. Huang, and H. Qin, "A fuzzy logic-based hybrid estimation of distribution algorithm for distributed permutation flowshop scheduling problems under machine breakdown," *Journal of the Operational Research Society*, vol. 67, no. 1, pp. 68–82, 2016.

[20] S.-H. Yoon and J. A. Ventura, "An application of genetic algorithms to lot-streaming flow shop scheduling," *IIE Transactions*, vol. 34, no. 9, pp. 779–787, 2002.

[21] E. Vallada, R. Ruiz, and J. M. Framinan, "New hard benchmark for flowshop scheduling problems minimising makespan," *European Journal of Operational Research*, vol. 240, no. 3, pp. 666–677, 2015.

[22] M. T. Jensen, "Generating robust and flexible job shop schedules using genetic algorithms," *Evolutionary Computation, IEEE Transactions on*, vol. 7, no. 3, pp. 275–288, 2003.

[23] M. Nawaz, E. E. Enscore Jr, and I. Ham, "A heuristic algorithm for the m-machine,n-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[24] R. Ruiz, C. Maroto, and J. Alcaraz, "Two new robust genetic algorithms for the flowshop scheduling problem," *Omega*, vol. 34, no. 5, pp. 461–476, 2006.

[25] L. Davis, "Applying adaptive algorithms to epistatic domains." in *IJCAI*, vol. 85, 1985, pp. 162–164.

[26] Z. Michalewicz, *Genetic algorithms+ data structures= evolution programs*. springer, 1996.

[27] D. C. Mattfeld, *Evolutionary search and the job shop investigations on genetic algorithms for production scheduling*. Physica-Verlag, 1996.

[28] Y. Zhang, X. Li, and Q. Wang, "Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization," *European Journal of Operational Research*, vol. 196, no. 3, pp. 869–876, 2009.
[29] Q. K. Pan, M. F. Tasgetiren, and Y. C. Liang, "A discrete differential evolution algorithm for the permutation flowshop scheduling problem," *Computers & Industrial Engineering*, vol. 55, no. 4, pp. 795–816, 2008.
[30] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 2, pp. 182–197, 2002.
[31] A. Tiwari, P.-C. Chang, M. Tiwari, and N. J. Kollanoor, "A pareto block-based estimation and distribution algorithm for multi-objective permutation flow shop scheduling problem," *International Journal of Production Research*, vol. 53, no. 3, pp. 793–834, 2015.
[32] X. Li and S. Ma, "Multi-objective memetic search algorithm for multi-objective permutation flow shop scheduling problem," *IEEE Access*, vol. 4, pp. 2154–2165, 2016.
[33] X. Wang and L. Tang, "A machine-learning based memetic algorithm for the multi-objective permutation flowshop scheduling problem," *Computers & Operations Research*, vol. 79, pp. 60–77, 2017.
[34] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. Da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *Evolutionary Computation, IEEE Transactions on*, vol. 7, no. 2, pp. 117–132, 2003.