Engineering Applications of Artificial Intelligence 37 (2015) 279–292

Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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

A discrete teaching-learning-based optimisation algorithm for realistic flowshop rescheduling problems

Jun-qing Li [a,b], Quan-ke Pan [c,*], Kun Mao [a]

[a] State Key Laboratory of Synthetic Automation for Process Industries, Northeastern University, ShenYang 110819, PR China
[b] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[c] State Key Lab of Digital Manufacturing Equipment & Technology in Huazhong University of Science & Technology, Wuhan, 430074, P. R. China

ARTICLE INFO

Article history:
Received 8 January 2014
Received in revised form
11 June 2014
Accepted 23 September 2014

Keywords:
Flowshop problem
Multi-objective
Teaching-learning-based optimisation
Rescheduling

ABSTRACT

In this study, we proposed a discrete teaching-learning-based optimisation (DTLBO) for solving the flowshop rescheduling problem. Five types of disruption events, namely machine breakdown, new job arrival, cancellation of jobs, job processing variation and job release variation, are considered simultaneously. The proposed algorithm aims to minimise two objectives, i.e., the maximal completion time and the instability performance. Four discretisation operators are developed for the teaching phase and learning phase to enable the TLBO algorithm to solve rescheduling problems. In addition, a modified iterated greedy (IG)-based local search is embedded to enhance the searching ability of the proposed algorithm. Furthermore, four types of DTLBO algorithms are developed to make detailed comparisons with different parameters. Experimental comparisons on 90 realistic flowshop rescheduling instances with other efficient algorithms indicate that the proposed algorithm is competitive in terms of its searching quality, robustness, and efficiency.

1. Introduction

The flowshop scheduling problem (FSP) can be found in many realistic production horizons, such as chemical, textile, metallurgical, printed circuit board, automobile, and iron and steel making manufacture (Garey et al., 1976; Nawaz et al., 1983; Ruiz et al., 2006; Pan et al., 2008). The FSP has been verified to be an NP-hard problem (Garey et al., 1976; Nawaz et al., 1983), and many researchers have applied heuristics and meta-heuristics for solving FSP, such as genetic algorithm (GA) (Ruiz et al., 2006), and particle swarm optimisation (PSO) (Pan et al., 2008). Most of the literature assumes that all of the machines are continuously available, and all of the jobs are deterministic during the production horizon. However, in a realistic environment, many disruptions, such as a machine breakdown, new job arrival, cancellation of jobs, job processing variation, and job release variation, can make it impossible to apply the traditional scheduling approach to an industrial horizon. In recent years, more and more researchers have focused on rescheduling problems, in other words, on considering disruptions in the scheduling process. Vieira et al. gave a framework of strategies, policies, and methods for the rescheduling of manufacturing systems (Vieira et al., 2003). Ouelhadj and Petrovic provided a review of the state of the art of dynamic scheduling for different methods, such as heuristics, meta-heuristics, multi-agent systems, and other artificial intelligence techniques (Ouelhadj and Petrovic, 2009).

Studies that have considered random machine breakdowns are the following: Allahverdi and Mittenthal addressed the problem of minimising the makespan in a two-machine flowshop when the machines were subject to random breakdowns (Allahverdi and Mittenthal, 1995). Schmidt analysed the deterministic scheduling problems when the machines were not continuously available for processing (Schmidt, 2000), while Zandieh and Gholami proposed an immune algorithm for scheduling a hybrid flow shop (HFS) with sequence-dependent setup times and machines with random breakdowns (Zandieh and Gholami, 2009). Very recently, Wang and Choi presented a decomposition-based approach to solve flexible flow shop (FFS) scheduling under random machine breakdown (Wang and Choi, 2012). Mirabi et al. investigated the random machine breakdown in a two-stage HFS context (Mirabi et al., 2013). Xiong et al. solved the machine breakdown event in flexible job shop scheduling problems (FJSP) (Xiong et al., 2013). To solve new job arrival events, the following studies were performed: Hosseini and Tavakkoli-Moghaddam proposed a hybrid algorithm that combined GA and simulated annealing (SA) to solve a two-machine flowshop, where each job has a time window and can arrive in its time window randomly (Hosseini and Tavakkoli-Moghaddam 2013). Rahmani

* Corresponding authors at: State Key Lab of Digital Manufacturing Equipment & Technology in Huazhong University of Science & Technology, Wuhan, 430074, P. R. China
E-mail addresses: lijunqing.cn@gmail.com (J.-q. Li), panquanke@mail.neu.edu.cn (Q.-k. Pan).

http://dx.doi.org/10.1016/j.engappai.2014.09.015
0952-1976/

and Heydari considered a two-machine flowshop with uncertain processing times and unexpected arrivals of new jobs (Rahmani and Heydari, 2014). Rahmani et al. considered how to achieve a stable schedule despite the arrival of new unpredicted jobs into the process (Rahmani et al., 2013). Studies that address the cancellation of jobs are the following: Madureira et al. presented an evolutionary computing algorithm for dynamic scheduling problems that have random cancellation of jobs (Madureira et al., 2004). Silva et al. investigated a dynamic environment in which the orders are cancelled during the scheduling process by using GAs and ant colony optimisation (ACO) algorithms (Silva et al., 2008). To address job processing time variation, Wang et al. developed a hybrid algorithm with GAs and a hypothesis-test method to solve the stochastic flow shop scheduling problem (Wang et al., 2005). Wang and Choi proposed a decomposition-based holonic approach for minimising the makespan of an FFS that has stochastic processing times (Wang and Choi, 2014). For job release time variations, Nilsson et al. discussed the modelling and analysis of real-time systems that are subject to random time delays in a communications network (Nilsson et al., 1998). Tang et al. proposed a neural network model and algorithm to solve the dynamic hybrid flow shop scheduling problem (Tang et al., 2005). However, most existing work addresses these disruptions only independently. Katragjini et al. studied rescheduling problems with consideration of simultaneous disruptions under flowshop scheduling problems (Katragjini et al., 2013). Rahmani et al. investigated the rescheduling problem in flexible flow shop problems, where three types of disruptions are considered, i.e., new job arrivals, machine breakdowns, and job processing time variations (Rahmani et al., 2013).

Very recently, by mimicking the teaching-learning process, a novel population-based optimisation method called a teaching-learning-based optimisation (TLBO) algorithm was proposed by Rao et al. (2012), Rao and Patel (2013), Rao and Kalyankar (2013) and Rao and Patel (2013). TLBO includes two vital components, which are the teacher and the learners. The whole algorithm evolves through two main phases, namely the teaching phase and the learning phase. In the teaching phase, each learner improves its status by using the difference between the teacher and the mean result of the current population. During the learning phase, each learner studies more knowledge from another learner in the population. Through two types of learning, a learner can evolve to find a near-optimal solution. It has been verified that TLBO is competitive with other meta-heuristics, such as GA, ACO, PSO, and artificial bee colony (ABC) (Rao et al., 2012; Rao and Patel, 2013; Rao and Kalyankar, 2013; Rao and Patel, 2013).

In this study, to solve the flowshop rescheduling problems, we proposed a discrete version of TLBO. Five types of disruptions are considered simultaneously in this work, i.e., the machine breakdown, new job arrival, cancellation of jobs, job processing variation, and job release variation. A bi-objective performance measure that consists of makespan and instability is also used, as given in reference (Katragjini et al., 2013). The motivation of using makespan as a performance measure is as follows: (1) the makespan criterion is commonly used as the efficiency performance measurement in the rescheduling problem, such as (Katragjini et al., 2013; Nof and HANK GRANT, 1991; Liu and Zhou, 2013; Tang et al., 2014); (2) the makespan criterion is considered in reference (Katragjini et al., 2013) with the same instances, and to make a detailed comparison with the IG algorithm in (Katragjini et al., 2013), we also use the makespan as a performance measurement; (3) in this study, the latest job, which arrives in a rush order, does not have to be placed at the end of the sequence; thus, the latest job does not necessarily determine the makespan because this problem is not a single machine problem. The instability objective is measured by the number of tasks whose starting times have been altered in the new schedule.

The remainder of this paper is organised as follows: Section 2 briefly describes the problem. Next, the canonical TLBO is presented in Section 3. Section 4 reports the discretisation of TLBO, while Section 5 gives the framework of the proposed algorithm. Section 6 illustrates the experimental results and compares them to the current performing algorithms from the literature to demonstrate the superiority of the proposed algorithm. Finally, the last section gives the concluding remarks and future research directions.

2. Problem definition

In this study, similar to in reference (Katragjini et al., 2013), we consider the rescheduling problem of a permutation flowshop that is subject to random disruption events. In the considered problem, we have a set of N ($i=1,2,\ldots,n$) jobs to be processed on a set of M ($j=1,2,\ldots,m$) machines. Each job should access each machine following the same processing order, i.e., from the first machine to the last machine. Hereafter, we call the static schedule without considering any disruption an on-going schedule and the schedule after considering the disruption event a new schedule. For simplicity, we give the following assumptions:

- For any machine breakdown event, we assume that the breakdown time and duration are not known a priori, and the job that is preempted due to the machine breakdown resumes its processing from the point at which the interruption occurred.
- Machine breakdowns occur only on busy machines. Three situations under machine breakdown disruptions must be considered for the breakdown machine, i.e., the first situation for operations that have a completion time of less than the disruption time point t, the second situation for operations that have a completion time that overlaps with the machine breakdown disruption, and the third situation for operations that have a starting time that is later than the disruption time point t.
- When a new job arrives in the system, reactive procedures are prompted to introduce the new job to the first stage in the system.
- All of the jobs that are interrupted with any disruption should remain with their assigned machine, and the subsequent work should be delayed or postponed if necessary.
- All of the jobs that have started their first operations when the disruption occurs should retain their scheduling orders.
- There are infinite buffers between any two stages.
- Preemption is not allowed; in other words, each job should wait for the completion of the predecessor job on the same machine.
- The overlap between consecutive operations of the same job is not allowed; in other words, the following operation cannot be started until the completion of the predecessor operation of the same job.

2.1. Notation

The following notation is used to formulate the mathematical model for the flowshop rescheduling problem that considers the five types of disruptions.

Data or fixed parameters

$p_{i,j}$	The processing time for operation O_{ij} in an on-going schedule.
$s_{i,j}$	The starting time of operation O_{ij} in an on-going schedule.
$c_{i,j}$	The completion time of operation O_{ij} in an on-going schedule.
$r_{i,j}$	The release time of operation O_{ij}.
B_s^j	The starting time point of the machine breakdown disruption on machine j.

B_e^j The recovery time point of the machine breakdown disruption on machine j.
w_1 The penalty coefficient.
M Sufficiently large constant.

Decision variables

$\overline{s_{i,j}}$ The starting time of operation O_{ij} in the new schedule.
$\overline{c_{i,j}}$ The completion time of operation O_{ij} in the new schedule.
$\overline{p_{i,j}}$ The processing time of operation O_{ij} in the new schedule.
x_{ij} A 0/1 variable that is equal to one if and only if operation O_{ij} has a different starting time in the on-going and new schedules.
$y_{i,j,1}$ A 0/1 variable that is equal to one if and only if $\overline{s_{i,j}}+\overline{p_{i,j}} \le B_s^j$.
$y_{i,j,2}$ A 0/1 variable that is equal to one if and only if $\overline{s_{i,j}} < B_s^j$ and $\overline{s_{i,j}}+\overline{p_{i,j}} > B_s^j$.
$y_{i,j,3}$ A 0/1 variable that is equal to one if and only if $B_s^j \le \overline{s_{i,j}}$.

2.2. Mathematical model

With the variables defined above, the mathematical model for the flowshop rescheduling problem is given as follows:

$$f = w_1 * f_1 + (1-w_1) * f_2 \tag{1}$$

$$f_1 = \min\left\{\max_{1\le i\le n} \overline{c_{i,m}}\right\} \tag{2}$$

$$f_2 = \min\left\{\sum_{i=1}^{n}\sum_{j=1}^{m} x_{ij}\right\} \tag{3}$$

s.t.

$$\overline{c_{i,j}} - \overline{s_{i,j}} - \overline{p_{i,j}} + (1-y_{i,j,1})M \ge 0 \tag{4}$$

$$\overline{c_{i,j}} - \overline{s_{i,j}} - \overline{p_{i,j}} - (1-y_{i,j,1})M \le 0 \tag{5}$$

$$\overline{c_{i,j}} - \overline{s_{i,j}} - \overline{p_{i,j}} - B_e^j + B_s^j + (1-y_{i,j,2})M \ge 0 \tag{6}$$

$$\overline{c_{i,j}} - \overline{s_{i,j}} - \overline{p_{i,j}} - B_e^j + B_s^j - (1-y_{i,j,2})M \le 0 \tag{7}$$

$$\overline{c_{i,j}} - \max\{\overline{s_{i,j}}, B_e^j\} - \overline{p_{i,j}} + (1-y_{i,j,3})M \ge 0 \tag{8}$$

$$\overline{c_{i,j}} - \max\{\overline{s_{i,j}}, B_e^j\} - \overline{p_{i,j}} - (1-y_{i,j,3})M \le 0 \tag{9}$$

$$\sum_{k=1}^{3} y_{i,j,k} = 1, i \in \{1,2,\ldots,n\}; j \in \{1,2,\ldots,m\}; k \in \{1,2,3\} \tag{10}$$

$$\overline{c_{i_1,j}} - \overline{c_{i_2,j}} - \overline{p_{i_1,j}} \ge 0 \text{ or } \overline{c_{i_2,j}} - \overline{c_{i_1,j}} - \overline{p_{i_2,j}} \ge 0, \forall i_1 \ne i_2 \tag{11}$$

$$\overline{s_{i,j}} \ge r_{i,j}, i \in \{1,2,\ldots,n\}; j \in \{1,2,\ldots,m\} \tag{12}$$

$$\overline{s_{i+1,j}} \ge \overline{c_{i,j}}, i \in \{1,2,\ldots,n-1\}; j \in \{1,2,\ldots,m\} \tag{13}$$

$$x_{ij} = \{0,1\}, i \in \{1,2,\ldots,n\}; j \in \{1,2,\ldots,m\} \tag{14}$$

$$y_{ijk} = \{0,1\}, i \in \{1,2,\ldots,n\}; j \in \{1,2,\ldots,m\}; k \in \{1,2,3\} \tag{15}$$

In this mathematical model, the total weighted objective and the makespan objective are given in Eqs. (1) and (2), respectively. Eq. (3) illustrates the second objective of the problem, i.e., the instability objective, which is used to minimise the number of tasks whose starting times have been altered in the new schedule. Three situations under the machine breakdown disruption are guaranteed by Constraints (4) to (9). The first situation that is guaranteed by Constraints (4) and (5) is for the operations that have completed their tasks before the machine breakdown. Constraints (6) and (7) guarantee the second situation, which is for the operations that overlap with the machine breakdown, while Constraints (8) and (9) guarantee the third situation, in which the operations begin their work after the machine breakdown disruption. Constraint (10) shows that only one case can occur in the problem that is under study. Constraint (11) shows that multiple jobs cannot be processed on the same machine at the same time. Constraint (12) implies that a job cannot be started until its release event occurs. In Constraint (13), the operation sequence is realised for the same job; in other words, the following operation cannot be started until the completion of the predecessor operation of the same job. Constraints (14) and (15) define the value ranges for the decision variables.

3. The canonical TLBO

TLBO mimics the learning process of a teacher and a population of learners. In the canonical TLBO, several learners construct a population of activities. The best learner of the current population is selected as the teacher. All of the learners will perform two evolving phases, i.e., the teaching phase and the learning phase.

3.1. Teaching phase

In the canonical TLBO, the first part of the algorithm is the teaching phase, where the learners learn through the teacher by using the difference between the teacher and the mean result of the current population. In a problem with n dimension, at any iteration t, let X_t^i (i=1,2,…,m) be the ith learner in the population, let $M_{t,j}$ be the mean result of the learners in a particular subject j (j=1,2,…,n), and let X_t^b be the best learner in the current population, which is also selected as the current teacher. The detailed implementation of the teaching phase is given as follows:

Step 1. Compute the difference $D_{t,j}$
Let $D_{t,j}$ be the difference between the teacher X_t^b and the mean result $M_{t,j}$. Formula (16) gives the computation process of $D_{t,j}$.

$$D_{t,j} = r_t(X_{t,j}^b - T_F M_{t,j}) \tag{16}$$

where $X_{t,j}^b$ is the result of the teacher in subject j, r_t is a random number in the range [0,1], and $T_F$is the teaching factor that decides the value of the mean to be changed. The value of T_F can be either 1 or 2, which is again a heuristic step and is decided randomly with equal probability (Rao et al., 2012).
Step 2. Generate a new learner
Learner X_t^i updates its state in subject j by combining the difference $D_{t,j}$ and its current state:

$$X_{t,j}^{i'} = X_{t,j}^i + D_{t,j} \tag{17}$$

where $X_{t,j}^{i'}$ is the updated value of the current i^{th} learner X_t^i in subject j.
Step 3. If $X_t^{i'}$ is better than X_t^i, then replace the latter by the new value.

3.2. Learning phase

In the learning phase, several learners evolve through learning from each other. The main steps of the learning phase are as follows:

Step 1. For the learner X_t^i, randomly select another learner X_t^k.
Step 2. If X_t^i is better than X_t^k, then let

$$X_t^{i'} = X_t^i + r_i(X_t^i - X_t^k) \tag{18}$$

Step 3. If X_t^k is better than X_t^i, then let

$$X_t^{i\prime} = X_t^i + r_i(X_t^k - X_t^i) \quad (19)$$

Step 4. If $X_t^{i'}$ is better than X_t^i, then replace the latter by the new value.

4. Discretisation of TLBO

From the detailed descriptions of the canonical TLBO, we can see that the difference function is used in both the teaching and learning phases. However, the difference function can be directly applied only when solving continuous optimisation problems. In other words, when solving the discrete optimisation problems, we must develop a discrete version of TLBO. To make TLBO adapt to the flowshop rescheduling problem, we develop two types of teaching phase heuristics, which are called TP-I and TP-II, respectively, and we also implement two types of learning phase heuristics, which are called LP-I and LP-II, respectively. The detailed steps of the above heuristics are given as follows:

4.1. Teaching phase TP-I

We first implement the function TP-I with an idea that directly arises from the canonical TLBO. The detailed implementation is given as follows:

4.1.1. Discretisation of M_t

For the mean result of the learners at iteration t, we develop a heuristic as follows:

Step 1. For each subject j (j=1,2,…,n), compute the sum of each learner in the population. The result is denoted as $S_t = \{S_{t,1}, S_{t,2}..., S_{t,n}\}$.
Step 2. Uniformly spread each element of S_t into the range of [1,n] by using the following formula: $S_{t,j}' = mod(S_{t,j} - 1, n) + 1$, where the function $mod(x, y)$ is used to find the remainder of the division of x by y, $S_{t,j}'$ is the uniform value and j=1,2,…,n.

For example, we are given three learners, e.g., {1, 2, 3, 4, 5, 6}, {6, 3, 2, 4, 5, 1}, and {2, 3, 4, 6, 5, 1}, where n=6. The sum value S_t={9, 8, 10, 14, 15, 8}, and the uniform result is {3, 2, 4, 2, 3, 2}.

4.1.2. Discretisation of D_t

For the difference between the teacher and the mean result at iteration t, we use formula (16) in the canonical TLBO.

For the above example, suppose that {1, 2, 3, 4, 5, 6} is the teacher, which is the best of the three learners. Let T_F=1, and r_t=0.5; then, D_t={-1.5, -0.5, -1, 0.5, 0.5, 1.5}

4.1.3. Generate a new unfeasible learner

For each learner X_t^i in the population, we first use formula (17) to generate a new learner X_t^i; second, for the newly-generated learner X_t^i, we transfer each element into an integer by taking the integer part of each element. It should be noted that the newly-generated learner X_t^i might not be a feasible solution because of the repeated elements. Therefore, it must go through the following steps to make it feasible:

4.1.4. Repair the newly-generated learner

The repair function is implemented as follows:

Step 1. For each repeated element in $X_t^{i'}$, only the first occurrence remains, and we delete all of the following occurrences.
Step 2. For each blank position in $X_t^{i'}$, find the unassigned job that has the largest number of occurrences in the population. In other words, for each subject j, count the number of occurrences of each job from all learners in the population, and then find the job that has the largest number of occurrences. If this criterion is not satisfied, then skip this position.
Step 3. For the remaining blank position, we randomly select an unassigned job to fill with it.

In the above example, for the learner {6, 3, 2, 4, 5, 1}, the update vector is {4.5, 2.5, 1, 4.5, 5.5, 2.5}. Then, the resulting integer vector is {4, 2, 1, 4, 5, 2}, which corresponds to the newly-generated unfeasible learner. Because there are repeated elements in the vector, an unfeasible learner should go through the repair phase. In step 1, the remaining vector is {4, 2, 1, -1, 5, -1}, where '-1' represents the blank position. In step 2, the vector becomes {4, 2, 1, 6, 5, -1}. Then, in the last step, the final vector is {4, 2, 1, 6, 5, 3}. It should be noted that the computational time complexity of the repair function that is discussed above is $O(nm)$.

4.1.5. Update the current learner

Evaluate the newly-generated learner, and update X_t^i by $X_t^{i'}$ if the latter is better than the former.

4.2. Teaching phase TP-II

The first type of teaching phase heuristic TP-I arises directly from the canonical TLBO, which goes through five procedures and should undergo a translation from a real number to an integer number. The main computational time is consumed in the repair process and has the computational time complexity of $O(nm)$. However, TP-I has the following disadvantages: (1) in learning from the difference between the teacher and the mean result, several good characteristics of the current learner could be lost; and (2) in the repair function, the randomly selected job could produce a solution that has a lower performance.

In TP-II, we embed the IG heuristic in the TLBO to enhance the searching ability of the algorithm. The detailed implementation is given as follows:

4.2.1. Discretisation of M_t

Instead of computing the mean result of the learners at iteration t, we randomly select a learner in the current population to be the mean result M_t. The main idea behind this method is that, in the current population, any learner can learn from the teaching process between the teacher and a random learner rather than the mean result (which is used in the canonical TLBO). Compared with the method that is discussed in sub-Section 4.1.1, this heuristic has the following advantages: (1) computational time is saved; and (2) the teaching information comes from a randomly selected learner, which can enhance the exploration capability of the algorithm.

4.2.2. Discretisation of D_t

The difference D_t between the teacher X_t^b and the randomly selected learner $X_t^r$is computed as follows: (1) first, initialise an empty vector, which is called D_t; (2) second, for each subject j, if $X_{t,j}^b = X_{t,j}^r$, then skip it; otherwise, record j into the vector D_t. Therefore, the resulting vector D_t represents the different element positions between X_t^b and X_t^r. For example, given X_t^b={6, 3, 2, 4, 5, 1}, and X_t^r={1, 2, 3, 4, 5, 6}, then D_t={1, 2, 3, 6}, which implies that four elements, i.e., subjects 1, 2, 3, and 6 in X_t^r are different from the teacher.

Procedure *MIG_localSearch*

Input: the difference vector D_t, the destruction length d, and the current learner X_t^i

Output: a feasible learner

Begin

Step 1. Divide the jobs in X_t^i into two groups based on the disruption event time t, where group1 contains the jobs that cannot be rescheduled and group2 includes the jobs that can be rescheduled.

Step 2. Destruction stage

Step 2.1 If $d \geq len(D_t)$, where $len(D_t)$ is the length of D_t, we then perform the following two steps: first, randomly select d elements in the difference vector D_t; second, store the selected jobs into a vector that is called D_d; otherwise, store the elements in D_t into D_d.

Step 2.2 For each element L_k in D_d, delete the job at position L_k in X_t^i, and memorise the deleted jobs in a new vector called X_t^d. Then, there are two vectors, i.e., $\overline{X_t^i}$, which contains the remaining jobs of X_t^i, and X_t^d, which includes the deleted jobs from X_t^i, where $len(\overline{X_t^i})+len(X_t^d)=n$.

Step 3. Construction stage

Step 3.1 For each job J_k in X_t^d, perform the following steps until all of the elements in X_t^d have been processed.

Step 3.2 If $len(\overline{X_t^i})>J_{max}$, then randomly select J_{max} candidate positions from $\overline{X_t^i}$; otherwise, all of the positions in $\overline{X_t^i}$ are candidate positions.

Step 3.3 For each candidate position i in $\overline{X_t^i}$, insert J_k at i, and evaluate the newly-generated partial solution.

Step 3.4 Obtain the optimal position L_b for the job J_k with the minimum objective value.

Step 3.5 Insert J_k at position L_b of $\overline{X_t^i}$, and go back to step 3.1.

Step 4. Update the current learner

Step 4.1 If the newly-generated learner $\overline{X_t^i}$ is better than the current learner X_t^i, then replace the current learner with $\overline{X_t^i}$.

Step 4.2 If the newly-generated learner is not better than the current learner then ignore it.

End

Fig. 1. Pseudo code of the function *MIG_localSearch*.

4.2.3. IG-based local search

To improve the searching ability of the algorithm, we developed a modified integrated greedy (IG) function that is called *MIG_localSearch*. In contrast with the canonical IG heuristic in (Ruiz and Stützle, 2007), in the destruction stage of *MIG_localSearch*, the jobs that are erased from the current solution are selected based on the difference vector rather than on a random method, as in (Ruiz and Stützle, 2007). Furthermore, to solve the rescheduling problem at a large scale, the maximal number of candidate positions for an erased job in the construction stage is limited to J_{max}. In other words, we limit the number of possible insert positions to decrease the computational complexity of the proposed algorithm. The detailed implementation of *MIG_localSearch* is given in Fig. 1.

4.2.4. Generate a new learner

To generate a new learner during the teaching phase, we perform the proposed *MIG_localSearch* function by using the two parameters, i.e., the current learner X_t^i and the difference vector D_t. After applying the *MIG_localSearch* function, the new learner is generated, and the teaching phase is completed.

4.2.5. A simple example using MIG_localSearch

Here, we give a simple example to illustrate the procedure of the proposed *MIG_localSearch* function. We are given a 10-job and 5-machine flowshop rescheduling problem, two learners, $X_t^i=\{1, 2, 3, 4, 5, 8, 6, 7, 9, 10\}$ and $X_t^r=\{1, 2, 3, 4, 6, 8, 10, 9, 7, 5\}$, and the teacher $X_t^b=\{1, 2, 3, 4, 5, 7, 8, 9, 10, 6\}$. Suppose that the first four jobs are divided into the first group by the disruption event time. Before applying the *MIG_localSearch* function, we first compute the difference between X_2 and the teacher. The resulting difference vector is $D_t=\{5, 6, 7, 9, 10\}$. Then, we obtain the input parameters for the *MIG_localSearch* as follows: the current learner is X_t^i, the destruction length is $d=3$, and the difference vector is $D_t=\{5, 6, 7, 9, 10\}$.

In the destruction stage, we first randomly select three job positions from D_t, for example, let $D_d=\{5, 7, 9\}$. Then, the current learner should delete the jobs at positions 5, 7, and 9. Therefore, the resulting learner is $\overline{X_t^i}=\{1, 2, 3, 4, –, 8, –, 7, –, 10\}$, and $X_t^d=\{5, 6, 9\}$. The detailed destruction process can be seen in Fig. 2 (a).

In the construction stage, the jobs J_5, J_7, and J_9 are inserted into $\overline{X_t^i}$ one by one. The optimal position for each newly-inserted job is the position that minimises the objective values of the partial solution. The detailed illustration of the construction stage can be seen in Fig. 2 (b).

4.3. Learning phase-I

For the learner X_t^i, we randomly select another learner X_t^k, and then, we apply the first three steps, as discussed in Section 3.2. Then, the newly-generated solution must go through the repair function that is discussed in Section 4.1.4 to make it feasible.

Given three learners, {1, 2, 3, 4, 5, 6}, {6, 3, 2, 4, 5, 1}, and {2, 3, 4, 6, 5, 1}, suppose that {1, 2, 3, 4, 5, 6} and {6, 3, 2, 4, 5, 1} are selected to be the two learners and the former is better than the latter. Let $r_i=0.3$, and then, apply formula (18) to {1, 2, 3, 4, 5, 6}. The resulting

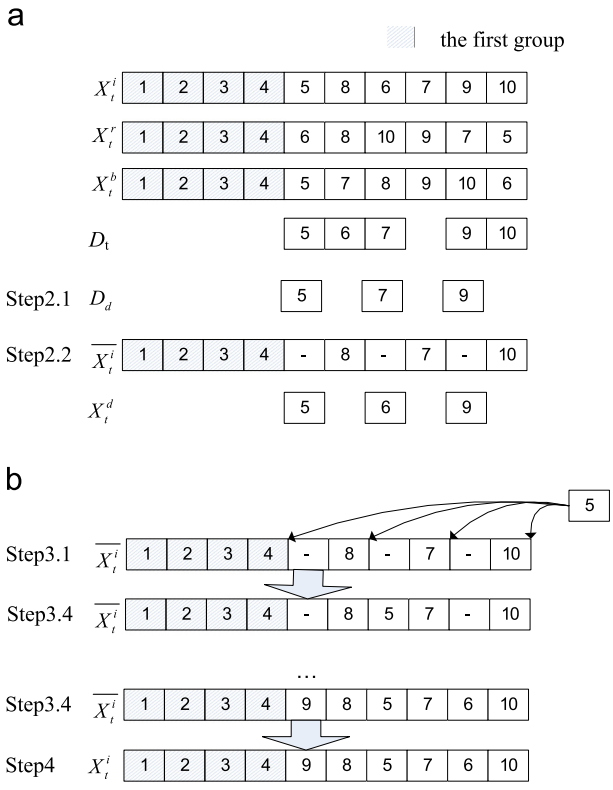

Fig. 2. Flowchart of the *MIG_localSearch* function.

vector is {−1.5, 1.5, 3.5, 4, 1, 8.5}. After the repair process, the new resulting solution is {4, 1, 3, 6, 5, 2}. Then, apply the last step in Section 3.2 to update the selected learner.

4.4. Learning phase-II

For the learner X_t^i, perform the following steps.

Step 1. From the current population, randomly select another learner X_t^r that is different from the current learner X_t^i.
Step 2. Compute the difference between X_t^r and X_t^i, and then, store the difference into a vector called D_t.
Step 3. Apply the *MIG_localSearch* function by inputting the three parameters, i.e., the current learner X_t^i, the difference vector D_t, and the destruction length d.

5. The framework of DTLBO

5.1. Encoding

In this study, we use the permutation representation mechanism for solving the flowshop rescheduling problem. In other words, in the solution representation, each element is represented by an integer that corresponds to the job number. For example, given a flowshop rescheduling problem with 5 jobs and 3 machines, one of the solutions is {3, 2, 5, 4, 1}, which means that on each machine, the processing order is J_3, J_2, J_5, J_4, and J_1.

5.2. Rescheduling decoding approach

In this study, we simultaneously consider five types of disruption, i.e., machine breakdowns, new job arrivals, cancellation of jobs, job processing variation, and job release variation.

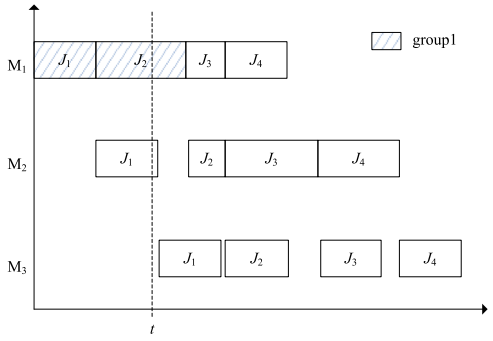

Fig. 3. Job division.

5.2.1. Job division

When any disruption occurs at time point t, the first step to react to the event is to divide all of the jobs into two groups, i.e., π_{pf} and π_p. The job whose first operation has been started will be classified into the first group π_{pf}, while the second group π_p contains the subsequent jobs. For the following example in Fig. 3, four jobs are to be processed on three machines. At time point t, a machine breakdown event occurs on machine M_2. Then, the four jobs will be classified into two groups based on whether their first operation has been started. Therefore, the first group π_{pf} contains J_1 and J_2, while the second group π_p includes J_3 and J_4. The jobs in the first group should keep the scheduling sequence, while the jobs in the second group can be rescheduled to minimise the given criteria.

5.2.2. Event processing method

(1) Machine breakdown
In this study, machine breakdowns occur only on busy machines, and the affected operation that has started its work on the breakdown machine can continue its remaining work on the same machine. To schedule each operation after the machine breakdown disruption, three situations must be considered for the breakdown machine, i.e., the operations that finished before the disruption, the operations that overlapped with the disruption, and the operations that started after the disruption. Fig. 4 gives an example to illustrate the decoding procedure for the three situations.
Fig. 4(a) gives the first situation, which is guaranteed by Constraints (4) and (5) in Section 2. In the first situation, operation O_{12} has completed its tasks before the machine breakdown disruption, and therefore, it will retain the starting and completion time as in the on-going schedule. Fig. 4(b) reports the second situation, which is guaranteed by Constraints (6) and (7) in Section 2. In the second situation, operation O_{22} overlaps with the machine breakdown disruption. In that situation, we should retain the assigned machine for the interrupted operation and delay the subsequent work until the recovery of the breakdown machine. Therefore, in the second situation, the interrupted operation O_{22} will retain its starting time as in the on-going schedule while postponing its completion time. Fig. 4(c) lists the third situation, which is guaranteed by Constraints (8) and (9) in Section 2. In the last situation, operation O_{22} begins its task after the machine breakdown disruption. We should guarantee that the starting time of the affected operation is not earlier than the recovery time of the breakdown machine.
(2) New job arrival
In the situation in which a new job arrives at time point t, all of the jobs should be divided into two groups, and the newly arrived job should be classified into the second group. In other words, the newly arrived job should be included in the

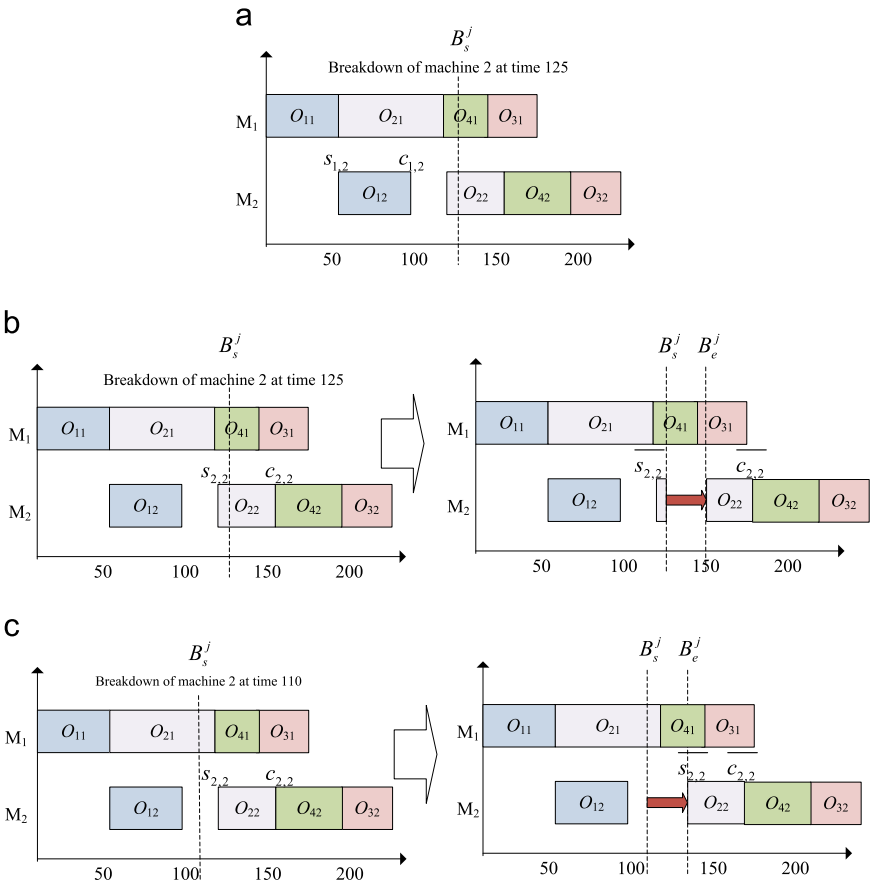

Fig. 4. Machine breakdown case. (a) The first situation: the completion time of the job is less than B_s^j. (b) The second situation: the completion time of the job is between B_s^j and B_e^j. (c) The third situation: the starting time of the job is later than B_s^j.

rescheduled parts. After the arrival of the new job, the representation length of each learner in the population should be adjusted.

(3) Cancellation of jobs
In the cancellation of jobs situation, all of the work of the job to be cancelled that has been completed should remain, while the subsequent work should be cancelled. After the cancellation of the affected job, the representation length of each learner should also be adjusted.

(4) Job processing variation
At time point t, the processing of the affected job can be increased or decreased. In this situation, the completion time of the affected operation will affect the following operation on the same machine. Fig. 5 gives an example of the job processing variation, where job J_1 must increase its processing time on machine M_2 at time t. The rectangle that is filled with a grid and labelled 'J_1' represents the variation duration.

(5) Job release variation
At time point t, the affected operation should begin its processing on the given machine according to the on-going scheduling. However, the release time of the operation varies because of many factors, such as the material that is delayed and the transportation time variation. In this case, the assigned machine should wait until the operation is available. The situation of job release variation is given in Fig. 6.

(6) Multiple disruptions

At time point t, when there are multiple disruptions that occur simultaneously, we should process these multiple disruptions immediately. The detailed procedures are given as follows:

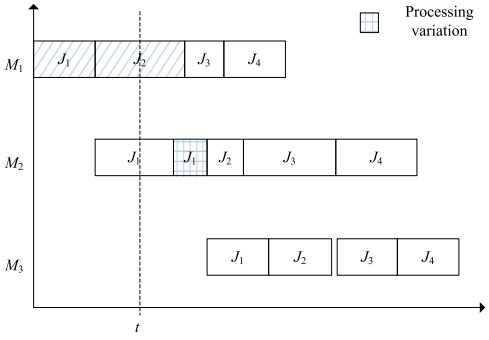

Fig. 5. Job processing variation.

Step 1. Divide all of the jobs into two groups based on the disruption time t.

Step 2. For the machine breakdown event, increase the breakdown duration for the processing time of the affected operation on the breakdown machine; for the newly arrived job, add the job to the second group for rescheduling; for the cancelled job, retain its completed work and delete it from the two divided groups; for the release variation job, postpone its starting time on the affected machine; and for the processing time variation job, change its processing time on the affected machine.

w?>Step 3. If the machine breakdown and processing time variation occur simultaneously, then add the maximal value between the machine breakdown duration D_{mb} and processing time variation V_{pt}, i.e., $\max\{D_{mb}, V_{pt}\}$, to the processing time of the affected operation.

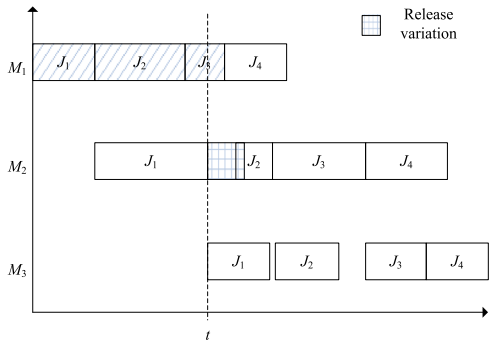

Fig. 6. Job release variation.

Step 4. If the machine breakdown and release time variation occur simultaneously, then select the maximal value between the machine restart time and job release time as the starting time of the affected operation.
Step 5. After processing all of the disruption events, reschedule all of the jobs in the second group to minimise the objective values of the system.

5.3. Normalisation of the objective

Because the makespan and instability values are measured in different units and could have different orders of magnitude, in this study, we used the normalisation process, which can be referred to in (Katragjini et al., 2013), to scale their values in such a way that they all fall into the range of 0 to 1. After applying the normalisation process, the objective function for the solution S is computed as follows:

$$f(S) = w_1 * Np(f_1(S)) + (1 - w_1) * Np(f_2(S)) \quad (20)$$

where $Np(f_1(S))$ and $Np(f_2(S))$ represent the normalisation process function for the objective function values $f_1(S)$ and $f_2(S)$, respectively. These functions are calculated as follows:

$$Np(f_1(S)) = \frac{f_1(S) - low(f_1)}{up(f_1) - low(f_1)} \quad (21)$$

$$Np(f_2(S)) = \frac{f_2(S) - low(f_2)}{up(f_2) - low(f_2)} \quad (22)$$

where $low(.)$ and $up(.)$ represent the lower and upper bounds for the corresponding objective function, respectively. The computational process for the lower and upper bounds can be found in (Katragjini et al., 2013).

5.4. Initialisation

To begin the algorithm, a population of learners should be initialised. It is crucial to generate an initial population that has a high level of quality and diversity. In this study, the initialisation method is the following:

Step 1. Generate a learner by using the NEH method (Nawaz et al., 1983).
Step 2. Perform the following step for P_s-1 times.
Step 3. Randomly generate a learner. If the newly generated solution is not equal to any individual in the current population, then insert it into the population. Otherwise, discard it.

5.5. Framework

The flowchart of the proposed algorithm is the following:

Step 1. Set the system parameters.
Step 2. Generate the initial population of learners.
Step 3. Evaluate each learner and select the best learner as the current teacher. Record the best solution that has been found thus far.
Step 4. If the stop condition is satisfied, then stop the algorithm. Otherwise, perform the following steps:
Step 5. Teaching phase
  Step 5.1 Apply the teaching phase heuristic TP-I or TP-II for each learner in the current population.
  Step 5.2 Record the best solution that has been found thus far, and update the current teacher.

Step 6. Learning phase
  Step 6.1 Apply the learning phase heuristic LP-I or LP-II for each learner in the current population.

Step 6.2 Record the best solution that has been found thus far.

5.6. A simple example

Suppose that we have a flowshop rescheduling problem that has five jobs and three machines, and the processing times are given in Table 1. At time point 10, a machine breakdown event occurs on machine M_1. Given the on-going schedule {4, 1, 2, 5, 3}, the Gantt chart is illustrated in Fig. 7. Then, the detailed rescheduling process can be listed as follows: first, the two divided groups are {4} and {1, 2, 5, 3}. In other words, the rescheduling part is {1, 2, 5, 3}; second, after the TLBO method is applied, many neighbouring solutions are generated and evaluated. Fig. 8 reports the optimal solution considering the machine breakdown disruption for the example problem that is given in Fig. 7. In the two figures, each operation is represented by a rectangle. The starting and completion time of each operation are listed in the upper left corner and lower right corner, respectively. The sequence of each operation on each machine represents the scheduling order. For example, in Fig. 7, on machine M_1, the processing sequence is J_4, J_1, J_2, J_5, J_3. In Fig. 8, the machine breakdown event on M_1 is considered and is represented by a rectangle labelled with 'B'.

It can be seen from Figs. 7 and 8 that only the two operations O_{41} (the first operation of job J_4) and O_{33} (the last operation of job J_3) have the same starting time in the on-going and new schedules. Therefore, we have thirteen tasks whose starting times have been altered in the new schedule, and thus, the value of f_2 is 13. Then, the fitness value is equal to 173.33, with $f_1 = 242$, $f_2 = 13$, and $w_1 = 0.7$.

6. Numerical results

To test the efficiency and effectiveness of the proposed algorithm, we select 90 Taillard's instances with different disruption events from (Katragjini et al., 2013), which can be downloaded from the website "http://soa.iti.es/rruiz". Five types of disruption events are considered in the proposed algorithm, i.e., machine breakdown, new job arrivals, cancellation of jobs, job processing variation, and job release variation. The proposed algorithm is

Table 1
Processing time.

	J_1	J_2	J_3	J_4	J_5
M_1	30	32	33	30	30
M_2	20	35	28	20	20
M_3	50	37	35	50	10

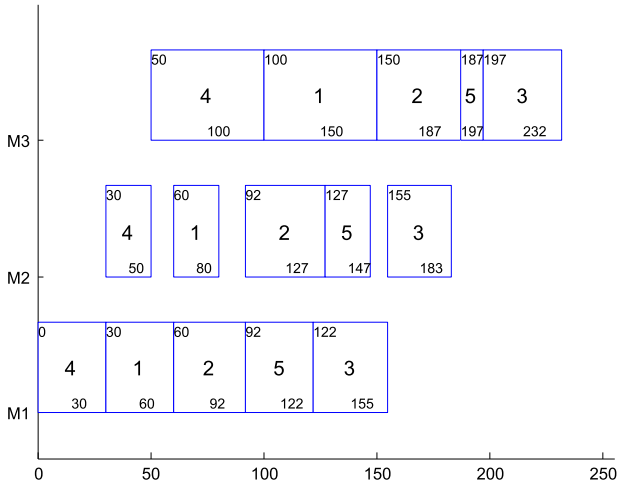

Fig. 7. Ongoing schedule.

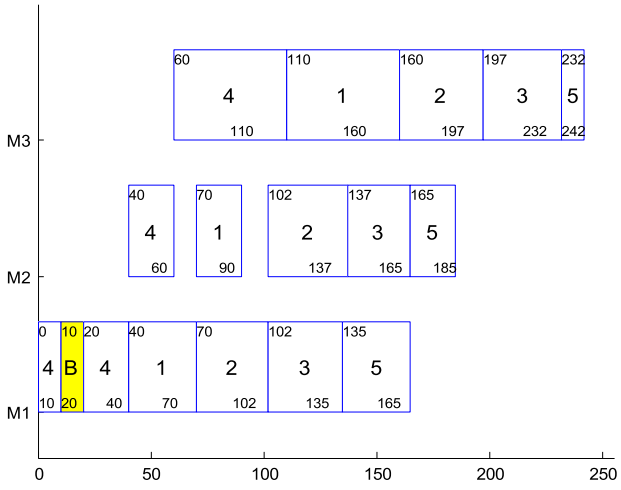

Fig. 8. Scheduling after machine breakdown.

coded in C++, on a DELL *i*7 CPU with 16GB memory. Each instance is performed with 20 runs, and the best and average values are collected for comparison.

6.1. Experimental instances

The given 90 instances range from 20 jobs and 5 machines to 100 jobs and 20 machines. For each instance, we tested five independent benchmarks with different disruption events from Katragjini et al. Katragjini et al. (2013). In other words, in total, we tested 450 instances to verify the performance of our proposed algorithms. To make the comparison results clear and easy to understand, we group each instance by the initial Taillard problem size. Therefore, in the end, we obtain nine groups of instances, and each group contains fifty different and independent problems.

The disruption events benchmarks and the detailed description for all types of disruption events can be found in http://soa.iti.es/. For example, for the machine breakdown disruption, the schedule disruption is simulated by generating random machine breakdowns at time t, $0 \leq t \leq \mathrm{Cmax}(B)$, where $\mathrm{Cmax}(B)$ denotes the makespan of baseline B. The baseline B for each instance is generated by using the Iterated Greedy algorithm (IG) of Ruiz and Stutzle (2007) (Ruiz and Stützle, 2007). In addition, machine breakdowns occur only on busy machines, i.e., machines do not undergo failures during idle times. The downtime duration is determined immediately after the event occurs. Down-times are generated using a uniform distribution in the range $U[1,\ldots,99]$. No other disruption is observed on the same machine before the breakdown disturbance is recovered. A job that is preempted due to a machine breakdown resumes its processing from the point at which the interruption occurred. At most, one disruption of a known duration D occurs at every time t, i.e., only one machine can be affected by a breakdown at time t.

6.2. Experimental parameters

In this study, the proposed algorithm has several parameters. We conducted a preliminary experiment to set the parameters. In the experiment, we try out several typical values for each parameter while fixing the other parameters. After the preliminary experiments and analysis, we find two key parameters that have an important role for the algorithm performance. According to reference (Katragjini et al., 2013) and our preliminary experiments, the other less important parameters are set as follows: (1) The destruction length d is set in the following way: if the length of the second group (hereafter called L_{se}) that was discussed in Section 5.2.1 exceeds or is equal to 4, then set d to 4; otherwise, set d to L_{se} -1. (2) According to the computational complexity of the proposed algorithm, the maximal number of candidate positions for an erased job in the construction stage is given by $J_{max}=50$. (3) The time limit to stop the algorithm is $S_c=50$ s.

To set the value for the two key parameters, i.e., the population size P_s and the teacher learning factor T_F, we implement six types of algorithms with different values, which are given in Table 2. The canonical TLBO are implemented by using TP-I and LP-I heuristics in the teaching phase and learning phase, respectively. For each instance, we memorised the best solution that can be found by all of the compared algorithms, and we calculated the relative percentage deviation over the best solution for each compared algorithm, which is computed as follows:

$$RPD_i = \frac{Comp_i^k - Best_i}{Best_i} \times 100 \tag{23}$$

where $Comp_i^k$ is the optimal solution that is found by the kth compared algorithm, while $Best_i$ is the best solution that is found by all of the compared algorithms. In the comparison results, we calculated the average relative percentage deviation ($\overline{RPD}$) for each group of the same problem size.

The detailed comparisons of the best objective values for the given nine groups of instances among the six proposed algorithms are given in Table 3. The weighted coefficient is set to 0.5 to represent the same importance in the two objectives. It can be seen from Table 3 that for solving the given nine groups of instances with different problem scales, the proposed TLBO-V is the best among the given six algorithms. The following algorithms are TLBO-VI, TLBO-IV, TLBO-III, TLBO-II, and TLBO-I, respectively. Therefore, in the following experiments, we set the first two parameters to be the same as with TLBO-V.

6.3. Effectiveness of different DTLBO heuristics

From the experimental analysis discussed in Section 6.2, we conclude that the proposed TLBO-V is the best algorithm among

Table 2
Different values for the first two parameters.

Algorithm	P_s	T_F
TLBO-I	10	1
TLBO-II	20	1
TLBO-III	10	2
TLBO-IV	20	2
TLBO-V	50	1
TLBO-VI	50	2

Table 3
Comparison results of the best $\overline{RPD}$ values ($w_1=0.5$).

Problem	$\overline{RPD}$					
	TLBO-I	TLBO-II	TLBO-III	TLBO-IV	TLBO-V	TLBO-VI
Ta20 × 5	3.91	3.92	0.97	1.05	0.00	1.48
Ta20 × 10	4.38	4.02	3.44	3.51	2.12	0.00
Ta20 × 20	1.79	1.73	1.21	0.75	1.07	0.00
Ta50 × 5	2.07	2.32	2.34	2.04	0.00	2.13
Ta50 × 10	2.78	2.09	1.75	1.09	0.00	1.68
Ta50 × 20	1.43	1.32	0.21	0.32	0.00	1.07
Ta100 × 5	0.20	0.00	0.14	0.21	0.34	0.38
Ta100 × 10	0.21	0.11	0.00	0.09	0.19	0.43
Ta100 × 20	0.35	0.36	0.07	0.00	0.26	0.64
Average	1.90	1.76	1.13	1.01	**0.44**	0.87

Table 4
Four algorithms with different implementations.

	Teaching phase	Learning phase
DTLBO-I	TP-I	LP-I
DTLBO-II	TP-I	LP-II
DTLBO-III	TP-II	LP-I
DTLBO-IV	TP-II	LP-II

Table 5
Comparison results of the best $\overline{RPD}$ values ($w_1=0.5$).

Problem	$\overline{RPD}$			
	DTLBO-I	DTLBO-II	DTLBO-III	DTLBO-IV
Ta20 × 5	5.17	0.02	0.00	0.05
Ta20 × 10	5.25	0.76	0.01	0.00
Ta20 × 20	5.80	0.75	0.00	0.00
Ta50 × 5	14.55	0.00	3.61	3.93
Ta50 × 10	4.88	0.04	0.00	1.22
Ta50 × 20	4.00	0.00	0.86	1.79
Ta100 × 5	4.67	0.00	1.15	0.93
Ta100 × 10	2.51	0.00	0.28	0.08
Ta100 × 20	0.74	0.19	2.29	0.00
Average	5.29	**0.20**	0.91	0.89

Table 6
Comparison results of the best $\overline{RPD}$ values ($w_1=0.1$).

Problem	TLBO	IG	DTLBO
Ta20 × 5	9.89	3.16	0.00
Ta20 × 10	6.94	1.73	0.00
Ta20 × 20	6.43	1.15	0.00
Ta50 × 5	11.94	10.89	0.00
Ta50 × 10	5.46	5.29	0.00
Ta50 × 20	2.34	2.34	0.00
Ta100 × 5	6.60	6.41	0.00
Ta100 × 10	2.85	2.78	0.00
Ta100 × 20	1.75	1.74	0.00
Average	6.02	3.94	0.00

the six compared algorithms, and it will be named DTLBO-I hereafter. In this section, we also develop three other types of DTLBO algorithms, which are called DTLBO-II, DTLBO-III, and DTLBO-IV, to test the effects of different discretisation approaches for five types of disruptions. The detailed differences in the implementations of the four algorithms are given in Table 4.

Table 5 gives the comparison results that were collected from the proposed algorithms for solving the given nine groups of instances.

Table 7
Comparison results of the best $\overline{RPD}$ values ($w_1=0.5$).

Problem	TLBO	IG	DTLBO
Ta20 × 5	9.40	2.77	0.00
Ta20 × 10	5.62	1.46	0.00
Ta20 × 20	3.43	0.00	0.31
Ta50 × 5	13.52	12.13	0.00
Ta50 × 10	4.22	4.00	0.00
Ta50 × 20	2.81	2.81	0.00
Ta100 × 5	6.83	6.45	0.00
Ta100 × 10	2.51	2.40	0.00
Ta100 × 20	1.19	1.17	0.00
Average	5.50	3.69	0.03

Table 8
Comparison results of the best $\overline{RPD}$ values ($w_1=0.9$).

Problem	TLBO	IG	DTLBO
Ta20 × 5	4.14	1.04	0.00
Ta20 × 10	1.48	0.55	0.00
Ta20 × 20	0.24	0.13	0.00
Ta50 × 5	9.91	9.14	0.00
Ta50 × 10	2.19	2.19	0.00
Ta50 × 20	1.61	1.61	0.00
Ta100 × 5	5.70	5.52	0.00
Ta100 × 10	2.14	2.10	0.00
Ta100 × 20	1.62	1.62	0.00
Average	3.23	2.66	0.00

Table 9
Comparison results of the best $\overline{RPD}$ values ($w_1=0.1$).

Problem	$\overline{RPD}$				
	DTLBO-II	hGA	IG	ILS	PSO
Ta20 × 5	0.00	2.22	1.90	11.05	2.54
Ta20 × 10	0.00	1.01	2.29	7.94	3.08
Ta20 × 20	0.13	0.00	2.44	5.93	2.06
Ta50 × 5	0.00	14.31	7.98	19.90	18.58
Ta50 × 10	0.00	8.97	3.86	10.10	10.52
Ta50 × 20	0.00	5.63	4.70	7.90	6.95
Ta100 × 5	0.00	4.50	1.13	5.11	5.12
Ta100 × 10	0.00	2.36	0.54	3.82	4.29
Ta100 × 20	0.00	5.17	0.50	5.08	6.03
Average	**0.01**	4.91	2.82	8.54	6.58

Table 10
Comparison results of the average $\overline{RPD}$ values ($w_1=0.1$).

Problem	$\overline{RPD}$				
	DTLBO-II	hGA	IG	ILS	PSO
Ta20 × 5	0.00	4.06	1.73	7.51	3.49
Ta20 × 10	0.00	2.05	1.82	5.39	3.64
Ta20 × 20	0.00	0.05	1.39	2.27	3.32
Ta50 × 5	0.00	8.48	6.97	5.76	7.60
Ta50 × 10	0.00	5.52	3.85	1.40	4.78
Ta50 × 20	0.00	4.47	2.75	1.83	2.76
Ta100 × 5	0.00	2.20	1.86	0.34	1.28
Ta100 × 10	0.06	1.65	2.35	0.00	1.61
Ta100 × 20	0.00	2.09	2.30	0.07	1.74
Average	**0.01**	3.40	2.78	2.73	3.36

In Table 5, the first column gives the name of each group of benchmarks, which is represented by a pair of numbers, i.e., the number of jobs and the number of machines. The following four columns report the best $\overline{RPD}$ values that were collected by the four algorithms, i.e.,

Table 11
Comparison results of the best $\overline{RPD}$ values ($w_1=0.5$).

Problem	$\overline{RPD}$				
	DTLBO-II	hGA	IG	ILS	PSO
Ta20 × 5	0.00	1.98	2.10	7.56	2.19
Ta20 × 10	0.06	0.04	0.00	6.16	1.67
Ta20 × 20	0.00	0.05	1.73	4.26	1.11
Ta50 × 5	0.00	15.69	7.62	18.89	17.35
Ta50 × 10	0.00	7.19	2.49	9.03	8.95
Ta50 × 20	0.00	6.65	3.09	8.25	7.39
Ta100 × 5	0.00	3.97	1.76	4.98	4.62
Ta100 × 10	0.00	2.03	0.56	2.55	3.82
Ta100 × 20	0.00	4.48	0.59	4.75	5.07
Average	**0.01**	4.68	2.22	7.38	5.80

Table 12
Comparison results of the average $\overline{RPD}$ values ($w_1=0.5$).

Problem	$\overline{RPD}$				
	DTLBO-II	hGA	IG	ILS	PSO
Ta20 × 5	0.00	1.27	1.75	4.10	2.40
Ta20 × 10	0.00	2.10	1.70	4.55	3.14
Ta20 × 20	0.00	0.55	1.22	2.06	2.78
Ta50 × 5	0.00	7.08	6.00	5.22	6.93
Ta50 × 10	0.00	5.47	3.66	1.39	4.24
Ta50 × 20	0.00	3.92	2.07	1.21	2.30
Ta100 × 5	0.00	2.05	1.77	0.62	1.54
Ta100 × 10	0.90	2.11	2.99	0.00	2.16
Ta100 × 20	0.12	1.52	1.58	0.00	0.98
Average	**0.11**	2.90	2.53	2.13	2.94

Table 13
Comparison results of the best $\overline{RPD}$ values ($w_1=0.9$).

Problem	$\overline{RPD}$				
	DTLBO-II	hGA	IG	ILS	PSO
Ta20 × 5	0.00	0.39	0.56	1.30	0.70
Ta20 × 10	0.00	0.00	0.52	1.69	0.43
Ta20 × 20	0.02	0.00	0.34	0.51	0.12
Ta50 × 5	0.00	8.51	8.12	11.38	9.93
Ta50 × 10	0.00	9.74	6.62	11.04	9.67
Ta50 × 20	0.00	8.82	4.76	9.40	6.82
Ta100 × 5	0.00	4.23	1.31	4.55	3.27
Ta100 × 10	0.00	2.49	1.58	3.55	3.50
Ta100 × 20	0.00	2.55	0.72	3.84	2.60
Average	**0.00**	4.08	2.72	5.25	4.11

DTLBO-I, DTLBO-II, DTLBO-III, and DTLBO-IV. The last line in Table 5 tells the average performance for each of the compared algorithms. It can be concluded from Table 5 that (1) for solving the first three groups of benchmarks that have a relatively small scale, e.g., problems with 20 jobs, DTLBO-IV performs the best among the four compared algorithms; (2) for solving the medium-scale problems that have 50 jobs, DTLBO-II is the best algorithm. It should be noted that DTLBO-I obtains a relatively worse result for solving the "Ta50×5" group of benchmarks; (3) for solving the last three groups of instances which have relatively large scales, DTLBO-II shows superior performance compared with the other three algorithms; (4) from the last row in the table, we can see that, on average, DTLBO-II is the best algorithm among the four algorithms and is obviously better than the other compared algorithms. It should be noted that DTLBO-I is obviously worse than the other three compared algorithms. The comparison results in Table 5 verify that the proposed IG-based local search function *MIG_localSearch* can enhance the searching ability of the proposed algorithm. However, because of the computational complexity, it is a good choice to utilise the local search function *MIG_localSearch* in the learning phase rather than in both the teaching and learning phases.

Table 14
Comparison results of the average $\overline{RPD}$ values ($w_1=0.9$).

Problem	$\overline{RPD}$				
	DTLBO-II	hGA	IG	ILS	PSO
Ta20 × 5	0.00	0.41	0.88	0.66	1.24
Ta20 × 10	0.00	0.26	0.58	0.90	0.76
Ta20 × 20	0.00	0.20	0.24	0.34	0.38
Ta50 × 5	0.00	5.20	3.78	4.23	5.92
Ta50 × 10	0.00	7.67	5.45	4.96	7.97
Ta50 × 20	0.00	5.99	2.46	3.11	5.74
Ta100 × 5	0.00	3.11	1.57	1.14	2.96
Ta100 × 10	0.00	2.38	2.22	0.54	2.58
Ta100 × 20	0.00	1.76	1.23	1.30	1.56
Average	**0.00**	3.00	2.05	1.91	3.23

Table 15
Comparisons of the average computational times.

Problem	$\overline{RPD}$				
	DTLBO-II	hGA	IG	ILS	PSO
Ta20 × 5	0.34	0.80	1.96	0.02	0.55
Ta20 × 10	0.44	0.55	2.54	0.11	0.77
Ta20 × 20	0.75	1.10	2.03	0.13	1.04
Ta50 × 5	20.10	8.29	12.46	1.45	5.65
Ta50 × 10	23.17	10.57	12.46	2.13	5.64
Ta50 × 20	25.07	11.95	12.03	2.66	5.73
Ta100 × 5	25.62	12.45	12.92	1.56	3.39
Ta100 × 10	32.01	15.02	8.96	1.92	3.77
Ta100 × 20	41.12	15.44	10.01	1.62	4.83
Average	18.74	8.46	8.37	1.29	3.48

6.4. Comparisons with the IG algorithm for three types of disruptions

In this section, we make a detailed comparison with the canonical TLBO and IG (developed by Katragjini et al. (2013) by using the same instances, the same disruption events (machine breakdowns, new job arrivals and release time delays) and the same objective function (makespan and instability).

Tables 6, 7, and 8 give the comparison results that were collected from the three compared algorithms, with w_1 set to 0.1, 0.5, and 0.9, respectively. In the three comparison tables, the first column tells the name of each group of benchmarks. The next three columns report the $\overline{RPD}$ values that were collected from the three algorithms, i.e., TLBO, IG, and DTLBO. The last line tells the average performance for each compared algorithm. It can be concluded from the three tables that (1) in the case in which the weight coefficient value (w_1) is set to 0.1, which means that we assign more importance to the instability objective, the proposed DTLBO algorithm obtained optimal values for all of the instances. The average performance in the last line also shows that DTLBO is obviously better than the other two algorithms; (2) for a comparison with $w_1=0.5$, DTLBO can obtain eight better results out of nine groups of instances. The second best algorithm, IG, can obtain only one optimal result; (3) for the case in which $w_1=0.9$, which assigns more importance to the makespan objective, DTLBO can also obtain all of the optimal values for the considered problems. The average performance also shows the efficiency of the proposed algorithm.

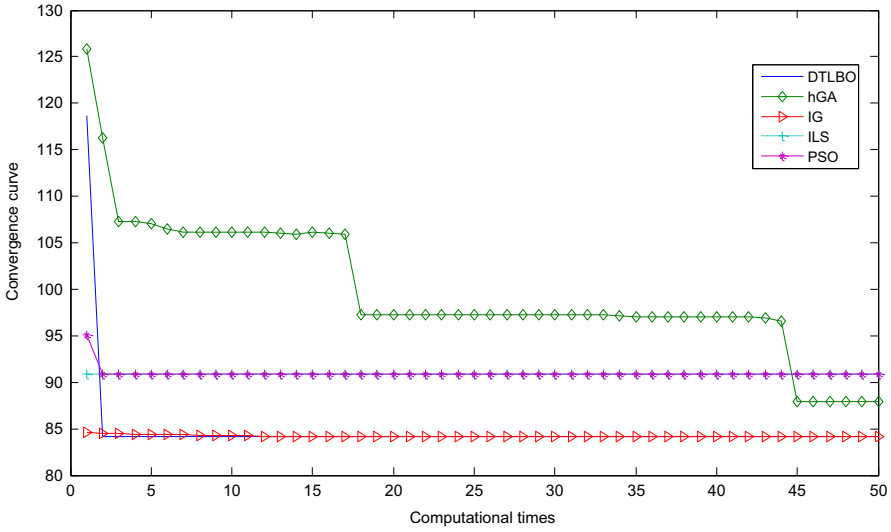

Fig. 9. Convergence curve for one instance of "Ta20×5".

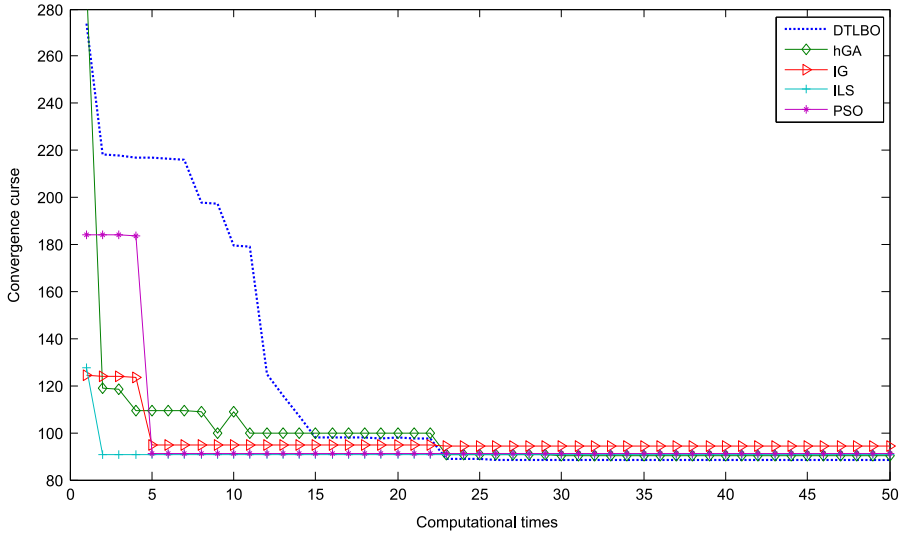

Fig. 10. Convergence curve for one instance of "Ta100×20".

6.5. *Comparisons with other algorithms for five types of disruptions*

To the best of our knowledge, there is not any present algorithm for solving the flowshop rescheduling problem while considering the five disruption events that are discussed above. Therefore, to make further detailed comparisons with the present algorithms, we coded the following four efficient algorithms: hGA by Ruiz and Maroto (2006), IG by Katragjini et al. (2013), ILS by Dong et al. (2009), and PSO by Liao et al. (2007). The parameters for the compared algorithms are set to the same values as in the literature, except that the stop condition is set to 50 s.

To make detailed comparisons, we tested each compared algorithm while considering all of the three different importances in the two objectives. In other words, for each compared algorithm, we took three independent runs for $w_1=0.1$, 0.5, and 0.9, respectively. Tables 9 and 10 give the comparison results of the best and average $\overline{RPD}$ values, respectively, in the case in which the weight coefficient value (w_1) was set to 0.1, which means that we give more importance to the instability objective. Tables 11 and 12 report the comparison results of the best and average $\overline{RPD}$ values, respectively, where w_1 is set to 0.5 for the same importance in the two objectives. Last, Tables 13 and 14 tell the comparison results for the best and average $\overline{RPD}$ values, respectively, where w_1 is set to 0.9 for more importance in the first makespan objective. In the six comparison tables, the first column gives the problem name, and the following five columns report the $\overline{RPD}$ results that were collected by the five compared algorithms, i.e., DTLBO-II, hGA, IG, ILS, and PSO, respectively.

It can be concluded from Table 9 that (1) for solving the first three groups of benchmarks with relatively small scales, on average, DTLBO-II performs the best among the five compared algorithms. The hGA algorithm can obtain the optimal result for solving "Ta20×20", which is slightly better than the proposed algorithm; (2) for solving medium-scale problems that have 50 jobs, DTLBO-II obtains all of the optimal values, which is obviously better than the other compared algorithms; (3) for solving the last three groups of instances, which have relatively large scales, DTLBO-II also shows superior performance compared with the other three algorithms. At the same time, IG performs slightly worse than DTLBO-II when solving the last three groups of instances, which is better than the other three compared algorithms, i.e., hGA, ILS, and PSO; (4) from the last row in the table, we can see that, on average, DTLBO-II is the best algorithm among the five compared algorithms, and the other algorithms are IG, hGA, PSO, and ILS, respectively.

For considering the average performance in Table 10, we can see that (1) on average, DTLBO-II obtains eight optimal values for the given nine groups of benchmarks, except for "Ta100×10", which is slightly worse than the result from ILS; and (2) from the last line in the table, we can see that, on average DTLBO-II performs the best. The other algorithms are ILS, IG, PSO, and hGA. The above comparison results verify the robustness of the proposed algorithm.

Tables 11 and 13 give the comparison results in which more importance is given to the makespan objective. It can be concluded from the two tables that (1) DTLBO-II can obtain eight optimal solutions out of the nine groups of benchmarks, which verifes the efficiency of the proposed algorithm; and (2) on average, when considering the searching ability, DTLBO-II is the best among the five compared algorithms. The other algorithms are IG, hGA, PSO, and ILS. It can be seen from Tables 12 and 14 that, when considering the average performance in which more importance is given to the first objective, DTLBO-II shows the best among the five compared algorithms, which further verifies the robustness of the proposed algorithm.

Table 15 gives the comparisons of the average computational times consumed by the five compared algorithms. It can be seen from Table 15 that (1) when solving the three groups of small-scale instances, DTLBO-II gives better performance than hGA, IG, and PSO, which shows that the proposed algorithm holds a better convergence performance; and (2) when solving the medium and large-scale problems, the proposed algorithm consumes more than 20 s, which is obviously larger than the other compared algorithms. However, the solutions that were obtained by the proposed algorithm are obviously better than the results of the other compared algorithms. Considering the results performance and the computational times, we can conclude that the proposed algorithm captures a balance between the exploitation and exploration ability.

To further verify the convergence ability of the proposed algorithm, we select two instances, which are from the first group and the last group, to make detailed comparisons on the convergence performance. Figs. 9 and 10 report the convergence comparisons for the two instances. It can be concluded from the two figures that (1) when solving the small-scale instance, the proposed algorithm converges to an optimal result quickly, which is similar to the IG algorithm and better than the other compared algorithms; and (2) when solving the large-scale instance, DTLBO is the algorithm that obtains the best result among the five algorithms, while IG and PSO quickly converge to a relatively worse result. Figs. 9 and 10 further verify that the proposed algorithm captures a balance between the exploration and exploitation ability.

6.6. Experimental analysis

In the proposed algorithm, we developed two types of teaching phase heuristics, named TP-I and TP-II, respectively, and two types of learning phase heuristics, namely LP-I, and LP-II, respectively. In TP-I and LP-I, the procedure to repair the newly-generated learner discussed in Section 4.1.4 consumed the main computational resources with the computational time complexity of $O(nm)$. In TP-II and LP-II, the function *MIG_localSearch* is the main factor and has the computational time complexity of $O(n^3m)$. To further decrease the computational times that are consumed in the *MIG_localSearch* function, we limit the number of candidate insert positions in the modified IG-based local search procedure. The learning process during the teaching phase can improve the exploitation and exploration ability of the learner, while the learning phase can further improve the efficiency of each learner in the population. Through the above two phases, the proposed algorithm shows better performance in solving both the small- and larger-scale flowshop rescheduling problems.

7. Conclusions

In this study, we proposed a discrete version of TLBO for solving flowshop rescheduling problems. In the proposed algorithms, a detailed implementation for the teaching and learning phases is investigated. In the teaching phase, by learning the difference between the teacher and the mean result of the current population, each learner can improve both the solution quality and the exploration ability. In the learning phase, by learning from the other learners and applying the modified IG-based local search function, each learner can further improve the performance. Five types of disruptions are simultaneously considered in the proposed algorithms. Experimental comparisons with different versions of DTLBO and other efficient algorithms verify the efficiency and effectiveness of the proposed algorithm. Future work is to be mainly focused on the following issues: (1) Apply the proposed algorithm for solving rescheduling problems in hybrid and flexible environments; and (2) Develop a fast evaluation method for multi-objective flowshop rescheduling problems to further decrease the computational complexity of the proposed algorithm.

Acknowledgement

This research is partially supported by the National Science Foundation of China 61174187, 51435009, 61374187 and 61104179, Program for New Century Excellent Talents in University (NCET-13-0106), Specialized Research Fund for the Doctoral Program of Higher Education (20130042110035), Science Foundation of Liaoning Province in China (2013020016), Basic scientific research foundation of Northeast University under Grant N110208001 and N130508001, Starting foundation of Northeast University under Grant 29321006, and IAPI Fundamental Research Funds (2013ZCX02).

References

Allahverdi, A., Mittenthal, J., 1995. Scheduling on a two-machine flowshop subject to random breakdowns with a makespan objective function. Eur. J. Op. Res. 81 (2), 376–387.
Dong, X., Huang, H., Chen, P., 2009. An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion. Comput. Op. Res. 36 (5), 1664–1669.
Garey, M.R., Johnson, D.S., Sethi, R., 1976. The complexity of flowshop and jobshop scheduling. Math. Op. Res. 1 (2), 117–129.
Hosseini, N., Tavakkoli-Moghaddam, R., 2013. Two meta-heuristics for solving a new two-machine flowshop scheduling problem with the learning effect and dynamic arrivals. Int. J. Adv. Manuf. Technol. 65 (5–8), 771–786.
Katragjini, K., Vallada, E., Ruiz, R., 2013. Flow shop rescheduling under different types of disruption. Int. J. Prod. Res. 51 (3), 780–797.
Liao, C.J., Tseng, C.T., Luarn, P., 2007. A discrete version of particle swarm optimisation for flowshop scheduling problems. Computers &. Op. Res. 34 (10), 3099–3111.
Liu, L., Zhou, H., 2013. On the identical parallel-machine rescheduling with job rework disruption. Comput. Ind. Eng. 66 (1), 186–198.
Madureira, A., Ramos, C., Silva, S.C., 2004. Toward dynamic scheduling through evolutionary computing. WSEAS Trans. Syst. 3 (4), 1596–1604.
Mirabi, M., Ghomi, S.M.T., Jolai, F., 2013. A two-stage hybrid flowshop scheduling problem in machine breakdown condition. J. Intell. Manuf. 24 (1), 193–199.
Nawaz, M., Enscore, E.E., Ham, I., 1983. A heuristic algorithm for the *m*-machine *n*-job flow-shop sequencing problem. Omega 11 (1), 91–95.
Nilsson, J., Bernhardsson, B., Wittenmark, B., 1998. Stochastic analysis and control of real-time systems with random time delays. Automatica 34 (1), 57–64.
Nof, S.Y., HANK GRANT, F., 1991. Adaptive/predictive scheduling: review and a general framework. Prod. Plan. Control 2 (4), 298–312.
Ouelhadj, D., Petrovic, S., 2009. A survey of dynamic scheduling in manufacturing systems. J. Sched. 12 (4), 417–431.
Pan, Q.K., Fatih, T.M., Liang, Y.C., 2008. A discrete particle swarm optimisation algorithm for the no-wait flowshop scheduling problem. Comput. Op. Res. 35 (9), 2807–2839.
Rahmani, D., Heydari, M., 2014. Robust and stable flow shop scheduling with unexpected arrivals of new jobs and uncertain processing times. J. Manuf. Syst. 33 (1), 84–92.
Rahmani, D., Heydari, M., Makui, A., Zandieh, M., 2013. A new approach to reducing the effects of stochastic disruptions in flexible flow shop problems with stability and nervousness. Int. J. Manag. Sci. Eng. Manag. 8 (3), 173–178.

Rao, R.V., Kalyankar, V.D., 2013. Parameter optimisation of modern machining processes using teaching-learning-based optimisation algorithm. Eng. Appl. Artif. Intell. 26, 524–531.
Rao, R.V., Patel, V., 2013. Multi-objective optimisation of two stage thermoelectric cooler using a modified teaching–learning-based optimisation algorithm. Eng. Appl. Artif. Intell. 26, 430–445.
Rao, R.V., Patel, V., 2013. Multi-objective optimisation of heat exchangers using a modified teaching-learning-based optimisation algorithm. Appl. Math. Model. 37, 1147–1162.
Rao, R.V., Savsani, V.J., Vakharia, D.P., 2012. Teaching–Learning-Based Optimisation: an optimisation method for continuous non-linear large scale problems. Inf. Sci. 183, 1–15.
Ruiz, R., Maroto, C., 2006. A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. Eur. J. Op. Res. 169 (3), 781–800.
Ruiz, R., Stützle, T., 2007. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. Eur. J. Op. Res. 177 (3), 2033–2049.
Ruiz, R., Maroto, C., Alcaraz, J., 2006. Two new robust genetic algorithms for the flowshop scheduling problem. Omega 34 (5), 461–476.
Schmidt, G., 2000. Scheduling with limited machine availability. Eur. J. Op. Res. 121 (1), 1–15.
Silva, C.A., Sousa, J.M.C., Runkler, T.A., 2008. Rescheduling and optimisation of logistic processes using GA and ACO. Eng. Appl. Artif. Intell. 21 (3), 343–352.
Tang, L., Liu, W., Liu, J., 2005. A neural network model and algorithm for the hybrid flow shop scheduling problem in a dynamic environment. J. Intell. Manuf. 16 (3), 361–370.
Tang, L., Zhao, Y., Liu, J., 2014. An improved differential evolution algorithm for practical dynamic scheduling in steelmaking-continuous casting production. IEEE Trans. Evol. Comput. 18 (2), 209–225.
Vieira, G.E., Herrmann, J.W., Lin, E., 2003. Rescheduling manufacturing systems, a framework of strategies, policies, and methods. J. Sched. 6 (1), 39–62.
Wang, K., Choi, S.H., 2012. A decomposition-based approach to flexible flow shop scheduling under machine breakdown. Int. J. Prod. Res. 50 (1), 215–234.
Wang, K., Choi, S.H., 2014. A holonic approach to flexible flow shop scheduling under stochastic processing times. Comput. Op. Res. 43, 157–168.
Wang, L., Zhang, L., Zheng, D.Z., 2005. A class of hypothesis-test-based genetic algorithms for flow shop scheduling with stochastic processing time. Int. J. Adv. Manuf. Technol. 25 (11–12), 1157–1163.
Xiong, J., Xing, L., Chen, Y., 2013. Robust scheduling for multi-objective flexible job-shop problems with random machine breakdowns. Int. J. Prod. Econ. 141 (1), 112–126.
Zandieh, M., Gholami, M., 2009. An immune algorithm for scheduling a hybrid flow shop with sequence-dependent setup times and machines with random breakdowns. Int. J. Prod. Res. 47 (24), 6999–7027.