[1]

An improved particle swarm optimization algorithm for dynamic job shop scheduling problems with random job arrivals

Zhen Wang[a,b], Jihui Zhang[a,*] Shengxiang Yang[c]

[a] *Institute of Complexity Science, College of Automation, Qingdao University, Qingdao,266071,China*
[b] *College of Electrical Engineering, Qingdao University, Qingdao,266071,China*
[c] *Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, United Kingdom*

Abstract

Random job arrivals that happen frequently in manufacturing practice may create a need for dynamic scheduling. This paper considers an issue of how to reschedule the randomly arrived new jobs to pursue both performance and stability of a schedule in a job shop. Firstly, a mixed integer programming model is established to minimize three objectives, including the discontinuity rate of new jobs during the processing, the makespan deviation of initial schedule, and the sequence deviation on machines. Secondly, four match-up strategies from references are modified to determine the rescheduling horizon. Once new jobs arrive, the rescheduling process is immediately triggered with ongoing operations remain. The ongoing operations are treated as machine unavailable constraints (MUC) in the rescheduling horizon. Then, a particle swarm optimization (PSO) algorithm with improvements is proposed to solve the dynamic job shop scheduling problem. Improvement strategies consist of a modified decoding scheme considering MUC, a population initialization approach by designing a new transformation mechanism, and a novel particle movement method by introducing position changes and a random inertia weight. Lastly, extensive experiments are conducted on several instances. The experiments results show that the modified rescheduling strategies are statistically and significantly better than the compared strategies. Moreover, comparative studies with five variants of PSO algorithm and three state-of-the-art meta-heuristics demonstrate the high performance of the improved PSO algorithm.

Keywords: New job arrival, Dynamic job shop scheduling, Match-up strategy, Particle swarm optimization, Multi-objective problem

1. Introduction

With the fast development of globalization and information technologies, how to quickly meet diversified needs of customers has been an urgent issue for modern manufacturing enterprises [1, 2]. Meanwhile, the major character of production mode presents the trend of multi-variety and small-batch manufacturing, and the job shop manufacturing has been one of the main production modes in industrial manufacturing enterprises.

Job shop scheduling problem (JSSP) is one of the most popular combinatorial optimization problems. Over the past decades, JSSPs have attracted considerable attention and extensive techniques have been developed to solve static JSSPs [3, 4]. However, industrial environments are inherently complex with occurrence of real time events, such as new job arrivals, machine breakdowns and due dates changing, etc. These may make the initial schedule to become inefficient or even invalid and require dynamic scheduling (also known as rescheduling) to update the initial schedule based on changes. Among those real time events, new job arrivals may happen more frequently than ever before in a fast-changing market environment [5]. Therefore, it is of significance to pay more attention to dynamic scheduling problems with consideration of new job arrivals in the modern manufacturing environment.

[1] *Corresponding author: J.H. Zhang. *E-mail address*: zhangjihui@qdu.edu.cn(J.H. Zhang).

Dynamic scheduling problems have received more concerns in recent years. Many rescheduling approaches have been developed in various manufacturing environments, including single-machine manufacturing systems [6], parallel-machine manufacturing systems [7], flow shops [8–11], job shops [12, 13] and flexible manufacturing systems [14]. However, compared with other manufacturing environments, there are fewer studies on the dynamic job shop scheduling problem (DJSSP) due to its high complexity. Studies on DJSSPs can date back to 1974. Holloway and Nelson [15] developed a multi-pass heuristic scheduling procedure for DJSSPs with due dates and variable processing times. Then, they extended the problem with intermittent job arrivals and statistical processing times [16]. Since then, increasing attentions had been attracted to dynamic scheduling problem. For detail reviews on DJSSPs, readers may refer to Ramasesh [17], Suresh and Chaudhuri [18], Vieira et al. [19], Ouelhadj and Petrovic [20], Potts and Strusevich [21], Mohan et al. [22] and Zhang et al. [23].

Early studies on DJSSPs with new job arrivals mainly concentrated on heuristic methods to obtain a near optimum solution. Muhlemann et al. [24] used dispatching disciplines for DJSSP with intermittent new job arrivals, estimation errors of uncertain processing times and machine breakdowns. Various performance measures were tested in the simulation process, including mean ratio of flow time to processing time, mean queueing time, mean lateness, percentage of tardy jobs and net CPU times. Chang [25] developed a new approach to provide real time estimations of the queueing times for the remaining operations of new jobs in a dynamic job shop. Jobs arrived continuously with interarrival time generated from a negative exponential distribution. Mean tardiness, and proportion of tardy jobs were tested as performance measures. Dominic et al. [26] proposed some new combined dispatching rules for DJSSP with continuously arriving new jobs. The considered performances included mean flow time, mean tardiness, maximum flow time, number of tardy jobs and tardiness variance. Lu and Romanowski [27] developed multi-contextual dispatching rules for DJSSP with continuously new job arrivals. Machine idle time and job's waiting time were combined with a single basic dispatching rule. Several composite dispatching rules were proposed to minimize the maximum completion time (makespan) and mean flow time. Sharma and Jain [28] addressed DJSSP with sequence-dependent setup times, in which jobs arrived continuously. Nine dispatching rules were developed and incorporated in a discrete event simulation model for several different performance measures, i.e. makespan, mean flow time, maximum flow time, mean tardiness, maximum tardiness, number of tardy jobs, total and mean setup time.

Obviously, when using priority dispatching rules or heuristics, schedules are easily constructed in real time. This type of scheduling approach is termed *completely reactive scheduling* (or *on-line scheduling*) [12]. *On-line scheduling* is an easy way to deal with real time events. But the solution quality is not very good due to the nature of these rules [20]. *Predictive-reactive scheduling* is another commonly used approach. In a *predictive-reactive scheduling* approach, an initial schedule is constructed and is revised to minimize the influence of real time events. Compared with *on-line scheduling*, it is able to obtain a schedule with high quality as well as better performance of the system. It is easy to perform simple schedule adjustments. Most of the definitions reported in the literature on dynamic scheduling refer to this category. In recent years, more and more meta-heuristic approaches, including genetic algorithm (GA) [29], particle swarm optimization (PSO) algorithm [30], artificial bee colony algorithm (ABC) [31], bat algorithm (BA) [32], brain storm optimization algorithm [33, 34] and generalized pigeon-inspired optimization algorithm [35], etc. have been widely applied. As to apply meta-heuristics on DJSSPs with new job arrivals, Fattahi and Fallahi [29] proposed a meta-heuristic algorithm based on GA and considered two objectives to balance efficiency and stability of the schedules. Kundakci and Kulak [36] introduced efficient hybrid GA methodologies for minimizing the makespan. Gao et al. [37] proposed a two-stage ABC algorithm to minimize the makespan. They extended the same problem to multiple objectives [38] including the minimization of makespan, the average of earliness and tardiness, and the maximum machine workload ($Mworkload$) and the total machine workload ($Tworkload$). Four ensembles of heuristics were proposed to solve this problem. Another extended work [39] was carried out to consider both uncertainty in the processing time and new jobs inserted. Fuzzy processing time was used to describe uncertainty in the processing time. Most recently, Gao et al. [13] developed a discrete Jaya algorithm to minimize instability and one of the following indices: makespan or total flow time or $Mworkload$ or $Tworkload$.

Although, there are limited research papers in the literature that address DJSSPs with new job arrivals. Dynamic scheduling is more urgent under the background of economic globalization. To adapt to the increasingly fierce market competition and to meet the rapidly changing market demand, there is still necessity to study this problem for devising faster rescheduling strategy and exploring effective solutions.

The rescheduling approach in this paper belongs to the group of *predictive-reactive scheduling* methods. Two issues should be addressed. One is the rescheduling strategy to deal with new jobs. The other is how to revise the

initial schedule properly. The simplest strategy is to insert new jobs to the end of the initial schedule and schedule them when all the machines are available. In this "*insertion in the end*" strategy, the initial schedule remains. But it is unsuitable for rush jobs. Instead, the "*right-shifting*" strategy can be applied on all the old jobs' operations that are not started yet. Obviously, this strategy may lead to poor performance of the initial schedule. "*Total rescheduling*" is another strategy to tackle rush job arrivals. Rescheduling operations include all new job operations and old jobs' operations that are not started yet. All rescheduling strategies in the above-mentioned literature fall into this category. Unfortunately, the stability of the initial schedule cannot be guaranteed in this strategy. In fact, only a part of the initial schedule should be modified sometimes, which is the main idea of match-up strategy [40]. Match-up strategy is regarded as an effective rescheduling strategy to handle real time events [20]. It can provide good results with both performance and stability. In match-up strategy, the rescheduling horizon within the initial schedule are defined firstly. A new scheduling problem is re-optimized during the rescheduling horizon. Then the rescheduling is integrated into the initial schedule. Moratori et al. [40, 41] have used match-up strategies to address the DJSSP with new job arrivals in a real world manufacturer. Both good performance and stability were obtained. But it was required that the operations of the initial schedule which have started before the arrival time of new jobs (termed *initialPoint*) must be completed before the rescheduling process begins. So that, the starting time of rescheduling horizon (termed $feasiblePoint$ in [41]) is normally later than *initialPoint*. However, it is not necessary in a job shop in some cases. As we know, in a job shop, process routes of jobs are not necessarily identical. If the relevant machine to the first operation of a new job is available when new job arrives, the rescheduling process can be activated once the new job arrives. In other words, the rescheduling process will not have to wait for completion of ongoing operations. Otherwise, they can be treated as machine unavailable constraints (MUC) in the rescheduling horizon. In this context, the main motivation of this paper is to pursue good performance and stability as well as fast response to new job arrivals in a job shop. We combine the *event-driven* rescheduling policy [20] with the match-up strategy. The rescheduling process is immediately triggered with ongoing operations remain once new jobs arrive. Consequently, the $feasiblePoint$ of rescheduling horizon is set to be equal to the *initialPoint*. The idle time of machines can be further utilized to some extent, so as to improve the performance of minimizing the makespan of the new job.

In previous studies, various regular and nonregular performance measures have been addressed. As for regular performance measures, they aim to finish all jobs as early as possible, i.e. the average or maximum job's completion times (makespan), tardiness, lateness, or total flow times. While some other special requirements are considered in nonregular performance measures, such as machine workload, instability, and so on. In recent years, more and more researches pursue both high efficiency and strong stability of schedules. In our work, three objectives are to be minimized to balance stability and efficiency. Besides the makespan deviation and sequence deviation, a novel nonregular performance measure is introduced with consideration of discontinuity rate of new jobs.

JSSPs have been proved to be strongly NP-hard, DJSSPs with new job arrivals are clearly NP-hard. As mentioned above, many meta-heuristics have been widely used. PSO algorithm is one of the famous heuristic algorithms based on swarm intelligence. It was first introduced by Kennedy and Eberhart [42]. PSO algorithm has been successfully applied to many continuous optimization problems because of its easy implementation and quick convergence [43]. Due to the discrete property, the original PSO algorithm cannot be used directly to solve DJSSPs. There are relatively fewer studies on PSO algorithm to solve DJSSPs. In this paper, an improved PSO algorithm is proposed. Improvement strategies include the modified decoding scheme, population initialization approach by designing a new transformation mechanism, and a novel particle movement method by introducing position changes and random inertia weight. The main contributions are as follows.

1. A novel nonregular performance measure is introduced in the mathematical model with consideration of discontinuity rate of new jobs. A mixed-integer programming with three objectives of pursuing performance and stability is established.

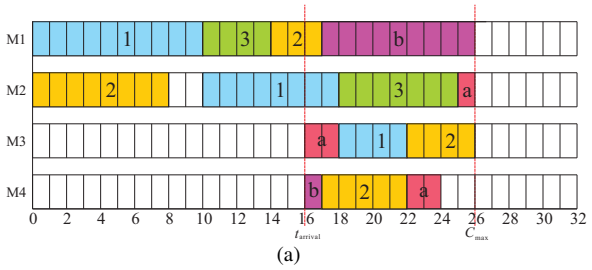
2. The *event-driven* rescheduling policy and the match-up strategy are combined to the rescheduling strategies.

3. An improved PSO algorithm is developed to solve the DJSSP with new job arrivals.

The rest of the paper is organized as follows. Section 2 describes the proposed problem and the developed mathematical model. Section 3 introduces the combined rescheduling strategy. Section 4 presents the improved algorithm in detail. Numerical experiments are presented in Section 5. Conclusions appear in Section 6.

2. Problem statement

The DJSSP addressed in this paper can be described as follows. There are N jobs (termed as *old jobs*) which have been scheduled on M machines. Each job consists of a predetermined sequence of operations. Each operation must be processed on the designated machines. Old jobs have a common due date which cannot be violated. N' new jobs may randomly arrive at the shop floor. The initial schedule has not been fully completed yet when new jobs arrive. The attributes of new jobs, such as the process routes, the processing time of each operation, due dates, and priorities are unknown until their arrivals. New jobs are intended to be processed efficiently without excessively delay. Meanwhile, too much changes of the initial schedule are not allowed. Therefore, our objective is to obtain a good performance for new jobs and stability for old jobs. For the former, in view of the random arrival times and different processing times of new jobs, regular performance measures such as makespan or flow time are not accurate enough to reflect a good performance for new jobs due to lack of efficiency. An example is illustrated in Fig. 1. Two new jobs arrive at the same time $t_{\text{arrival}} = 16$. They are completed also at the same time $C_{\max} = 26$. The total processing times of them are 5 and 10 units respectively. There is no difference either in makespan or in flow time between them. In fact, as shown in Fig. 1, job b is processed efficiently with no delay, while the processing of each operation of job a is not so continuous as job b in this schedule. Ideally, it is hoped that no discontinuity between the processing of each operation for each new job. For this, a novel nonregular performance measure named the discontinuity rate (DR) of new jobs is proposed, which is the ratio of the total delay time to the total processing time of new jobs. The sum of the total delay time and the total processing time of new jobs is the time interval between the arrival and the maximal completion time of new jobs. As for the high stability for old jobs, the makespan deviation (MD) of old jobs and the sequence deviation (SD) on machines are considered. MD can be defined as the relative rate of makespan of old jobs. SD considers changes to the relative order of operations in the initial and new schedule sequences, which is adapted from Abumaizar and Svestka [44] (see the mathematic model for details).



(a)

Fig. 1: An example of DR

The notations are given as follows.

Indices:

e, i: indices of old jobs;

e', i': indices of new jobs;

g, j: indices of machines;

k, l: indices of operations of jobs;

m: index of operations on machines;

Parameters:

OO_i: total number of operations of job i;

OM_j: total number of operations on machine j;

o_{ik}: operation k of job i;

$$\delta_{ikj} = \begin{cases} 1, & \text{if } o_{ik} \text{ is processed on machine } j; \\ 0, & \text{otherwise.} \end{cases}$$

p_{ij}: processing time of job i on machine j;

$C^{\sigma}_{\max}$: makespan of the initial schedule;

D: due date of old jobs;

H: a large enough positive number.

Variables:

S': minimum arrival time of new jobs;

S_{ik}: starting time of o_{ik};

C_{ik}: completion time of o_{ik};

t_{start}: start time of the rescheduling horizon;

t_{end}: end time of the rescheduling horizon;

$OO_{i\text{rd}}$: total number of operations to be rescheduled in the rescheduling horizon of job i;

$$\mu_{iej} = \begin{cases} 1, & \text{if job } i \text{ precedes job } e \text{ on machine } j \\ 0, & \text{otherwise.} \end{cases}$$

$$\omega_{igj} = \begin{cases} 1, & \text{if processing on machine } g \text{ precedes that on machine } j \text{ for job } i \text{ ;} \\ 0, & \text{otherwise.} \end{cases}$$

$$\eta_{mj} = \begin{cases} 1, & \text{if opertion } m{+}1 \text{ is not a successor of operation } m \text{ on machine } j; \\ 0, & \text{otherwise.} \end{cases}$$

Sets:

OP_{re}: the set of operations to be rescheduled.

The main assumptions are as follows.

1. Raw materials are enough and have been ready for both old and new jobs.
2. Each machine can process only one operation at any time.
3. An operation of a job can be processed on only one machine at any time, and it must be processed after its preceding operations are completed.
4. Preemption is not allowed. The processing of an operation cannot be interrupted, so if a job needs a machine which is occupied, it must wait until the machine is available.
5. Flexible routing for jobs is not considered.
6. The setup time of any operation is independent and included in the processing time.
7. The transportation time is ignored.

Based on the above notations and assumptions, the considered DJSSP with new job arrivals is formulated as follows.

$$\text{Min } DR = \frac{(\max\limits_{1\le i'\le N'}\max\limits_{1\le k\le OO_{i'}} C_{i'k} - S') - \sum_{i'=1}^{N'}\sum_{k=1}^{OO_{i'}}\sum_{j=1}^{M}\delta_{i'kj}p_{i'j}}{\sum_{i'=1}^{N'}\sum_{k=1}^{OO_{i'}}\sum_{j=1}^{M}\delta_{i'kj}p_{i'j}} \tag{1}$$

$$\text{Min } MD = \frac{\max\limits_{1\le i\le N}\max\limits_{1\le k\le OO_i} C_{ik} - C^{\sigma}_{\max}}{C^{\sigma}_{\max}} \tag{2}$$

$$\text{Min } SD = \frac{1}{M}\sum_{j=1}^{M}\sum_{m=1}^{OM_j}\frac{\eta_{mj}}{OM_j - 1} \tag{3}$$

Subject to

$$C_{ik} \le D, \forall i = 1,2,\ldots,N;\ k = 1,2,\ldots,OO_i; \tag{4}$$

$$C_{ik} = S_{ik}\delta_{ikj} + p_{ij}, \forall i = 1,2,\ldots,N;\ j = 1,2,\ldots,M;\ k = 1,2,\ldots,OO_i; \tag{5}$$

$$C_{i'k} = S_{i'k}\delta_{i'kj} + p_{i'j}, \forall i' = 1,2,\ldots,N';\ j = 1,2,\ldots,M;\ k = 1,2,\ldots,OO_{i'}; \tag{6}$$

$$\begin{gathered} S_{el}\delta_{elj} + H(1-\mu_{iej}) \ge S_{ik}\delta_{ikj} + p_{ij}, \forall e,i = 1,2,\ldots,N; \\ e \ne i;\ j = 1,2,\ldots,M;\ l = 1,2,\ldots,OO_e;\ k = 1,2,\ldots,OO_i; \end{gathered} \tag{7}$$

$$\begin{gathered} S_{e'l}\delta_{e'lj} + H(1-\mu_{i'e'j}) \ge S_{i'k}\delta_{i'kj} + p_{i'j}, \forall e',i' = 1,2,\ldots,N'; \\ e' \ne i';\ j = 1,2,\ldots,M;\ l = 1,2,\ldots,OO_{e'};\ k = 1,2,\ldots,OO_{i'}; \end{gathered} \tag{8}$$

$$\begin{gathered} S_{ik}\delta_{ikj} + H(1-\omega_{igj}) \ge S_{i(k-1)}\delta_{i(k-1)g} + p_{ig}, \forall i = 1,2,\ldots,N; \\ g,j = 1,2,\ldots,M;\ g \ne j;\ k = 1,2,\ldots,OO_i; \end{gathered} \tag{9}$$

$$\begin{gathered} S_{i'k}\delta_{i'kj} + H(1-\omega_{i'gj}) \ge S_{i'(k-1)}\delta_{i'(k-1)g} + p_{i'g}, \forall i' = 1,2,\ldots,N'; \\ g,j = 1,2,\ldots,M;\ g \ne j;\ k = 1,2,\ldots,OO_{i'}; \end{gathered} \tag{10}$$

$$S_{i'1} \ge t_{\text{start}}, \forall i = 1,2\ldots,N;\ j = 1,2,\ldots,M; \tag{11}$$

$$C_{i'k} \le t_{\text{end}}, \forall i' = 1,\ldots,N';\ k = 1,2,\ldots,OO_{i'}; \tag{12}$$

$$\mu_{iej} = 0 \text{ or } 1, \forall e,i = 1,2,\ldots,N;\ e \ne i;\ j = 1,2,\ldots,M; \tag{13}$$

$$\text{and } \mu_{i'e'j} = 0 \text{ or } 1, \forall e',i' = 1,2,\ldots,N';\ e' \ne i';\ j = 1,2,\ldots,M. \tag{14}$$

It is a multi-objective scheduling problem. In this formulation, the objective function (1) is to minimize the DR of new jobs, where $\max\limits_{1\le i'\le N'}\max\limits_{1\le k\le OO_{i'}} C_{i'k}$ is the makespan of new jobs. $\sum_{i'=1}^{N'}\sum_{k=1}^{OO_{i'}}\sum_{j=1}^{M}\delta_{i'kj}p_{i'j}$ is the total processing time of new jobs. $(\max\limits_{1\le i'\le N}\max\limits_{1\le k\le OO_{i'}} C_{i'k} - S') - \sum_{i'=1}^{N'}\sum_{k=1}^{OO_{i'}}\sum_{j=1}^{M}\delta_{i'kj}p_{i'j}$ is the total delay time of new jobs during the processing. DR reflects the process efficiency of new jobs. The objective function (2) minimizes the MDS of old jobs, where, $\max\limits_{1\le i\le N}\{\max\limits_{1\le k\le OO_i} C_{ik}\}$ is the makespan of old jobs after rescheduling. The objective function (3) is to minimize the SD on machines. MD and SD indicate the instability of initial schedule after rescheduling. Constraint (4) requires that the completion time of each operation of old jobs is equal to or less than the due date, which satisfies the requirement that the due date of old jobs cannot be violated. Constraints (5) and (6) are used to calculate the completion time of each operation. Constraints (7) and (8) are machine constraints indicating that each machine can execute one operation at any time. Constraints (9) and (10) are precedence constraints indicating that each operation can be executed only when its precedence operation was finished. Constraints (11) and (12) ensure that new jobs must be processed during the rescheduling horizon. The 0-1 restrictions for the decision variables are specified in constraints (13) and (14).

3. Rescheduling strategy

The focus of this section is to present the rescheduling strategy when new jobs arrive. For simplicity, it is assumed that only a job arrives at a time. If simultaneous arrival of two or more jobs happens, different priorities are given to jobs. To pursue different performances simultaneously, the match-up strategy is used in this paper. As mentioned before, there are three main steps.

Step 1, Set the rescheduling horizon.

Step 2, Solve a new scheduling problem.

Step 3, Integrate the new schedule into the initial one.

The data of a JSSP in Kundakcı and Kulak [45] shown in Table 1 is applied to illustrate how to determine the rescheduling horizon. An initial schedule for three jobs was produced and presented by a Gantt chart in Fig. 2(a). The makespan is 30. Suppose a new job arrives at 16. The processing requirement of the new job is shown in Fig. 2(b).

Table 1: The example of a job shop scheduling problem in [45].

Jobs	Process routes	Processing times			
		M1	M2	M3	M4
1	M1 → M2 → M3 → M4	10	8	4	2
2	M2 → M1 → M4 → M3	8	3	5	4
3	M1 → M2 → M4 → M3	4	7	3	2

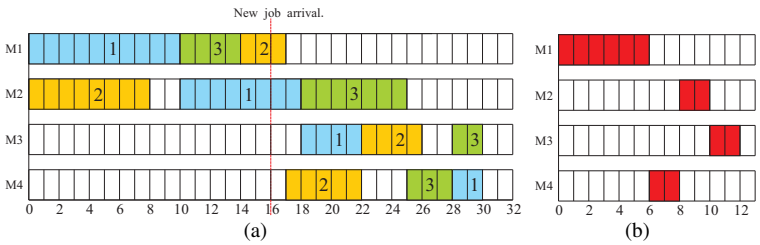


Fig. 2: An example of DJSSP. (a) initial schedule. (b) process routes of a new job.

The rescheduling process is triggered once a new job arrives. So the start time of the rescheduling horizon t_{start} is the arrival time of the new job. The idle time collection method proposed by Moratori et al. [41] is adapted to determine the end time of the rescheduling horizon t_{end}. There are four strategies to be discussed, the idle times are accumulated. The following strategies might be applied to collect idle times from t_{start}.

- Strategy 1: Collect all idle times on the required machines by the new job without regarding to the precedence constraints. The collected idle time is not necessary as a single time slot. Split idle times can be accumulated. The total idle time collected on each machine is equal to the processing time of each new operation (Fig. 3(a)).
- Strategy 2: Collect continuous idle times on each machine required for the new job. The time slot on the required machine must be large enough to contain the new operation (Fig. 3(b)).
- Strategy 3: As Strategy 1, but the precedence constraints of the new job are considered (Fig. 3(c)).
- Strategy 4: As Strategy 2, but the precedence constraints of the new job are considered (Fig. 3(d)).

Applying above strategies, the end time of the rescheduling horizon (t_{end}) is defined as the maximum of the completion time of the new job on each machine.

Fig. 3 demonstrates the idle time collection using four strategies to set the rescheduling horizon. When the new job arrives at $t_{\text{start}} = 16$, o_{22} on machine 1 and o_{12} on machine 2 are on-going operations. The on-going operations

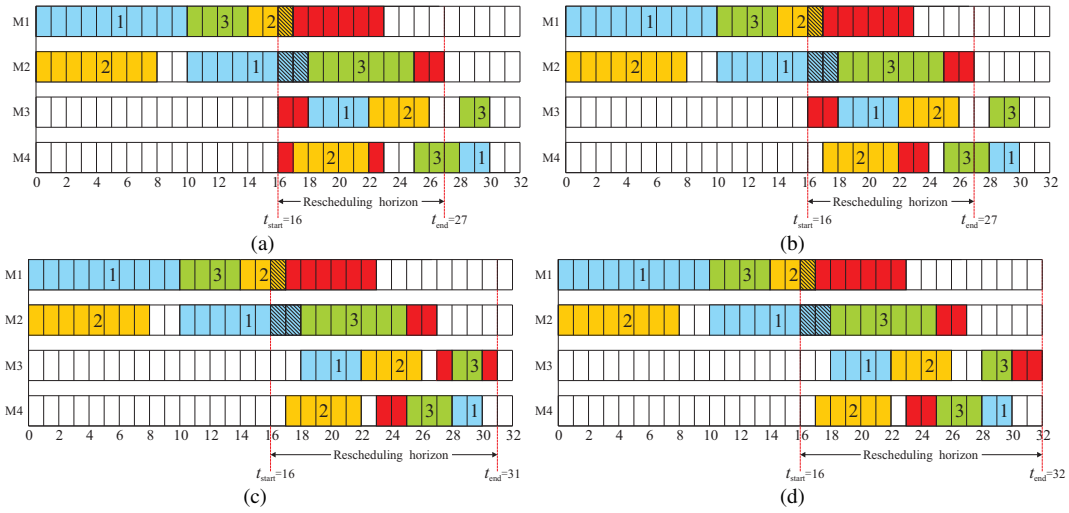


Fig. 3: Determination of the rescheduling horizon: (a) Strategy 1. (b) Strategy 2. (c) Strategy 3. (d) Strategy 4.

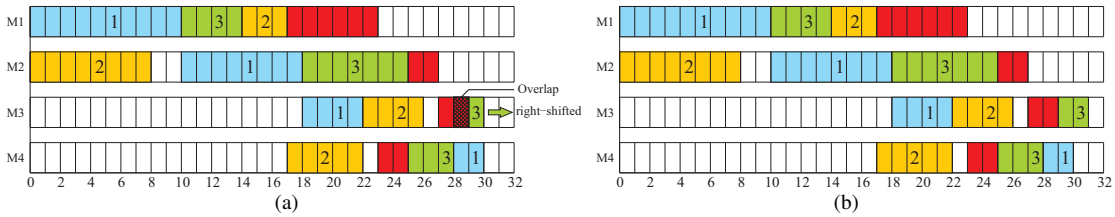


Fig. 4: Verification and repair of the solution: (a) overlap of the new job and job 3. (b) right-shifting policy.

on machines remain till they are finished. So rescheduling during the interval [16,17] on machine 1 and during the interval [16,18] is forbidden. They will be treated as MUC in the rescheduling horizon. The MUC is depicted in shadow in Fig. 3. The operations that are crossed by t_{start} will be excluded in the new scheduling problem. So do the operations crossed by t_{end}.

All operations during the rescheduling horizon, excluding the operations which are crossed by t_{start} and t_{end}, form a new rescheduling problem. As shown in Fig. 3(a) and (b), operations in the new rescheduling problem include o_{13}, o_{23}, o_{24}, o_{32} and all operations of the new job. o_{33} is not included in the new scheduling problem. A new schedule can be obtained by solving the new scheduling problem. The algorithm is proposed in detail in Section 4.

In the last step, the new schedule is matched up with the initial one. The partial schedule, contained within the rescheduling horizon of the initial schedule, is replaced by the new one. Some verification and repair of the solution may be necessary. For instance, *right-shifting* policy [40] can be used when operations overlaps appear after the integration. As shown in Fig. 4(a), there is an overlap between the last operation of the new job and job 3 on machine 3. So, the last operation of job 3 should be right-shifted in the final schedule, as shown in Fig. 4(b).

4. The improved PSO algorithm

In this section, an improved PSO algorithm is proposed to solve the considered problem. First, the original PSO algorithm is introduced. Next, the encoding and decoding scheme are described. Subsequently, the population initialization, the particle movement and the fitness function are proposed.

4.1. The original PSO algorithm

PSO algorithm is a population-based meta-heuristic algorithm in which each individual is called a particle and is defined as the potential solution of the problem to be optimized in a d-dimensional search space. In each generation

t, each particle's optimal historical location $P_{i\text{best}}(t) = (p_{i1\text{best}}(t), \ldots, p_{id\text{best}}(t))$ and the optimal location of all particles $G_{\text{best}}(t) = (g_{1\text{best}}(t), \ldots, g_{d\text{best}}(t))$ are combined to adjust the velocity of the components $V_i(t) = (v_{i1}(t), \ldots, v_{id}(t))$, which is then used to calculate the next particle position $X_i(t) = (x_{i1}(t), \ldots, x_{id}(t))$. The iteration formula of particle velocity and position are as follows.

$$v_{id}(t+1) = v_{id}(t) + c_1 \times rand_1 \times (p_{id\text{best}}(t) - x_{id}(t)) + c_2 \times rand_2 \times (g_{d\text{best}}(t) - x_{id}(t)), \tag{15}$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1). \tag{16}$$

Eq. (15) indicates that update of the velocity is determined by three components: inertia, personal influence and social influence. $v_{id}(t)$ is referred to the inertia, which makes the particle maintain its previous velocity and direction. $(p_{id\text{best}}(t) - x_{id}(t))$ is the personal influence (termed self-cognition) that leads the particle to the previous position which is better than the current. $(g_{d\text{best}}(t) - x_{id}(t))$ is social influence (termed social-cognition), which makes the particle follow the best neighbors' direction. c_1 and c_2 are acceleration coefficients determining the relative influence of self-cognition and social components, respectively. Normally, $c_1 = c_2 = c$. $rand_1$ and $rand_2$ are two uniformly distributed random numbers independently generated between 0 and 1.

The pseudo code for the original PSO algorithm is described as Algorithm 1.

Algorithm 1 Pseudo code for the original PSO algorithm.

Input: The related parameters of PSO algorithm.
Output: The best particle G_{best}.

t=0,$G_{\text{best}} = NULL$
for each particle P_i **do**
  Initialize the position X_i and velocity V_i randomly
end for
while $t <$ maximum iteration **do**
  for each particle P_i **do**
    Calculate the fitness value $fitness(P_i)$
    if $fitness(P_i) > fitness(P_{i\text{best}})$ **then**
      $P_{i\text{best}} = P_i$
    end if
    if $fitness(P_i) > fitness(G_{\text{best}})$ **then**
      $G_{\text{best}} = P_i$
    end if
  end for
  for each particle P_i **do**
    Update the velocity and position according to Eqs. (15) and (16)
    Apply maximum velocity limitation and bound handling strategy
  end for
  $t = t + 1$
end while
return G_{best}

4.2. Encoding and decoding scheme

An appropriate encoding scheme is important for successful application of PSO algorithm to solve the considered DJSSP. The operation-based encoding which is commonly used for JSSPs, is adopted to represent a solution. A solution is encoded as an array of job indices to present operations in the rescheduling horizon. Each job appears in the string exactly $OO_{i\text{rd}}$ times and the length of the string is equal to the sum of all operations to be rescheduled.

As shown in Fig. 3(a), if a new job is denoted as job 4, operations in the rescheduling horizon are o_{13}, o_{23}, o_{24}, o_{32}, o_{41}, o_{42}, o_{43}, and o_{44}, respectively. The total numbers of operations to be rescheduled are 1, 2, 1, and 4 respectively for jobs 1 to 4. So the length of the string is 1+2+1+4=8. Suppose a solution is encoded as [4 3 4 4 2 1 2 4]. Each integer in the string uniquely indicates an operation and can be determined according to its order of occurrence in the sequence.

Fig. 5: Encoding scheme

So the solution can be translated into a unique list of ordered operations of $[o_{41}, o_{32}, o_{42}, o_{43}, o_{23}, o_{13}, o_{24}, o_{44}]$ as shown in Fig. 5.

The important feature of this encoding scheme is that any permutation of the string always yields a feasible schedule. The procedure of transferring the string into a feasible schedule is called decoding. With regarded to the decoding scheme, three types of schedules are often applied: *semi-active schedule*, *active schedule*, and *non-delay schedule* [46]. In *semi-active schedules*, a schedule uniquely is obtained from a consistently complete selection by sequencing operations as early as possible. No operation can be started earlier without altering the sequences. To improve the makespan, a *semi-active schedule* can be modified into an *active schedule* by shifting an operation to the left without delaying other jobs. If no machine is kept idle while an operation is waiting for processing, the schedule is a *non-delay schedule*. It is noticeable that a feasible schedule has to be active but the reverse is not necessarily true.

Within the limitations of MUC on machine 1 and 2 in Fig 3(a), the solution [4 3 4 4 2 1 2 4] is transferred into a feasible schedule as shown in Fig. 6(a). Obviously, it is a *semi-active schedule*. Operations of job 2 on machine 3 and 4 can be shifted to the left without delaying other jobs. Consequently, an *active schedule* is obtained as in Fig. 6(b). It is also a *non-delay schedule* because of no idle times on any machines while an operation is waiting for processing.

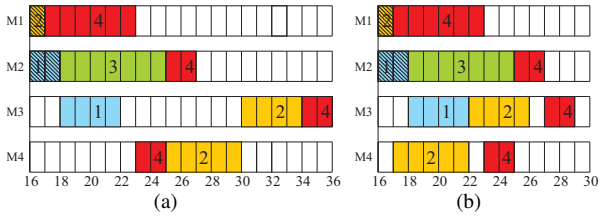


Fig. 6: Decoding scheme: (a) a semi-active schedule. (b) a non-delay schedule.

A widely used method to generate an *active schedule*, has been developed by Giffler and Thompson, termed GT algorithm [47]. In this paper, the GT algorithm is adapted to consider with the MUC in the rescheduling horizon. The decoding algorithm is described as Algorithm 2.

Algorithm 2 Decoding algorithm

Input: OP_{re}, the processing information of OP_{re} (including process routes and processing times), t_{start} and MUC.
Output: An active schedule.

while $OP_{\text{re}} \neq \Phi$ **do**
  Choose the first operation in the set OP_{re} and identify the required machine and processing time for it, termed m^* and p^*, respectively.
  Determine the earliest starting time for the chosen operation, termed s^*.
  if There is MUC on machine m^* **then**
    The starting time s^* is the upper boundary of the MUC
  else
    The starting time s^* is t_{start}
  end if
  Determine the earliest completion time of the chosen operation: $s^* + p^*$
  Add the scheduled time interval into MUC
  Delete the scheduled operation from OP_{re}
end while
return An active schedule

4.3. Population initialization

As mentioned above, the original PSO algorithm was used to solve continuous optimization problems. Due to the discrete solution spaces of JSSPs, the population initialization of original PSO algorithm should be modified from a continuous space to a discrete one. A real-integer encoding approach can be used to change a real potential solution into an integral one.

For illustrative purposes, some necessary functions are introduced as follows.

- $ceil(X)$. For an array X, it rounds the elements of X to its nearest integers towards infinity.
- $unidrnd(N)$. It returns a random number following the discrete uniform distribution with maximum N.
- $size(A, 2)$. For an m-by-n matrix A, it returns the number of columns, that is equal to n.
- $sum(A(2, :))$. For an m-by-n matrix A, it is the sum of the elements in the second row.

The search space for a traditional JSSP with N jobs and M machines consists of $N * M$ dimensions. Each job consists of sequential operations, which are normally equal to M. It is easy to transform integer series to the job index by the *ceil* function and returns an integer vector made of rounded up elements in the job index (see Algorithm 4 and Fig. 7).

However, numbers of operations of jobs in the rescheduling horizon are not necessarily the same and may vary greatly. For this, a novel transformation mechanism from a continuous domain to a discrete space is designed. A two-dimensional array is defined to express the total number of operations of rescheduling jobs (termed JON). The first line denotes the index of all jobs to be rescheduled, while the second line is the corresponding number of operations. For example, JON=[1,2,3,4;1,2,1,4] for the solution in Fig. 5.

The process of population initialization is described as Algorithm 3.

Algorithm 3 Population initialization

Input: The size of the population and JON.
Output: The initial population.

```
for each particle P_i do
    for j=1 to JON(j,1) do
        while JON(j,2) ≠ 0 do
            val ← unidrnd(JON(j,2))
            X_i ← val
            JON(j,2) ← JON(j,2) − 1
        end while
    end for
end for
return The initial population
```

4.4. Particle movement

Particle movement directly affects the performance of the PSO algorithm. Since the PSO algorithm was proposed in 1995, many improvements have been developed in the existing studies [48–51]. The earliest one of which is the introduction of an inertial weight ω by Shi and Eberhart [48] to improve the local search ability during the later stage of the algorithm. The inertia weight determines the contribution rate of a particle's previous velocity to its velocity at the current iteration and significantly affects the convergence and exploration-exploitation trade-off. Particle's movement can be expressed as

$$v_{id}(t+1) = \omega * v_{id}(t) + c_1 \times rand_1 \times (p_{id\text{best}}(t) - x_{id}(t)) + c_2 \times rand_2 \times (g_{d\text{best}}(t) - x_{id}(t)). \tag{17}$$

Since then, more studies focus on inertial weight improvements [52]. However, in most of improved PSO algorithms, each particle learns both from its own historical best position and the global best position. In our previous work [53], position changes are also included into the update formula of particle's velocity. Moreover, a random

inertia weight is adopted following the idea of Clerc [49]. The update of particle's velocity and position is described as follows.

$$v_{id}(t+1) = \omega * v_{id}(t) + c_1 \times rand_1 \times (p_{id\text{best}}(t) - 2x_{id}(t) + x_{id}(t-1)) + c_2 \times rand_2 \times (g_{d\text{best}}(t) - 2x_{id}(t) + x_{id}(t-1)), \quad (18)$$

$$x_{id}(t+1) = x_{id}(t) + v_{id}(t+1), \quad (19)$$

$$\omega = \mu + \sigma * N(0,1), \quad (20)$$

$$\text{and } \mu = \mu_{\min} + (\mu_{\max} - \mu_{\min}) * \text{rand}(0,1). \quad (21)$$

where ω is the inertia weight and follows a normal distribution. It determines the inheritance of the current velocity. σ is the variance of ω. μ_{max} and μ_{min} are the maximum and minimum of its mathematical expectation. Another problem to be solved is to re-map the updated position. Particles have been mapped to the discrete domain during the initialization phase, including the position and the velocity. The above velocity update leads to regaining into a continuous domain. Therefore a modified mechanism is necessary. A novel modified mechanism is proposed in this paper. Firstly, in view of the continuous property of the particles after updating, a ranked-order value (ROV) rule based on random key technique [54] is applied to convert continuous position values of particles to integer permutations. Then the *ceil* function is used to obtain an integer vector made of rounded up elements in the job index. The next crucial step is to modify occurrence times (OC) of each job index to accord with the number of operations to be rescheduled, which can be found in the second line in JON.

The process of particle movement is proposed as Algorithm 4. An example is depicted in Fig. 6.

Algorithm 4 Particle's movement

Input: The size of the swarm, JON, V_i, X_i, and G_{best}.
Output: The updated swarm.

for each particle P_i **do**
  Initialize a temporary variable $Temp$ and a temporary array $Temparray$: $Temp = 0, Temparray = NULL$.
  Update V_i using Eq. (18)
  Update X_i using Eq. (19)
  Convert X_i into an integer permutation by ROV rule
  $Temparray \leftarrow ceil\ (X_i * size(JON, 2)/sum(JON(2, :)))$
  for j=1 to $JON(j, 1)$ **do**
    while occurrence times OC(j) in $Temparray \neq JON(j, 2)$ **do**
      if OC(j) > $JON(j, 2)$ **then**
        Replace j in $Temparray$ with $Temp$
        OC(j) ← OC(j)-1
      else
        Replace $Temp$ in $Temparray$ with j
        OC(j) ← OC(j)+1
      end if
    end while
  end for
end for
return The updated swarm

4.5. Fitness function

In order to evaluate each particle, a fitness function is defined for the considered problem. Each particle should be evaluated for the combination of three objective functions described in Section 2. Weighted sum method which is the simplest method to transform multiple objectives into one objective is adopted in this paper. The fitness function is formulated as follows.

$$Fitness = \alpha_1 \times f_{\text{DR}} + \alpha_2 \times f_{\text{SD}} + \alpha_3 \times f_{\text{MD}}, \quad (22)$$

$$\alpha_1 + \alpha_2 + \alpha_3 = 1, 0 \leq \alpha_1, \alpha_2, \alpha_3 \leq 1. \quad (23)$$

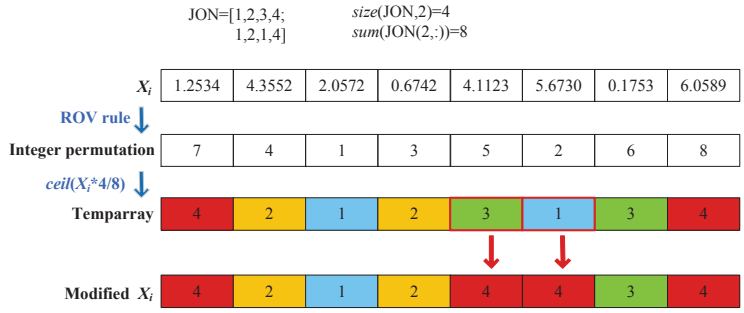


Fig. 7: An example of modified mechanism for particle's movement

where, f_{DR}, f_{SD}, and f_{MD} are the normalized objective functions of DR, SD and MD, respectively. $\alpha_1, \alpha_2, \alpha_3$ are their corresponding weights.

As we know, the value of different objective functions may have different order of magnitude. In order to apply weighted sum method, we normalize each objective function using the following equations [55]:

$$f_{\mathrm{DR}} = \frac{\mathrm{DR}(X)}{\max_{Y\in\Psi}\{|\mathrm{DR}(Y)|\}}, f_{\mathrm{SD}} = \frac{\mathrm{SD}(X)}{\max_{Y\in\Psi}\{|\mathrm{SD}(Y)|\}}, f_{\mathrm{MD}} = \frac{\mathrm{MD}(X)}{\max_{Y\in\Psi}\{|\mathrm{MD}(Y)|\}}. \tag{24}$$

where, Ψ is the set of particles in the current swarm.

To sum up, the flowchart of the improved PSO algorithm is shown in Fig. 8.

5. Experiments and discussions

In this section, computational experiments are carried out to investigate the effectiveness of the proposed rescheduling strategies and the performance of the improved PSO algorithm. All algorithms are coded in Matlab R2019a, and run on a 1.9-GHz Intel Core i3 processor with 4GB RAM.

5.1. Test instances

Firstly, initial schedules are generated from benchmark problem instances of the traditional JSSPs with $N \in \{6, 10, 15, 20, 30, 50, 100\}$ and $M \in \{5, 6, 10, 15, 20\}$, including FT06, FT10, LA1, LA6, LA21, LA26, LA31, LA36, TD61 and TD71. Makespan criterion of the initial schedules is indicated as $C^{\sigma}_{\max}$. The old jobs due dates are set as $(1+\theta)C^{\sigma}_{\max}$, where, θ is a relaxation factor, which is generated uniformly in the interval of [0.3,0.8].

Then, new jobs are randomly generated. For each problem instance, new jobs can be characterized by three parameters: arrival time, the number of operations and the processing times. New job arrives randomly. Similar to Moratori et al. [41], arrival time of new jobs $t_{\mathrm{arrival}} \in \{beginning, middle, end\}$, where *beginning*, *middle* and *end* refer to t_{arrival} equal to 20%, 50% and 80% of $C^{\sigma}_{\max}$, respectively. The reason for considering t_{arrival} follows the observation that the workload of the shop floor varies at different times in the schedule. For each new job i', the number of operations $OO_{i'}$ is not necessarily equal to M as the traditional JSSPs, but is an integer randomly generated from [1,M]. The corresponding machine to process each operation is randomly chosen from M machines.

For small instance FT06, six kinds of new jobs are shown in Table 2. Different job sizes are indicated by the number of operations (Num.), the longest processing time (LPtime), the shortest processing time (SPtime), the total processing time (TPtime) and process routes (Route). For other instance, job size is indicated by the number of operations. Three kinds are considered: *small*, *medium* and *large*. Without loss of generality, it can take values from the set $\{2, \lceil 0.5 * M\rceil, M\}$, where, the symbol $\lceil 0.5 * M\rceil$ denotes the nearest integer greater than or equal to $0.5 * M$. For instances, $\lceil 2.5\rceil = 3$. The processing time of job i' on machine j, $p_{i'j}$, is randomly generated from a uniform distribution U(1,100).

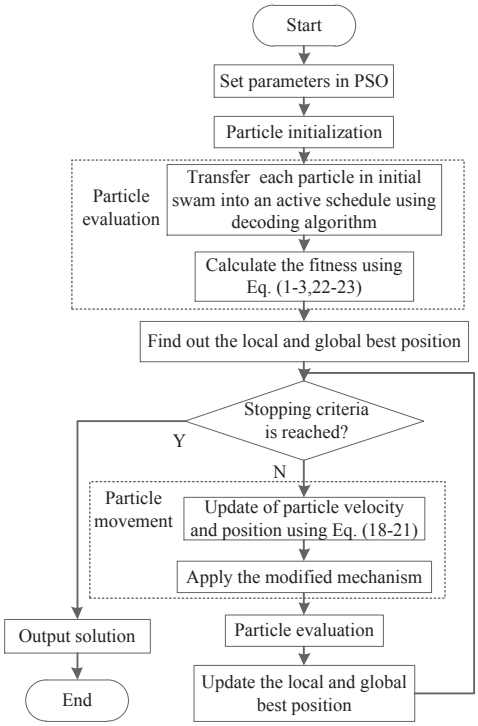


Fig. 8: Flowchart of the improved PSO algorithm

Table 2: New jobs' information.

New job index	Num.	LPtime	SPtime	TPtime	Route
J1	1	5	5	5	M5
J2	2	2	2	4	M4→ M6
J3	3	4	1	9	M3→ M6→ M5
J4	4	4	2	13	M4→ M2→ M5→ M1
J5	5	5	2	18	M1→ M4→ M5→ M6→ M3
J6	6	1	5	17	M5→ M3→ M6→ M4→ M2→ M1

Table 3: Reasonable parameter values.

Parameter	Value				
	1	2	3	4	5
c	2.0	2.5	3.0	3.5	4.0
σ	0.05	0.10	0.15	0.20	0.25
$\mu_{\max}$	0.8	0.9	1.0	1.1	1.2
$\mu_{\min}$	0.2	0.3	0.4	0.5	0.6
$Size$	10	20	30	40	50
$Iter$	10	50	100	150	200

5.2. Parameter setting

As shown in Section 3, the improved PSO algorithm has six parameters: $c, \sigma, \mu_{\max}, \mu_{\min}$, the swarm size ($Size$) and the maximum iteration ($Iter$). Taguchi method of design-of-experiments (DoE) [56] is used to determine these parameters on several preliminary tests. Accordingly, three different test problems FT06, FT10 and LA6 are considered. Meanwhile, five reasonable values for each parameter are selected, which can be found in Table 3. The interaction between the parameters is assumed to be negligible. The orthogonal array $L_{25}(5^6)$ is chosen. For each test problem and parameter combination, the improved PSO algorithm runs 20 times independently.

First, weights $\alpha_1, \alpha_2, \alpha_3$ in Eqs. (22) and (23) should be determined. After normalization, three objective functions DR, MD and SD have been converted into a single one using a simple weighted sum method in Eqs. (22) and (23). Weight represents the importance of each normalized objective function relative to the other ones. Among the three objective functions, the first one DR is the performance related to the new jobs, while MD and SD are the makespan stability and sequence stability, respectively, of the initial schedules. Both the new job and old jobs are given the same emphasis. Therefore, $\alpha_1 = \alpha_2 + \alpha_3 = 0.5$. Considering both the same importance of the makespan stability and sequence stability, $\alpha_2 = \alpha_3 = 0.25$.

The relative deviation (RD) of each fitness function value from the best fitness function value is considered with

$$RD = |\frac{F - FV_{Best}}{FV_{Best}}|. \tag{25}$$

In which, FV is the fitness function value of the test problem and FV_{Best} is the minimum fitness function value among the replications of the test problem. Applying Taguchi design analysis in $Minitab_{@}$, the significance rank of parameters is reported in Table 3 and the main effects plot for the means and the signal-to-noise ratios are shown in Fig. 9.

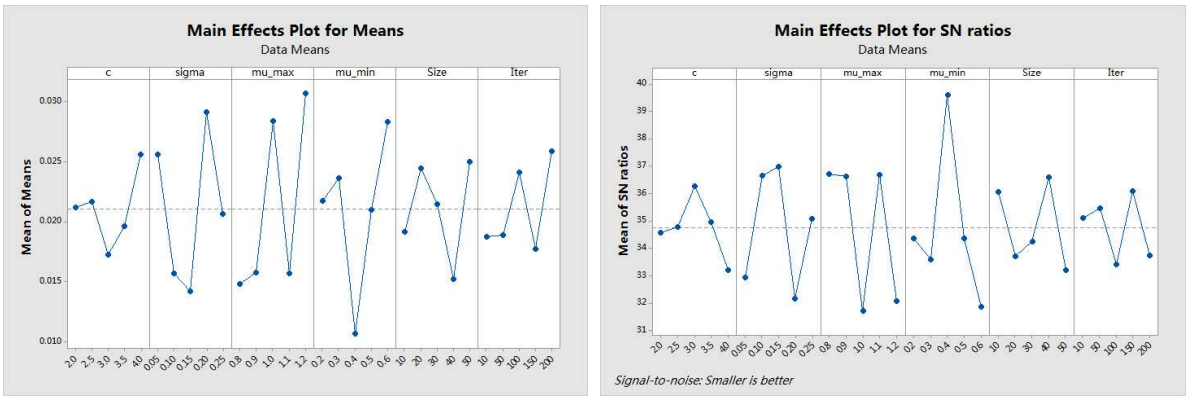
Table 4: The mean response values.

Level	c	σ	$\mu_{\max}$	$\mu_{\min}$	$Size$	$Iter$
1	0.02120	0.02560	0.01485	0.02170	0.01913	0.01873
2	0.02146	0.01573	0.01576	0.02363	0.02446	0.01889
3	0.01724	0.01419	0.02835	0.01066	0.02144	0.02409
4	0.01962	0.02911	0.01567	0.02096	0.01525	0.01773
5	0.02557	0.02046	0.03064	0.02831	0.02499	0.02584
Delta	0.00832	0.01492	0.01579	0.01765	0.00974	0.00812
Rank	5	3	2	1	4	6

According to Table 4, it can be noticed that $\mu_{\min}$ and $\mu_{\max}$ are two most important parameters. The next three parameters are σ, c and $Size$, respectively, while $Iter$ is less important. According to Fig. 9, the above parameters values are adopted as follows. $\mu_{\min} = 0.4$, $\mu_{\max} = 0.8$, $\sigma = 0.15$, $c_1 = c_2 = 3$, $Size = 40$, and $Iter = 150$.

5.3. Effectiveness of rescheduling strategies

To verify the effectiveness of rescheduling strategies Strategy 1 - Strategy 4 (S1-S4), we compare them with other five rescheduling strategies including four similar match up rescheduling strategies reported in Moratori and Petrovic



(a) (b)

Fig. 9: Main effects plot. (a) means. (b) signal-to-noise ratio.

[41] (S1M - S4M) and the total rescheduling (T). The main difference bewteen S and SM is that $feasiblePoint = initialPoint$ in S, while $feasiblePoint \geq initialPoint$ in SM (see Section 1 for details). In the total rescheduling T, $feasiblePoint = initialPoint$ is adopted.

We analyse the results by means of multifactor analysis of variance (ANOVA). Means and 95% confidence interval plots for $Fitness$, DR, MD and SD of one case (LA1, new job with 5 operations arriving at 20% of $C^{\sigma}_{\max}$) are plotted in Fig. 10. Clearly, S1-S4 perform better than S1M-S4M for all the objectives. The total rescheduling strategy are statistically non distinguishable from S1 and S2 for Fitness and from S2M for DR. But the total rescheduling strategy exhibit its weakness for MD and RD.

In addition, ANOVA is conducted to examine whether S1-S4 are statistically significantly different for each job size and each arrival time. Many ANOVAs are conducted to test for objectives values under several different situations. As shown in Tables 5 and 6, "Strategy*Arrival time" denotes the interaction between "Strategy and Arrival time". SOV refers to source of variant. Values in column F, P-value and F-crit are, Fisher statistics of the corresponding row effect, the probability of this value, and critical values of F, respectively. Effects with a P-value ≤ 0.05 are considered significant. Results indicate that DR, MD and SD values are influenced greatly by different arrival times and job sizes, but different strategies as well as pairwise interactions between three factors have no significant effect on the values. Overall, it is concluded that for the vast majority of cases, arrival times and job sizes are main independent factors, and there is no interaction effect between them, suggesting any one of the reschedule strategies can be chosen without considering the problem.

Due to space limitations, only the Gantt chart FT06 is described shown in Fig. 11. Fig. 11 (a) depicts the initial schedule, which is an optimal schedule. A new job with 3 operations arrives. S1 is used to determine the end time of the rescheduling horizon t_{end} as shown in Fig. 11 (b). The final new schedule is presented in Fig. 11 (c).

5.4. The performance of the improved PSO algorithm

To demonstrate the performance of the proposed PSO algorithm for solving DJSSPs when new jobs randomly arrived, five PSO variants are firstly utilized for comparison: linear decreasing inertia weight PSO (LDWPSO), PSO with stochastic inertia weight strategy (SIWPSO), second-order PSO (SecPSO) and PSO with self-adaptive inertia weight (SAPSO). The velocity update equations in each PSO algorithm are presented in Table 7. The parameters for five PSO variants are set to the same as the proposed PSO algorithm. Besides, it is compared with the optimal solution obtained by the CPLEX optimizer (version 12.9.0) as well as three state-of-the-art meta-heuristic algorithms in the literature, including hybrid artificial bee colony algorithm (HABC) [31], parallel bat algorithm (PBA) [32], and an adaptive scheduling algorithm (A-HEFT) [57]. The CPU time limits for each run on each problem instance is 60 s in CPLEX. The number of operations of each new job $OO_{i'}$ is set to be M. Arrival time is chosen as 20% of $C^{\sigma}_{\max}$. The parameters for these meta-heuristic algorithms are set in accordance with the corresponding literature. To be fair, the same encoding and decoding method as the proposed PSO algorithm are adopted.

Table 5: Results of ANOVA for each job size.

Job size	SOV	DR			MD			SD		
		F	P-value	F-crit	F	P-value	F-crit	F	P-value	F-crit
Small	Strategy	1.478	0.270	3.490	0.555	0.654	3.490	1.936	0.177	3.490
	Arrival time	57.021	≤0.001	3.885	28.083	≤0.001	3.885	24.377	≤0.001	3.885
	Strategy*Arrival time	0.761	0.614	2.996	0.305	0.922	2.996	0.444	0.835	2.996
Medium	Strategy	3.807	0.140	3.490	1.111	0.382	3.490	0.770	0.532	3.490
	Arrival time	84.809	≤0.001	3.885	218	≤0.001	3.885	108.177	≤0.001	3.885
	Strategy*Arrival time	2.946	0.0526	2.996	1.111	0.411	2.996	2.155	0.121	2.996
Large	Strategy	0.775	0.530	3.490	0.174	0.911	3.490	0.602	0.625	3.490
	Arrival time	129.831	≤0.001	3.885	10.333	0.002	3.885	72.372	≤0.001	3.885
	Strategy*Arrival time	0.775	0.605	2.996	0.174	0.978	2.996	1.549	0.243	2.996

Table 6: Results of ANOVA for each arrival time.

Arrival time	SOV	DR			MD			SD		
		F	P-value	F-crit	F	P-value	F-crit	F	P-value	F-crit
25%	Strategy	1.132	0.360	3.098	0.180	0.909	3.098	0.231	0.873	3.098
	Job size	3.808	0.018	2.866	3.358	0.0294	2.866	39.651	3.062E-09	2.866
	Strategy*Job size	1.389	0.249	2.278	0.214	0.996	2.278	1.111	0.402	2.277
50%	Strategy	1.473	0.252	3.098	0.571	0.640	3.098	1.057	0.389	3.098
	Job size	104.038	4.333E-13	2.866	125.821	7.062E-14	2.866	5.076	0.005	2.866
	Strategy*Job size	0.696	0.738	2.278	0.630	0.792	2.277	0.895	0.566	2.277
75%	Strategy	1.000	0.413	3.098	0.333	0.801	3.098	0.134	0.938	3.098
	Job size	9.427	0.00018	2.866	1223	1.372E-23	2.866	4.963	0.006	2.866
	Strategy*Job size	1.000	0.483	2.278	0.333	0.972	2.277	1.1633	0.369	2.277

Table 7: The velocity update equations in each PSO algorithm.

Algorithm	The velocity update equations
LDWPSO	Eq. (17), where, $\omega = \omega_{\max} - \frac{t*(\omega_{\max}-\omega_{\min})}{t_{\max}}$
SIWPSO	Eqs. (17), (20) and (21)
SecPSO	Eq. (18)
SAPSO	Eq. (17), where $\omega = \begin{cases} \omega_{\min} - \frac{(\omega_{\max}-\omega_{\min})*(f-f_{\min})}{(f_{avg}-f_{\min})}, & f \le f_{avg}; \\ \omega_{\max}, & \text{otherwise.} \end{cases}$ f is the objective function value

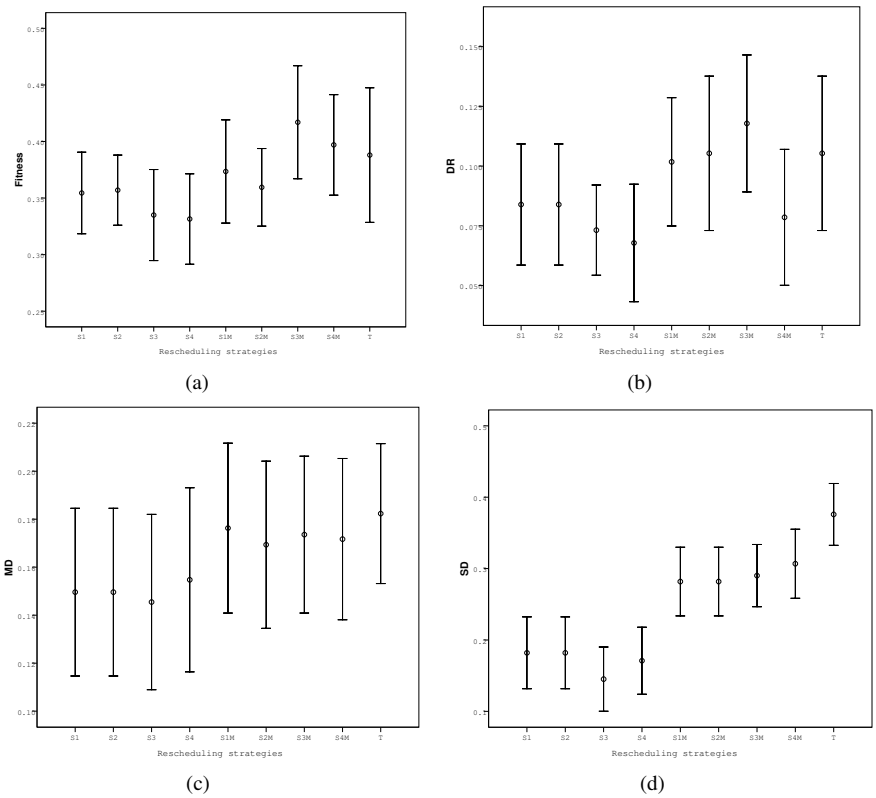


Fig. 10: Means and 95% confidence interval plots for Fitness (a), DR (b), MD (c), and SD(d).

Comparison results are listed in Tables 8-11. Table 8 reports the performances of six variants of PSO algorithms in term of the fitness value (Fitness) and average CPU times (Times) for rescheduling different of new jobs at different times by S1 on FT06. The average fitness value (μ) and standard deviation (σ_x) over 20 runs for each instance are evaluated. It is found that the performance of the proposed PSO algorithm outperforms other PSO variants in general. Furthermore, we can observe that Fitness values generally decrease as arrival times increase for all types of new jobs. It corresponds to the fact that the later the new job arrives, the less impact it has on the initial schedule. Tables 9-11 show the comparison results of the proposed PSO algorithm between the-state-of-the-art algorithm and CPLEX optimizer on other more instances with different new job sizes. Arrival time of new jobs are chosen as 20% of $C^{\sigma}_{\max}$. The symbol " -" means that instances were not solved to optimality within 60s. From Tables 9-11, it can be seen that the average fitness values obtained by the proposed PSO algorithm and HABC are closer to CPLEX than other algorithms for the first three instances. As for other instances, no optimal solutions can be obtained by CPLEX in the limited time, while the proposed PSO algorithm outperforms other algorithms in terms of both the average fitness value and standard deviation for all types of new job sizes in most instances.

Therefore, from the above comparison results, it can be concluded that the proposed PSO algorithm is an effective algorithm for solving DJSSPs when new jobs randomly arrived.

6. Conclusions

In this paper, we address the DJSSP when new jobs randomly arrived. A mixed-integer programming with three objectives is established. In order to realize quickly response for the new jobs, the *event-driven* rescheduling policy and four match-up strategies are combined in the rescheduling strategies. By comparison both to match-up strategies in the literature [41] and to total rescheduling strategy, the rescheduling strategies can perform better. Moreover, ANON results indicate that the four rescheduling strategies have no significant difference, so that any one of the

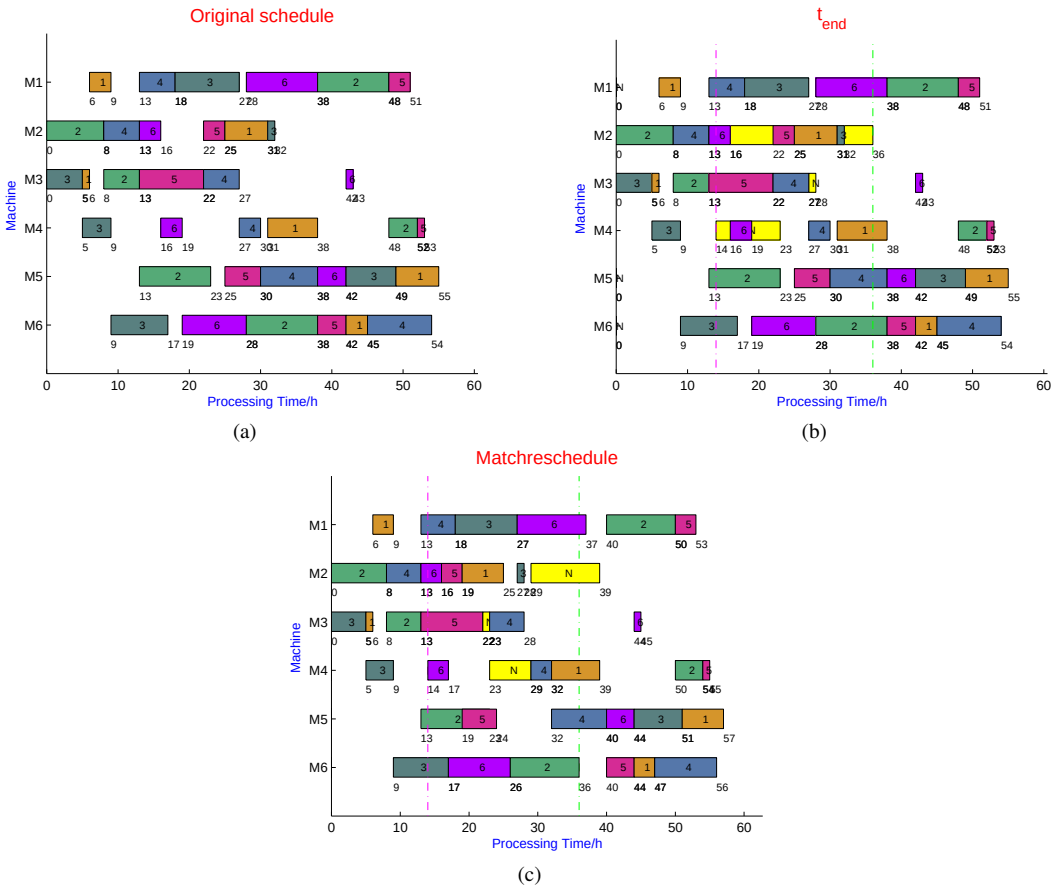


Fig. 11: Gantt chart for FT06 (a) initial schedule, (b) reschedule, (c) matchup schedule.

Table 8: The performances of six variants of PSOs in term of the fitness value and CPU time

Arrival Time	New job	OPSO			LDWPSO			SIWPSO			SecPSO			SAPSO			the proposed PSO		
		Fitness		Times	Fitness		Times	Fitness		Times	Fitness		Times	Fitness		Times	Fitness		Times
		μ	σ_x		μ	σ_x		μ	σ_x		μ	σ_x		μ	σ_x		μ	σ_x	
20%	J1	0.955	0.533	1.724	0.816	0.767	1.703	0.786	0.296	1.281	0.955	0.266	1.482	0.698	0.436	1.377	0.698	0.275	1.176
	J2	0.499	0.599	1.815	0.312	0.739	2.139	0.312	0.633	1.873	0.355	0.230	2.094	0.310	0.360	1.992	0.309	0.257	1.325
	J3	0.577	0.606	2.110	0.508	0.613	1.997	0.507	0.571	1.839	0.553	0.254	1.565	0.499	0.402	1.624	0.488	0.274	1.587
	J4	0.627	0.290	1.745	0.500	0.453	1.996	0.441	0.527	1.281	0.521	0.392	1.727	0.438	0.339	1.621	0.395	0.234	1.353
	J5	0.497	0.320	2.564	0.478	0.186	2.950	0.378	0.505	1.955	0.495	0.374	2.514	0.366	0.413	2.181	0.338	0.351	1.983
	J6	0.575	0.338	2.723	0.520	0.414	3.017	0.471	0.453	2.505	0.527	0.479	2.633	0.466	0.450	1.978	0.401	0.434	1.712
50%	J1	0.855	0.493	1.532	0.813	0.501	1.100	0.807	0.440	1.325	0.854	0.517	1.366	0.776	0.347	1.114	0.699	0.198	1.054
	J2	0.796	0.346	1.627	0.701	0.476	1.471	0.701	0.487	1.209	0.768	0.424	1.100	0.672	0.199	1.368	0.652	0.184	1.251
	J3	0.540	0.306	1.755	0.497	0.584	1.378	0.472	0.459	1.295	0.536	0.556	1.094	0.437	0.288	1.391	0.393	0.215	1.193
	J4	0.579	0.427	1.301	0.532	0.497	1.117	0.504	0.469	0.875	0.534	0.453	1.206	0.503	0.304	0.858	0.421	0.297	0.797
	J5	0.369	0.276	1.958	0.303	0.606	2.039	0.302	0.506	1.560	0.363	0.391	2.061	0.262	0.383	1.877	0.194	0.361	1.272
	J6	0.413	0.318	2.030	0.355	0.499	1.911	0.321	0.536	1.765	0.375	0.547	1.873	0.303	0.424	1.451	0.231	0.236	1.318
80%	J1	0.663	0.425	0.497	0.625	0.791	0.649	0.625	0.532	0.440	0.653	0.688	0.240	0.560	0.447	0.844	0.548	0.286	0.733
	J2	0.413	0.361	0.767	0.383	0.743	1.002	0.374	0.482	0.788	0.403	0.301	0.788	0.364	0.371	0.721	0.341	0.323	0.343
	J3	0.537	0.392	1.042	0.427	0.921	0.971	0.380	0.437	1.135	0.431	0.394	0.626	0.342	0.363	0.571	0.310	0.360	0.537
	J4	0.323	0.286	0.881	0.284	0.778	0.487	0.355	0.547	0.807	0.326	0.453	0.311	0.200	0.548	0.560	0.269	0.180	0.440
	J5	0.368	0.401	1.174	0.368	0.892	1.479	0.230	0.425	1.350	0.222	0.436	0.912	0.198	0.370	1.296	0.268	0.182	1.250
	J6	0.355	0.346	1.257	0.327	0.787	1.762	0.289	0.471	1.488	0.332	0.595	1.362	0.283	0.317	1.233	0.278	0.221	1.136

Table 9: Comparison results of the proposed PSO algorithm to the-state-of-the-art algorithm and CPLEX optimizer (small new job size).

Instance	CPLEX	the proposed PSO		HABC		A-HEFT		PBA	
		μ	σ_x	μ	σ_x	μ	σ_x	μ	σ_x
10×5 (LA01)	0.2732	0.3053	0.1295	0.3064	0.1175	0.3343	0.1814	0.3558	0.1382
15×5 (LA06)	0.3035	0.3467	0.0710	0.3365	0.1743	0.4386	0.1602	0.3890	0.2820
10×10 (FT10)	0.4242	0.4701	0.1337	0.4655	0.1361	0.5431	0.1540	0.5446	0.1940
15×10 (LA21)	-	0.4641	0.1207	0.5006	0.1690	0.5303	0.1507	0.5727	0.1878
20×10 (LA26)	-	0.4722	0.1822	0.5176	0.0988	0.5742	0.1874	0.5418	0.2653
30×10 (LA31)	-	0.5040	0.2074	0.5219	0.1267	0.5182	0.1960	0.5317	0.2374
15×15 (LA36)	-	0.4616	0.1834	0.4540	0.1081	0.4748	0.2444	0.4858	0.1506
50×20 (TD61)	-	0.5664	0.1768	0.6160	0.2560	0.5953	0.0825	0.5599	0.1118
100×20 (TD71)	-	0.6387	0.3757	0.6740	0.4350	0.6863	0.2485	0.6743	0.1660

Table 10: Comparison results of the proposed PSO algorithm to the-state-of-the-art algorithm and CPLEX optimizer (medium new job size).

Instance	CPLEX	the proposed PSO		HABC		A-HEFT		PBA	
		μ	σ_x	μ	σ_x	μ	σ_x	μ	σ_x
10×5 (LA01)	0.3872	0.3931	0.1008	0.3943	0.1471	0.5191	0.1798	0.4893	0.1657
15×5 (LA06)	0.4722	0.4880	0.1291	0.4825	0.1690	0.5192	0.1421	0.4984	0.1049
10×10 (FT10)	0.4608	0.4662	0.0274	0.4893	0.1214	0.6022	0.1485	0.5372	0.1194
15×10 (LA21)	-	0.5383	0.1632	0.5254	0.1176	0.6032	0.1174	0.5957	0.2095
20×10 (LA26)	-	0.4789	0.1529	0.5491	0.2022	0.4908	0.1909	0.5102	0.1418
30×10 (LA31)	-	0.5228	0.1615	0.5179	0.1133	0.5772	0.2173	0.5337	0.1887
15×15 (LA36)	-	0.5768	0.0880	0.6141	0.1103	0.6493	0.1325	0.6748	0.2116
50×20 (TD61)	-	0.6064	0.1037	0.6226	0.1393	0.5722	0.1869	0.6635	0.1465
100×20 (TD71)	-	0.7030	0.2899	0.7515	0.1938	0.6845	0.1597	0.7483	0.1539

Table 11: Comparison results of the proposed PSO algorithm to the-state-of-the-art algorithm and CPLEX optimizer (large job size).

Instance	CPLEX	the proposed PSO		HABC		A-HEFT		PBA	
		μ	σ_x	μ	σ_x	μ	σ_x	μ	σ_x
10×5 (LA01)	0.4449	0.4689	0.1020	0.4739	0.2277	0.5225	0.1595	0.4955	0.1282
15×5 (LA06)	0.6124	0.6182	0.0339	0.6178	0.0952	0.6275	0.1509	0.6255	0.1622
10×10 (FT10)	0.5318	0.5355	0.1300	0.5425	0.1758	0.5790	0.2324	0.5602	0.1704
15×10 (LA21)	-	0.5367	0.1134	0.6097	0.1636	0.6030	0.1815	0.5520	0.1460
20×10 (LA26)	-	0.5626	0.1488	0.5754	0.1725	0.5860	0.2738	0.5865	0.1267
30×10 (LA31)	-	0.5912	0.1267	0.5764	0.0539	0.6182	0.0959	0.6232	0.1549
15×15 (LA36)	-	0.6377	0.1975	0.6452	0.0687	0.6080	0.1774	0.6511	0.1933
50×20 (TD61)	-	0.5917	0.1723	0.6682	0.1196	0.6362	0.2140	0.6697	0.1240
100×20 (TD71)	-	0.7075	0.1447	0.7104	0.1634	0.7578	0.1518	0.7296	0.0986

reschedule strategies can be chosen. Besides, an improved PSO algorithm is developed to solve this problem. Improvement strategies consist of a modified decoding scheme, a population initialization approach by designing a new transformation mechanism, a novel particle movement method by introducing position changes and a random inertia weight. Extensive comparative experiments are conducted to examine the performance of the improved PSO algorithm on several instances. The experiment results demonstrate that the improved PSO algorithm is effective to solve DJSSP when new jobs randomly arrived. In the future, we will extend the proposed methods to flexible job shops and open shops. In addition, multi-objective evolutionary algorithms are another future research topic.

Acknowledgments

Firstly, the authors would like to thank all anonymous referees for their helpful comments that greatly improved the presentation and clarity of this work. Secondly, this research is supported by the Natural Science Foundation of China under Grant No. 61673228, 61703220 and 61402216, the Natural Science Foundation of Shandong Province under Grant No. ZR2010GM006.

References

[1] M. Stevenson, L. C. Hendry, B. G. Kingsman, A review of production planning and control: the applicability of key concepts to the make-to-order industry, International Journal of Production Research 43 (5) (2005) 869–898.

[2] K. Wang, L. Li, Y. Lan, P. Dong, G. Xia, Application Research of Chaotic Carrier Frequency Modulation Technology in Two-Stage Matrix Converter, Mathematical Problems in Engineering 2019 (11) (2019) 1–8.

[3] D. L. Bakuli, A survey of multi-objective scheduling techniques applied to the job shop problem (JSP), Applications of Management Science 12 (2006) 51–62.

[4] I. A. Chaudhry, A. A. Khan, A research survey: review of flexible job shop scheduling techniques, International Transactions in Operational Research 23 (3) (2015) 551–591.

[5] W. Liu, Y. Jin, M. Price, New scheduling algorithms and digital tool for dynamic permutation flowshop with newly arrived order, International Journal of Production Research 55 (11) (2017) 3234–3248.

[6] N. G. Hall, C. N. Potts, Rescheduling for new orders, Operations Research 52 (3) (2004) 440–453.

[7] E. Cakici, S. J. Mason, J. W. Fowler, H. N. Geismar, Batch scheduling on parallel machines with dynamic job arrivals and incompatible job families, International Journal of Production Research 51 (8) (2013) 2462–2477.

[8] D. Rahmani, M. Heydari, Robust and stable flow shop scheduling with unexpected arrivals of new jobs and uncertain processing times, Journal of Manufacturing Systems 33 (1) (2014) 84–92.

[9] P. Kunkun, Q. Pan, L. Gao, X. Li, S. Das, B. Zhang, A multi-start variable neigbourhood descent algorithm for hybrid flowshop rescheduling, Computers & Industrial Engineering 45 (2016) 92–112.

[10] Y. P. Fu, J. L. Ding, H. F. Wang, J. W. Wang, Two-objective stochastic flow-shop scheduling with deteriorating and learning effect in Industry 4.0, Applied Soft Computing 68 (2018) 847–855.

[11] Y. Fu, H. Wang, G. Tian, Z. Li, H. Hu, Two-agent stochastic flow shop deteriorating scheduling via a hybrid multi-objective evolutionary algorithm, Journal of Intelligent Manufacturing 30 (5) (2019) 2257–2272.

[12] I. Sabuncuoglu, M. Bayiz, Analysis of reactive scheduling problems in a job shop environment, European Journal of Operational Research 126 (3) (2000) 567–586.

[13] K. Z. Gao, F. J. Yang, M. C. Zhou, Q. K. Pan, P. N. Suganthan, Flexible job-shop rescheduling for new job insertion by using discrete jaya algorithm, IEEE Transactions on Cybernetics 49 (5) (2019) 1944–1955.

[14] Z. Zakaria, S. Petrovic, Genetic algorithms for match-up rescheduling of the flexible manufacturing systems, Computers & Industrial Engineering 62 (2) (2012) 670–686.

[15] C. A. Holloway, R. T. Nelson, Job shop scheduling with due dates and variable processing times, Management Science 20 (9) (1974) 1264–1275.

[16] R. Nelson, C. Holloway, R. M. L. Wong, Centralized scheduling and priority implementation heuristics for a dynamic job shop model, A I I E Transactions 9 (1) (1977) 95–102.

[17] R. Ramasesh, Dynamic job shop scheduling: a survey of simulation research, Omega 18 (1) (1990) 43–57.

[18] V. Suresh, D. Chaudhuri, Dynamic scheduling a survey of research, International Journal of Production Economics 32 (1) (1993) 53–63.

[19] G. E. Vieira, J. W. Herrmann, E. Lin, Rescheduling Manufacturing Systems: A Framework of Strategies, Policies, and Methods, Journal of Scheduling 6 (1) (2003) 39–62.

[20] D. Ouelhadj, S. Petrovic, A survey of dynamic scheduling in manufacturing systems, Journal of Scheduling 12 (4) (2009) 417–431.

[21] C. N. Potts, V. A. Strusevich, Fifty years of scheduling: a survey of milestones, Journal of the Operational Research Society 60 (1) (2009) S41–S68.

[22] M. Jatoth, L. Krishnanand, R. A. Neelakanteswara, A review of dynamic job shop scheduling techniques, in: 14th Global Congress on Manufacturing and Managemen, vol. 30, Elsevier, 34–39, 2019.

[23] J. Zhang, G. Ding, Y. Zou, S. Qin, J. Fu, Review of job shop scheduling research and its new perspectives under Industry 4.0, Journal of Intelligent Manufacturing 30 (2019) 1809–1830.

[24] A. P. Muhlemann, A. G. Lockett, C. K. Farn, Job shop scheduling heuristics and frequency of scheduling, International Journal of Production Research 20 (2) (1982) 227–241.

[25] F. C. R. Chang, Heuristics for dynamic job shop scheduling with real-time updated queueing time estimates, International Journal of Production Research 35 (3) (1997) 651–665.

[26] P. D. D. Dominic, S. Kaliyamoorthy, M. S. Kumar, Efficient dispatching rules for dynamic job shop scheduling, International Journal of Advanced Manufacturing Technology 24 (1-2) (2004) 70–75.

[27] M. S. Lu, R. Romanowski, Multicontextual dispatching rules for job shops with dynamic job arrival, International Journal of Advanced Manufacturing Technology 67 (1-4) (2013) 19–33.

[28] P. Sharma, A. Jain, Performance analysis of dispatching rules in a stochastic dynamic job shop manufacturing system with sequence-dependent setup times: simulation approach, CIRP Journal of Manufacturing Science and Technology 10 (4) (2015) 110–119.

[29] P. Fattahi, A. Fallahi, Dynamic scheduling in flexible job shop systems by considering simultaneously efficiency and stability, CIRP Journal of Manufacturing Science & Technology 2 (2) (2010) 114–123.

[30] Z. Wang, Y. Qi, H. Cui, J. Zhang, A hybrid algorithm for order acceptance and scheduling problem in make-to-stock/make-to-order industries, Computers & Industrial Engineering 127 (46) (2019) 841–852.

[31] L. Xixing, P. Zhao, D. Baigang, G. Jun, X. Wenxiang, Z. Kejia, Hybrid artificial bee colony algorithm with a rescheduling strategy for solving flexible job shop scheduling problems, Computers & Industrial Engineering 113 (2017) 10–26.

[32] T.-K. Dao, T.-S. Pan, T.-T. Nguyen, J.-S. Pan, Parallel bat algorithm for optimizing makespan in job shop scheduling problems, Journal of Intelligent Manufacturing 29 (2) (2018) 451–462.

[33] S. Cheng, Q. Qin, J. Chen, Y. Shi, Brain storm optimization algorithm: a review, Artificial Intelligence Review 46 (4) (2016) 445–458.

[34] S. Cheng, Y. Shi, Brain Storm Optimization Algorithms: Concepts, Principles, and Applications, vol. 23, Springer International Publishing AG., ISBN 978-3-030-15069-3.3257227892, 2019.

[35] S. Cheng, X. Lei, H. Lu, Y. Zhang, Y. Shi, Generalized pigeon-inspired optimization algorithms, Science China Information Sciences 62 (7)

(2019) 070211:1–070211:3.
[36] N. Kundakcı, O. Kulak, Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem, Computers & Industrial Engineering 96 (2016) 31–51.
[37] K. Z. Gao, P. N. Suganthan, T. J. Chua, C. S. Chong, T. X. Cai, Q. K. Pan, A two-stage artificial bee colony algorithm scheduling flexible job-shop scheduling problem with new job insertion, Expert Systems with Applications An International Journal 42 (21) (2015) 7652–7663.
[38] K. Z. Gao, P. N. Suganthan, M. F. Tasgetiren, Q. K. Pan, Q. Q. Sun, Effective ensembles of heuristics for scheduling flexible job shop problem with new job insertion, Computers & Industrial Engineering 90 (2015) 107–117.
[39] K. Z. Gao, P. N. Suganthan, Q. K. Pan, M. F. Tasgetiren, A. Sadollah, Artificial bee colony algorithm for scheduling and rescheduling fuzzy flexible job shop problem with new job insertion, Knowledge-Based Systems 109 (2016) 1–16.
[40] P. Moratori, S. Petrovic, J. Vázquez-Rodríguez, Integrating rush orders into existent schedules for a complex job shop problem, Applied Intelligence 32 (2) (2010) 205–215.
[41] P. Moratori, S. Petrovic, Match-up approaches to a dynamic rescheduling problem, International Journal of Production Research 50 (1) (2012) 261–276.
[42] J. Kennedy, R. Eberhart, Particle swarm optimization, in: IEEE International Conference on Neural Networks, IEEE, 1942–1948, 1995.
[43] M. R. Bonyadi, Z. Michalewicz, Particle swarm optimization for single objective continuous space problems: a review, Evolutionary Computation 25 (1) (2017) 1–54.
[44] R. J. Abumaizar, J. A. Svestka, Rescheduling job shops under random disruptions, International Journal of Production Research 35 (7) (1997) 2065–2082.
[45] N. Kundakcı, O. Kulak, Hybrid genetic algorithms for minimizing makespan in dynamic job shop scheduling problem, Computers & Industrial Engineering 96 (2016) 31–51.
[46] M. Pinedo, Scheduling Theory, Algorithms, and Systems, Prentice Hall, 2002.
[47] B. Giffler, G. L. Thompson, Algorithms for solving production-scheduling problems, Operations Research 8 (4) (1960) 487–503.
[48] Y. Shi, R. Eberhart, A modified particle swarm optimizer, in: IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence, IEEE, 69–73, 1998.
[49] M. Clerc, The swarm and the queen: towards a deterministic and adaptive particle swarm optimization, in: Proceedings of the 1999 Congress on Evolutionary Computation-CEC99, IEEE, 1951–1957, 1999.
[50] M. Clerc, J. Kennedy, The particle swarm-explosion, stability and convergence in a multi dimensional complex space, IEEE Transactions on Evolutionary Computations 6 (2) (2002) 58–73.
[51] Q. Ma, X. Lei, Q. Zhang, Mobile robot path planning with complex constraints based on the second-order oscillating particle swarm optimization algorithm, in: 2009 WRI World Congress on Computer Science and Information Engineering, IEEE, 244–248, 2009.
[52] J. C. Bansal, P. K. Singh, M. Saraswat, A. Verma, S. S. Jadon, A. Abraham, Inertia weight strategies in particle swarm optimization, in: 2011 Third World Congress on Nature and Biologically Inspired Computing, IEEE, 633–640, 2011.
[53] Z. Wang, J. H. Zhang, Y. Q. Qi, Job shop scheduling method with idle time in cloud manufacturing, Control and Decision 32 (5) (2017) 811–816.
[54] J. C. Bean, Genetic algorithms and random keys for sequencing andoptimization, ORSA Journal on Computing 6 (2) (1994) 154–160.
[55] Y. W. Leung, Y. Wang, Multiobjective programming using uniform design and genetic algorithm, IEEE Transactions on Systems Man & Cybernetics Part C 30 (3) (2000) 293–304.
[56] G. Taguchi, M. S. Phadke, Quality Engineering through Design Optimization, in: IEEE Global Telecommunications Conference, GLOBECOM'84: Communications in the Information Age., IEEE, 1106–1113, 1984.
[57] Z. Cao, L. Zhou, B. Hu, C. Lin, An adaptive scheduling algorithm for dynamic jobs for dealing with the flexible job shop scheduling problem, Business & Information Systems Engineering 61 (3) (2019) 299–309.