International Journal of Industrial Engineering Computations 14 (2023) 539–554

Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

Joint optimization of production and maintenance scheduling for unrelated parallel machine using hybrid discrete spider monkey optimization algorithm

Yarong Chen[a,b], Liuyan Zhong[a], Chunchun Shen[a], Jabir Mumta[a] and Fuh-Der Chou[a*]

[a]*School of Mechanical and Electronic Engineering, Wenzhou University, Wenzhou, 325035, Zhejiang, China*
[b]*School of Mechanical Science and Engineering, Huazhong University of Science and Technology, China*

C H R O N I C L E

Article history:
Received January 21 2023
Received in Revised Format March 5 2023
Accepted April 4 2023
Available online April, 4 2023

Keywords:
Unrelated parallel machine scheduling
Hybrid discrete spider monkey optimization
Mixed integer programming model
Variable maintenance
Makespan

A B S T R A C T

This paper considers an unrelated parallel machine scheduling problem with variable maintenance based on machine reliability to minimize the maximum completion time. To obtain the optimal solution of small-scale problems, we firstly establish a mixed integer programming model. To solve the medium and large-scale problems efficiently and effectively, we develop a hybrid discrete spider monkey optimization algorithm (HDSMO), which combines discrete spider monkey optimization (DSMO) with genetic algorithm (GA). A few additional features are embedded in the HDSMO: a three-phase constructive heuristic is proposed to generate better initial solution, and an individual updating method considering the inertia weight is used to balance the exploration and exploitation capabilities. Moreover, a problem-oriented neighborhood search method is designed to improve the search efficiency. Experiments are conducted on a set of randomly generated instances. The performance of the proposed HDSMO algorithm is investigated and compared with that of other existing algorithms. The detailed results show that the proposed HDSMO algorithm can obtain significantly better solutions than the DSMO and GA algorithms.

© 2023 by the authors; licensee Growing Science, Canada

1. Introduction

The production scheduling problem encompasses various branches, and one of the significant ones is the parallel machine scheduling problems (PMSPs). In the literature, PMSPs are typically classified to three groups: identical parallel machines (Pm), uniform parallel machines (Qm) and unrelated parallel machines (Rm) (Cheng & Sin, 1990). The category of unrelated PMSPs (UPMSPs) encompasses a broader scope than the other two categories, as it involves various machines carrying out identical tasks but with varying processing abilities or capacities. With the development of intelligent manufacturing industries, such as computerized numerical control machine tools, industrial robots, unrelated parallel machine production has become the most common operating mode of enterprises. Solving real-life UPMSPs is a major challenge for industrial experts and researchers, not only because they are mostly NP-hard but also, more importantly, because of the special characteristics or requirements they have in practice.

Abundant research has been conducted to address UPMSPs with different production environments, performance measures and solution methods (Pfund, Fowler, & Gupta, 2004). Azizoglu & Kirca (1999) and Lin et al. (2011) addressed the basic UPMSP. Rodriguez et al. (2013) and Wang & Alidaee (2019) studied the large-scale UPMSP. Fanjul-Peyro et al. (2019) and Rocha et al. (2008) studied the UPMSP with machine and job sequence-dependent setup times. Wang et al. (2020) addressed the UPMSP with random rework. Considering the development of globalization and distributed manufacturing, Lei et al.

* Corresponding author
E-mail: fdchou@tpts7.seed.net.tw (F.-D. Chou)
ISSN 1923-2934 (Online) - ISSN 1923-2926 (Print)
2023 Growing Science Ltd.
doi: 10.5267/j.ijiec.2023.4.001

(2020) and Lei et al. (2021) addressed the distributed UPMSP. Having a single objective tends to minimize the makespan (Lin, Pfund, & Fowler, 2011; Fanjul-Peyro, Ruiz, & Perea, 2019; Lei et al., 2020), the total weighted flow time (Pfund, Fowler, & Gupta, 2004), the total weighted completion time (TWCT) (Lin, Pfund, & Fowler, 2011; Rodriguez et al., 2013; Wang & Alidaee, 2019), the total weighted tardiness (Lin, Pfund, & Fowler, 2011), the makespan added to the weighted tardiness (TWT) (Rocha et al., 2008) and the expected TWT (Wang et al., 2020). The bi-objective (makespan and total tardiness) optimization problem is minimized simultaneously (Lei, Yuan, & Cai, 2021), and the multi-objective problem is to minimize the makespan simultaneously, TWCT and TWT (Lin & Ying, 2015; Lin et al., 2016). Furthermore, exact algorithms based on branch and bound (B&B) (Pfund, Fowler, & Gupta, 2004; Rocha et al., 2008) and mathematical programming (Fanjul-Peyro, Ruiz, & Perea, 2019; Rocha et al., 2008; Wang et al., 2020), heuristics (Lin, Pfund, & Fowler, 2011) and meta-heuristics, such as GAs (Lin, Pfund, & Fowler, 2011) and modified GAs (Wang et al., 2020), iterated greedy metaheuristics (Rodriguez et al., 2013), tabu search (TS) (Wang & Alidaee, 2019), simulated annealing (SA) (Wang et al., 2020), the imperialist competitive algorithm (ICA) with memory (Lei et al., 2020) and an improved artificial bee colony (ABC) algorithm (Lei, Yuan, & Cai, 2021), have been proposed.

It is noted that all the above studies assume that machines are continuously available over the scheduling horizon. However, in a real manufacturing environment, machines may not be available because of maintenance operations (Zhang et al., 2020), tool replacement (Dang et al., 2021), machine breakdowns (Kim & Kim, 2020), etc. UPMSPs with maintenance have been extensively studied in recent years for various maintenance activities and performance measures. For the UPMSP with at most one maintenance activity on each machine, Cheng et al. (2011) proved that the two problems of minimizing the total completion time and the total machine load can be optimally solved in polynomial time, Hsu et al. (2013) proposed models for the same problem as in reference (Cheng et al.,2011) considering three basic types of ageing effects and proved that the models could be solved optimally in polynomial time, and Lu et al. (2018) proposed an ABC-TS algorithm to find an approximately optimal solution in a reasonable time. For the UPMSP considering multiple maintenance activities and ageing effects simultaneously, Yang et al. (2012) applied the group balance principle to determine the optimal positions of the maintenance activities and the number of jobs in each group in the scheduling sequence on each machine. Tavana et al. (2015) presented an integrated three-stage model with the fuzzy analytic hierarchy process, the technique for order of preference by similarity to ideal solution, and goal programming. Gara-Ali et al. (2016) proposed a general model for the UPMSP with different maintenance systems and several performance criteria. For the UPMSP with multiple maintenance activities and sequence-dependent setup times, Avalos-Rosales et al. (2018) proposed a mathematical formulation with valid inequalities to obtain optimal solutions for small to medium instances and an efficient meta-heuristic algorithm based on a multistart strategy for solving larger instances. Lei and Yang (2022) proposed a multisubcolony ABC algorithm to simultaneously minimize the makespan and total tardiness. Lei and Yi (2021) presented a differentiated shuffled frog leaping algorithm and used its strong exploration ability to minimize the makespan. Wang & Pan (2019) presented a novel ICA with an estimation of distribution algorithm to simultaneously minimize the makespan and total tardiness. In addition, the distributed UPMSP with preventive maintenance (Lei & Liu, 2020), UPMSP with additional resources and maintenance (Lei & He, 2022) and UPMSP with release times and maintenance activities (Pang, Tsai, & Chou, 2021) have been researched.

Regarding the above works on the UPMSP with maintenance, three types of maintenance activities are generally assumed: periodically fixed, flexible and variable. For scheduling with fixed maintenance activities, the starting time and duration of each maintenance activity are predefined or given beforehand (Avalos-Rosales et al., 2018; Lei & Yang, 2022; Lei & Yi, 2021; Wang & Pan, 2019; Lei & He, 2022). For scheduling with flexible maintenance activities, the maintenance operation must be performed within a preplanned time window (Lei & Liu, 2020; Beldar et al., 2022) or below a specific threshold (Pang, Tsai, & Chou, 2021), and the duration is fixed (Pang, Tsai, & Chou, 2021) or related to the starting time of the maintenance (Beldar et al., 2022). For scheduling with variable or deteriorating maintenance activities, the maintenance starting times are treated as decision variables, and the maintenance duration is assumed to increase with the starting time (Cheng, Hsu, & Yang, 2011; Hsu et al., 2013; Lu et al., 2018) or is fixed (Yang et al., 2012; Tavana et al. 2015). However, in a real production environment, the starting time of a maintenance activity sometimes depends only on the reliability of the machine. To the best of our knowledge, scheduling with this kind of maintenance is very rare in the literature. Therefore, we considered the UPMSP with reliability-based maintenance in this paper, which assumes that the machine's status follows a discrete degradation process, and if the machine's reliability is less than the minimum acceptable level or threshold before starting job processing, the maintenance operation must be performed.

Additionally, a large number of meta-heuristics based on swarm intelligence, such as the GA (Lin, Pfund, & Fowler, 2011) , SA (Wang et al., 2020; Lin & Ying, 2015), ABC (Lei, Yuan, & Cai, 2021; Lei & Yang, 2022; Lei & Liu, 2020; Lei & He, 2022), TS (Wang & Alidaee, 2019), ICA (Lei et al., 2020), and hybrid algorithm ABC-TS (Lu et al., 2018), have been developed to deal with UPMSPs with maintenance. However, the application of meta-heuristics such as spider monkey optimization (SMO) has not been fully investigated. The SMO algorithm proposed by Bansal et al. (2014) is a swarm intelligence optimization algorithm inspired by the intelligent foraging behaviour of fission-fusion social structure-based animals. Since SMO has ability to trade-off between exploration and exploitation, SMO and its variants have been widely used to solve complex real-world optimization problems; these variants include numerical optimization and continuous constrained optimization (Sharma et al., 2016; Gupta et al., 2017). Cheruku et al. (2017) proposed a SMO-based rule miner (SM-RuleMiner) by incorporating a unique fitness function based on diabetes classification. In 2017, Sharma et al. suggested a method for determining the ideal placement and size of capacitors using a combination of a SMO approach based on a limaçon curve and a local search strategy inspired by the same curve. In recent years, SMO has been successfully applied to solve discrete optimization problems. Mumtaz et al. (2020) proposed a hybrid SMO algorithm for multilevel planning and scheduling problems of assembly lines. Yue et al., (2023) proposed a hybrid Pareto spider monkey optimisation algorithm for a two-stage flexible printed circuit board flow shop to minimize TWC and energy consumption simultaneously. Xia et al., (2021) introduced discrete SMO as a solution method for a vehicle routing problem involving uncertain demands. Their findings demonstrate that SMO exhibits strong global search capabilities.

This paper presents the HDSMO, a hybrid DSMO algorithm that employs a combination of discrete spider monkey optimization and GA techniques to effectively tackle the UPMSP problem with variable maintenance. To generate feasible initial solution, a three-phase constructive heuristic is proposed. To balance the exploration and exploitation capabilities of DSMO, an individual updating method considering the inertia weight is used. To enhance the search efficiency, a problem-oriented neighbourhood search method is applied. Experiments are conducted to compare the performance based on computational time and solution quality of HDSMO, DSMO, and GA on three different scales of the problem.

The research contributions of this paper can be summarized as follows. A DSMO algorithm composed of the spider monkey algorithm and GA is proposed. A hybrid DSMO algorithm is proposed with an initial solution generation method, discrete individual updating method and neighbourhood search method. In addition, the proposed methods have been evaluated and compared with the SMO and GA algorithms in a set of instances.

The subsequent sections of this paper are structured as follows. In Section 2, the UPMSP with maintenance considered in this study is presented, along with its mathematical model. The DSMO, and a proposed HDSMO that includes initial population generation, individual update, and neighbourhood search are presented in Section 3. The experimental results are presented in Section 4, where the proposed algorithm's validity is analyzed. In Section 5, conclusion and future research directions are provided.

2. Preliminaries

2.1. Problem description

A set $J = \{J_1, \cdots, J_j \cdots, J_n\}$ of n jobs is to be processed on m unrelated parallel machines M_i, $i = 1, \cdots, \mathrm{m}$. Let n_i denote the number of jobs assigned to M_i, and let $\sum_{i=1}^{m} n_i = \mathrm{n}$. We assume, as in most practical situations, that $m < \mathrm{n}$. The jobs are all available for processing at time zero. In this paper, we assume that a machine's reliability follows an exponential distribution, and we let L represent the cumulative processing time of the job being continuously processed or the age of the machine. Then, the reliability R of the machine is equal to $e^{-\lambda L}$, and λ is the machine failure rate. If the reliability falls below the threshold r_{th} before starting a job's processing, a maintenance operation must be performed to restore the machine to its original condition. Because both the frequency and location of the maintenance activities are decision variables, we define this maintenance activity as the variable maintenance like reference (Beldar et al., 2022). Using the three-field notation $\alpha|\beta|\gamma$ introduced by Graham et al. (1979), we denote our problem by $\mathrm{Rm/nr, VM/C_{max}}$, where nr denotes that the jobs are nonresumable, VM denotes variable maintenance, and the objective is to minimize the maximum completion time. The decision is to determine the allocation and sequence of n jobs on m machines and the maintenance activity arrangements. Since problem $\mathrm{Rm/nr, VM}/C_{max}$ is NP-hard (Lu et al., 2018), approximate methods are needed to solve real-size instances.

To obtain the near-optimal solution of the studied problem, two principles need to be followed. One is to allocate job J_j to the machine with the smallest processing time as often as possible, where the total processing times of the jobs allocated to the machines should be as close as possible. The second is to sequence the job on machine M_i to minimize the number of maintenance operations. Assume that all jobs between two maintenance operations are grouped into a batch; if the machine's reliability is equal to the threshold r_{th} before starting the last job in the batch, we define the batch as fully loaded. The decision regarding job sequencing in machine M_i is equivalent to the job-grouping batch decision.

Property: The batch is almost fully loaded, and the longer the processing time of the last job, the less maintenance is needed, meaning that a better solution can be obtained.

Proof: Suppose batch B_i^b on machine M_i, consisting of n_i^b jobs, can be divided into two sub-batches, job $J_{n_i^b}$ and the remaining $\mathrm{n}_i^b - 1$ jobs. Assume that the processing time of job $J_{\mathrm{n}_i^b}$ is $p_{in_i^b} = \max\ p_{ij}, J_j \in B_i^b$ and the total processing time of the $\mathrm{n}_i^b - 1$ jobs is $t^p_{\mathrm{n}_i^b - 1} = \sum_{j \neq \mathrm{n}_i^b} p_{ij}$, where $e^{-\lambda p_{in_i^b}} > r_{th}$ and $e^{-\lambda t^p_{\mathrm{n}_i^b - 1}} > r_{th}$. If job $J_{\mathrm{n}_i^b}$ is processed after the other $\mathrm{n}_i^b -$ 1 jobs, the completion time of batch B_i^b is $CT_{iB_i^b} = \sum_{j=1}^{\mathrm{n}_i^b} p_{ij}$, as shown in Fig. 1(a). If we swap job $J_{\mathrm{n}_i^b}$ with any job before it, the completion time of batch B_i^b is likely to be $CT'_{iB_i^b} = \sum_{j=1}^{\mathrm{n}_i^b} p_{ij} + x * t^{PM}$, and the number of maintenance operations is

$x \geq 0$, as shown in Fig.1(b). Because the continuous processing time of the first $n_i^b - 1$ jobs becomes longer after the swap, the machine reliability before starting job number n_i^b in the batch is probably lower than the threshold r_{th}, and maintenance is needed. Thus, $CT'_{iB_i^b} - CT_{iB_i^b} \geq 0$.

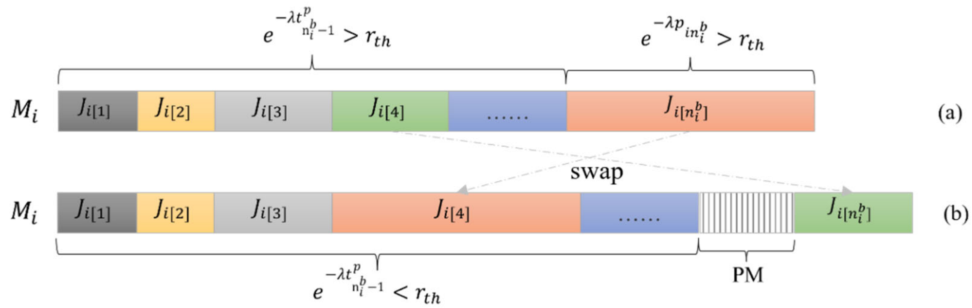


Fig. 1. An example of different job sequences for job-grouping batch

2.2. Mixed integer programming model

Indices and variables

i: $i = 1,2,3, \dots, m$, the machine index
j: $j = 1,2,3, \dots, n$, the job index
k: $k = 1,2,3, \dots, n$, the job position index in a machine
$ST_{i[k]}$: the start time at the k^{th} position of machine M_i
$CT_{i[k]}$: the completion time at the k^{th} position of machine M_i
t^{PM}: the time required for maintenance
λ: the failure rate in the exponential distribution of machine reliability
r_{th}: the threshold of machine reliability
$L^0_{i[k]}$: the age of machine M_i in position k before processing a job
$L^1_{i[k]}$: the age of machine M_i in position k after processing a job
$R^0_{i[k]}$: the reliability of machine M_i in position k before processing a job
p_{ij}: the processing time of job J_j in machine M_i
C_{max}: the largest completion time of the m machines

Decision variables

$$X_{ijk} = \begin{cases} 1 & \text{if job } \mathrm{J_j} \text{ is performed at } \mathrm{k^{th}} \text{ position of machine } \mathrm{M_i} \\ 0 & \text{Otherwise} \end{cases}$$

$$Y_{ik} = \begin{cases} 1 & \text{if maintenance is applied after } \mathrm{k^{th}} \text{ position of machine } \mathrm{M_i} \\ 0 & \text{Otherwise} \end{cases}$$

Mathematical model

$$min. \quad C_{max} \tag{1}$$

$$\sum_{j=1}^{n} X_{ijk} \leq 1, \forall \begin{cases} i = 1,2, \dots, m \\ k = 1,2, \dots, n \end{cases} \tag{2}$$

$$\sum_{i=1}^{m} \sum_{k=1}^{n} X_{ijk} = 1, \forall\, j = 1,2, \dots, n \tag{3}$$

$$ST_{i[1]} = 0, \qquad \forall\, i = 1,2, \dots, m \tag{4}$$

$$L^0_{i[1]} = 0, \forall\, i = 1,2, \dots, m \tag{5}$$

$$CT_{i[1]} = \sum_{j=1}^{n} p_{ij} * X_{ij1}, \forall\, i = 1,2, \dots, m \tag{6}$$

$$ST_{i[k]} \geq CT_{i[k-1]} + t^{PM} * Y_{ik}, \forall \begin{cases} i = 1,2, \dots, m \\ k = 2,3, \dots, n \end{cases} \tag{7}$$

$$CT_{i[k]} \geq ST_{i[k]} + \sum_{j=1}^{n} p_{ij} * X_{ijk}, \forall \begin{cases} i = 1,2, \dots, m \\ k = 2,3, \dots, n \end{cases} \tag{8}$$

$$L^1_{i[k]} \geq L^0_{i[k]} + \sum_{j=1}^{n} p_{ij} * X_{ijk}, \forall \begin{cases} i = 1,2, \dots, m \\ k = 2,3, \dots, n \end{cases} \tag{9}$$

$$L^0_{i[k]} = L^1_{i[k-1]}(1 - Y_{ik}), \forall \begin{cases} i = 1,2,\dots,m \\ k = 2,3,\dots,n \end{cases} \tag{10}$$

$$R^0_{i[k]} = e^{-\lambda L^0_{i[k]}}, \forall \begin{cases} i = 1,2,\dots,m \\ k = 2,3,\dots,n \end{cases} \tag{11}$$

$$R^0_{i[k]} \geq r_{th}, \forall \begin{cases} i = 1,2,\dots,m \\ k = 2,3,\dots,n \end{cases} \tag{12}$$

$$C_{max} \geq CT_{i[n]}, \forall\, i = 1,2,\dots,m \tag{13}$$

In this model, the objective is to minimize C_{max}, as shown in Eq. (1). Constraints (2) and (3) ensure that each job can only be processed at one position of one machine and that each position of one machine can only be occupied by one job. Constraint (4) specifies the start time of each machine at the first position. Constraint (5) specifies the age of the machine at the first position before processing the job. Constraint (6) specifies the completion time of each machine at the first position. The start and completion times for each machine at each position are defined by constraints (7) and (8). Constraints (9) and (10) define the age of the machine. Constraints (11) and (12) define the reliability function and threshold of the machine. Constraint (13) defines the maximum completion time of each machine.

3. HDSMO algorithm for the UPMSP with variable maintenance

3.1. Encoding and decoding of individuals

According to the definition of the proposed problem, the encoding only needs to specify which machine each job will be processed on; thus, the chromosome can be depicted as a vector that has a length of $\mathrm{n} + \mathrm{m} - 1$, where n denotes the total number of jobs and m indicates the total number of machines. For instance: $J_{1[1]}, \cdots, J_{1[n_1]}, 0, \cdots, 0, \cdots, 0, J_{m[1]}, \cdots, J_{m[n_m]}$, where the sequence of jobs processed on a single machine is represented by a string of numbers that are divided by 0, while the allocation of jobs to various machines is indicated by 0. An encoding example for a solution of a problem with 15 jobs and 4 machines is illustrated in Fig. 2.

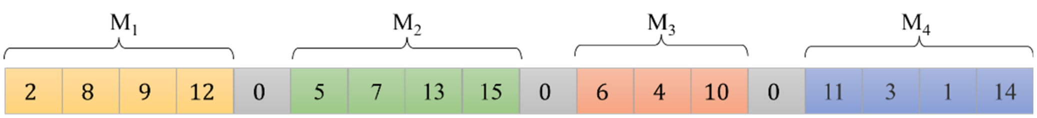


Fig. 2. An example of encoding

The decoding process calculates the objective value and fitness of individual SM_h. Assume the completion time CT_i is 0 and the set of completion times is $MC = \{\emptyset\}$. The decoding process is shown in **Algorithm 1**.

Algorithm 1: Decoding
Input: n, m, r_{th}, p_{ij}, λ, t^{PM}, $MC = \{\Phi\}$
Initialize the age and reliability of machine M_i, $L^0_{i[1]} = 0$, $R^0_{i[1]} = 1$
For spider monkey individuals $h = 1$ to N **do**
For machines $i = 1$ to m **do**
For machine job positions $k = 1$ to n_i, select job $J_{i[k]}$
If $R^0_{i[k]} \geq r_{th}$, set $L^1_{i[k]} = L^0_{i[k]} + p_{i[k]}$, $L^0_{i[k+1]} = L^1_{i[k]}$, $CT_{i[k]} = CT_{i[k]} + p_{i[k]}$
Else set $L^0_{i[k+1]} = p_{i[k]}$, $CT_{i[k]} = CT_{i[k]} + t^{PM} + p_{i[k]}$
End for
$CT_i = CT_{i[n_i]}$, $MC = MC \cup \{\mathrm{CT}_i\}$
End for
$C_h = \max\{CT_i, CT_i \in MC\}$, $f_h = \frac{1}{C_h}$
End for

3.2. Proposed DSMO algorithm

Empirical studies have demonstrated that the SMO algorithm exhibits strong performance when applied to continuous optimization (Bansal et al., 2014; Gupta et al., 2017). However, the problem $Rm/nr, VM/C_{max}$ is a combination optimization problem. The proposed DSMO algorithm incorporates distinct update techniques for discrete individuals during the local leader phase, global leader phase, and local leader decision phase. The following process outlines how these methods can facilitate the inheritance of genes from both the global optimal individual and the local optimal individual by an individual.

3.2.1 Local leader phase

In this stage, each individual SM_h updates its position with the information of LL_l, which is the local optima in group l, and an individual SM_r, which is different from SM_h in group l, as shown in Eq. (14). The individual is updated in two steps. First, the selected parent individual SM_h is attracted toward the local leader by Eq. (15). Second, the generated individual SM_h', after Step 1, is updated with the randomly selected individual SM_r to avoid premature stagnation, as shown in Eq. (16).

$$SM_{new_h} = p_1 \otimes f(p_r \otimes g(SM_h, LL_l), SM_r) \tag{14}$$

$$SM_h' = p_r \otimes g(SM_h, LL_l) = \begin{cases} g(SM_h, LL_l) & p_{x1} > p_r \\ SM_h & otherwise \end{cases} \tag{15}$$

$$SM_{new_h} = p_1 \otimes f(SM_h', SM_r) = \begin{cases} f(SM_h', SM_r) & p_{x2} < p_1 \\ SM_h' & otherwise \end{cases} \tag{16}$$

where p_{x1} and p_{x2} are uniformly distributed random numbers in the range [0,1]. p_1 and p_r are the crossover rates, and $p_1, p_r \in (0,1)$; $p_r = p_r + 0.4/Maxt$. We set the initial value to 0.1 in this paper, and $Maxt$ is the maximum number of iterations. $g(SM_h, LL_l)$ and $f(SM_h', SM_r)$ represent the crossover operations between individuals, and two kinds of crossover operators are designed. One retains the genes of the two parent individuals SM_h and LL_l, and the rest of the genes are randomly sequenced, as shown in Fig. 3 (a). The other operation is for two parent individuals without the same genes; a random interval in $[1, n + m - 1]$ is selected for crossover by mapping, as shown in Fig. 3 (b). The fitness value of the generated individual SM_{new_h} is calculated, and if the fitness of the new individual is better than that of the old one, then SM_h is replaced with the new individual SM_{new_h}.

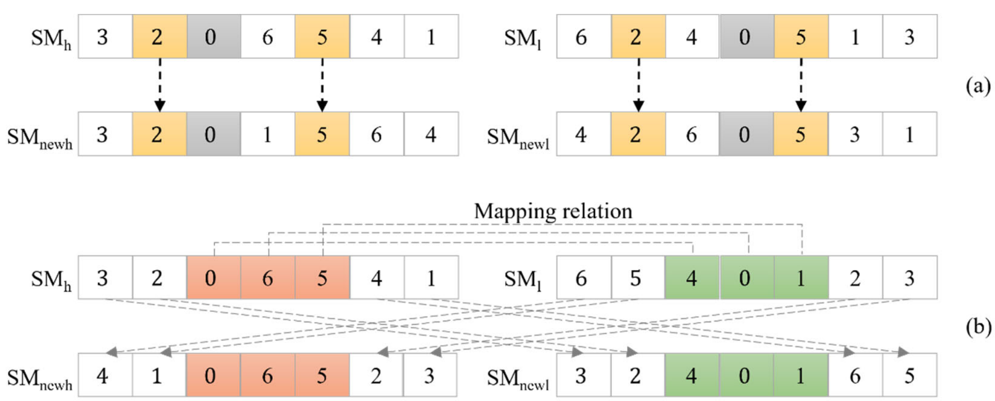


Fig. 3. Two kinds of crossover operations, (a) with the same genes and (b) without the same genes

3.2.2 Global leader phase

During the global leader phase, as demonstrated in Eq. (17), the status of the individual SM_h is revised, similar to the local leader phase. The differences are as follows: (a) the update probability p_h of the individual SM_h is dependent on its fitness, as shown in Eq. (18); (b) the individual SM_h updates its position according to the experience of the global leader, GL, and another random individual, SM_r, in the population, as shown in Eq. (19) and Eq. (20).

$$SM_{new_h} = p_2 \otimes f(p_h \otimes g(SM_h, GL), SM_r) \tag{17}$$

$$p_h = 0.9 \times \frac{f_h}{f_{max}} + 0.1 = 0.9 \times \frac{\frac{1}{C_h}}{f_{max}} + 0.1 \tag{18}$$

$$SM_h' = p_h \otimes g(SM_h, GL) = \begin{cases} g(SM_h, GL) & p_{y1} < p_h \\ SM_h & otherwise \end{cases} \tag{19}$$

$$SM_{new_h} = p_2 \otimes f(SM_h', SM_r) = \begin{cases} f(SM_h', SM_r) & p_{y2} < p_2 \\ SM_h' & otherwise \end{cases} \tag{20}$$

where p_{y1} and p_{y2} are uniformly distributed random numbers in the range [0,1]. p_2 is the crossover rate, and $p_2 \in (0,1)$. $g(SM_h, GL)$ and $f(SM_h', SM_r)$ represent the crossover operations between two individuals. The crossover operators are the same as in the local leader phase.

3.2.3 Global leader learning phase and local leader learning phase

The GL updates its position by using a greedy selection process, and SM_h, having the best fitness among all the spider monkeys, is selected as the new position of the GL. If the position of the GL remains unchanged, then let the global limit count be $n_{glc} = n_{glc} + 1$. Like the global leader learning phase, the positions of the LLs in all the groups are updated, selecting the SM_l with the best fitness in each group. If the position of the LL remains unchanged, then let the local limit count be $n_{llc} = n_{llc} + 1$.

3.2.4 Local leader decision phase and global leader decision phase

If the LL position of a group is not updated for a predetermined number of iterations, i.e., $n_{llc} > n_{lll}$, then the positions of the spider monkeys are updated by using information from both the LL and GL based on the probability p_r through Eq. (21).

$$SM_{new_h} = p_r \otimes f(p_r \otimes g(SM_h, GL), LL_l) \tag{21}$$

Similar to the local leader decision phase, if the global limit count is larger than the global leader limit, that is, $n_{glc} > n_{gll}$, then the population is split into subgroups until the number of groups reaches the maximum allowed number of groups MG, and then they are combined to form a single group again.

3.3 HDSMO algorithm

An HDSMO algorithm integrating the merits of DSMO with three improvements is presented in this section. First, a three-phase heuristic is proposed to generate a better initial solution. Second, the position update method considering the inertial weight is proposed to balance the exploration and exploitation capabilities of DSMO. Third, a problem-oriented neighbourhood search method with jump and swap operations is designed to improve the search efficiency of the HDSMO algorithm. The flowchart of HDSMO is shown in Fig. 4.

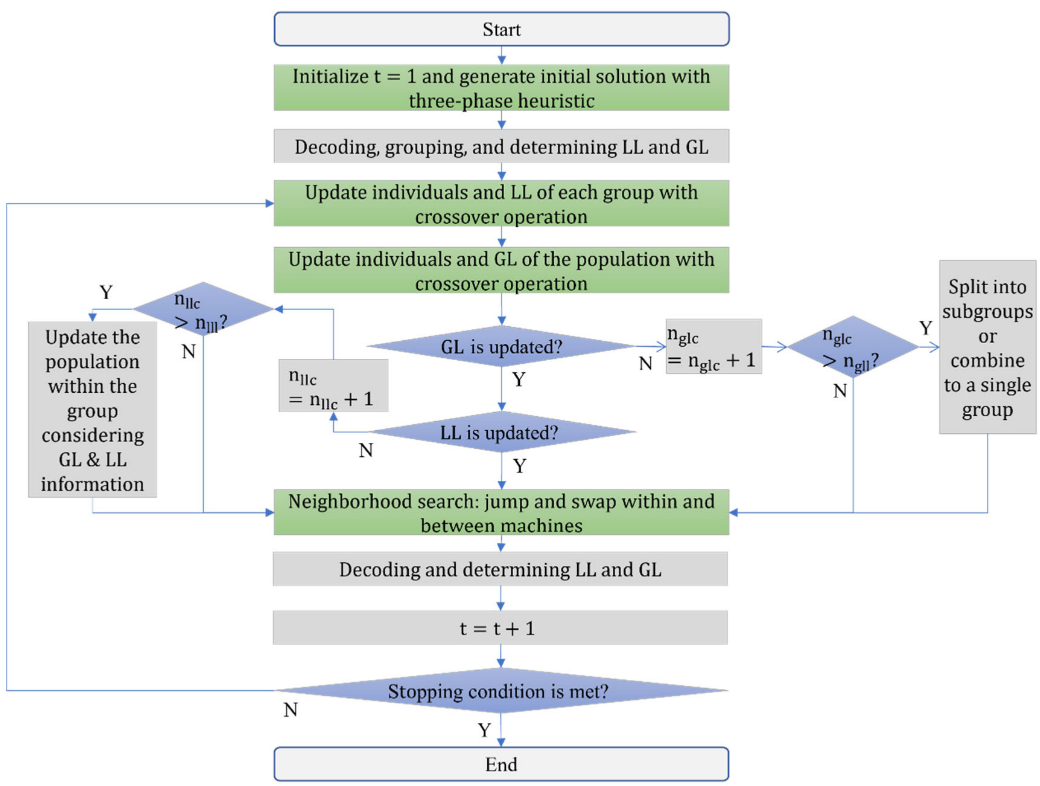


Fig. 4. Flow chart of the HDSMO algorithm

3.3.1 Initial population generation method

Based on the two-phase scheduling heuristic (Lin, Pfund, & Fowler, 2011) and the properties of the addressed problem, a three-phase heuristic is designed to generate the initial solutions.

Step 1: Apply the following linear programming relaxation model to generate a partial schedule. If $X_{ij} = 1$, job J_j is assigned to machine M_i; otherwise, job J_j is not assigned to any machine.

$$min \quad C_{max} \tag{22}$$

$$\sum_{j=1}^{n} p_{ij} X_{ij} \leq C_{max}, \forall i = 1,2, \dots, n \tag{23}$$

$$\sum_{i=1}^{m} X_{ij} = 1, \forall j = 1,2, \dots, n \tag{24}$$

$$0 \leq X_{ij} \leq 1, \forall \begin{cases} i = 1,2, \dots, m \\ j = 1,2, \dots, n \end{cases} \tag{25}$$

Step 2: For an unassigned job J_j, set $max_{i=1,\dots,m}\{X_{ij}\} = 1$ and $X_{kj} = 0$ for all $k \neq i$ to obtain a complete schedule.

Step 3: A group of n_i jobs are assigned to machine M_i in batches considering the machine's reliability threshold to obtain the solution of the proposed problem. The total processing time of batch B_i^b is estimated by $t_i^p = \frac{\ln r_{th}}{-\lambda} + \frac{\sum_{j=1}^{n_i} p_{ij}}{n_i}$. Then, the lower bound of the batch number n_{b_i} is estimated by Eq. (26).

$$n_{b_i} = \begin{cases} (\sum_{j=1}^{n_i} p_{ij}) div(t_i^p) - 1 & (\sum_{j=1}^{n_i} p_{ij}) mod(t_i^p) = 0 \\ \left(\sum_{j=1}^{n_i} p_{ij}\right) div(t_i^p) & else \end{cases} \tag{26}$$

Assuming USJ_i is the set of unscheduled jobs on machine M_i sequenced by the longest processing time rule, LJ_i is the first n_{b_i} jobs in the set USJ_i, and SJ_i is the remaining $n_i - n_{b_i}$ jobs. The process of job-grouping batch is shown in Fig.5.

Input: n, m, p_{ij}, λ, r_{th}, t^{PM}

Initialization: USJ_i, n_{b_i}, LJ_i, SJ_i, $CT_i = 0, L_{i[1]}^0 = 0, R_{i[1]}^0 = 1$

For machines $i = 1$ to m **do**

For machine job positions $k = 1$ to n_i **do**

Select job $J_{j'} = arg\{maxp_{ij'} | J_{j'} \epsilon SJ_i\}$, try to allocate job $J_{j'}$ to position k of machine M_i, and calculate the life $L_{i[k]}^1$ and reliability $R_{i[k]}^1$

If $R_{i[k]}^1 \geq r_{th}$, allocate job $J_{j'}$ to position k of machine M_i, and update $L_{i[k]}^0$, $CT_{i[k]}$, $SJ_i = SJ_i \backslash \{J_{j'}\}$,$USJ_i = USJ_i \backslash \{J_{j'}\}$

Else select job $J_{j^\#} = arg\{minp_{ij^\#} | J_{j^\#} \epsilon SJ_i\}$, try to allocate job $J_{j^\#}$ to position k of machine M_i, and calculate the life $L_{i[k]}^1$ and reliability $R_{i[k]}^1$

If $R_{i[k]}^1 \geq r_{th}$, allocate job $J_{j^\#}$ to machine M_i, and update $L_{i[k]}^0$, $CT_{i[k]}$, $SJ_i = SJ_i \backslash \{J_{j^\#}\}$, $USJ_i = USJ_i \backslash \{J_{j^\#}\}$

Else select job $J_{j\%} = arg\{maxp_{ij\%} | J_{j\%} \epsilon LJ_i\}$, allocate job $J_{j\%}$ to position k of machine M_i, and update $L_{i[k]}^0$, $CT_{i[k]}$, $LJ_i = LJ_i \backslash \{J_{j\%}\}$, $USJ_i = USJ_i \backslash \{J_{j\%}\}$

End if

End if

End for

End for

Output the solution and the objective value $C_{max} = maxCT_i$

Fig.5. The process of job-grouping batch

3.3.2 Local leader and global leader update with the inertia weight

The balance between global and local search throughout the course of a run is critical to the success of an evolutionary algorithm (Nickabadi et al., 2011). For the basic DSMO algorithm, the old individual position is totally inherited, which is not good for the solution search. The inertia weight is proposed to balance the exploration and exploitation characteristics of DSMO. The method of updating individuals is improved by considering the inertia weight p_w given in Eqs. (27) and (28), and the individual position update with inertia weight is shown in Eq. (29), which represents the inheritance judgement of the individual SM_h to the previous position.

$$SM_{new_h} = p_1 \otimes f(p_r \otimes g(p_w \otimes v(SM_h), LL_l), SM_r) \tag{27}$$

$$SM_{new_h} = p_2 \otimes f(prob_h \otimes g(p_w \otimes v(SM_h), GL), SM_r) \tag{28}$$

$$SM_h{}' = p_w \otimes v(SM_h) = \begin{cases} v(SM_h) & p_z < p_w \\ SM_h & otherwise \end{cases} \tag{29}$$

where p_z is a random number in the interval [0,1] and when $p_z < p_w$, operation $v(SM_h)$ is performed by mutation. For the individuals in the first and last 50% of the generation population, the mutation operations of order reversal and two-point exchange were adopted, respectively, as shown in Fig.6.

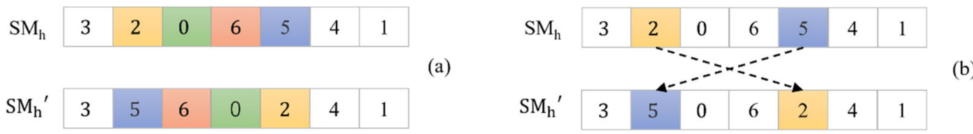


Fig.6. Two kinds of mutation operations: (a) order reversal and (b) swap

3.3.3 Neighbourhood search

For permutation problems, insert or jump and pairwise swap moves are widely used (Chen et al., 2021; Zhang & Chen, 2022) to improve the search efficiency. Two types of neighbourhood operators are defined in this paper to efficiently explore the solution space: (1) move one job from one batch to another batch of a machine or from one machine to another machine if it decrease the completion time, which is called a jump; (2) exchange two jobs from different batches of a machine or different machines if it decrease the completion time, which is called a swap.

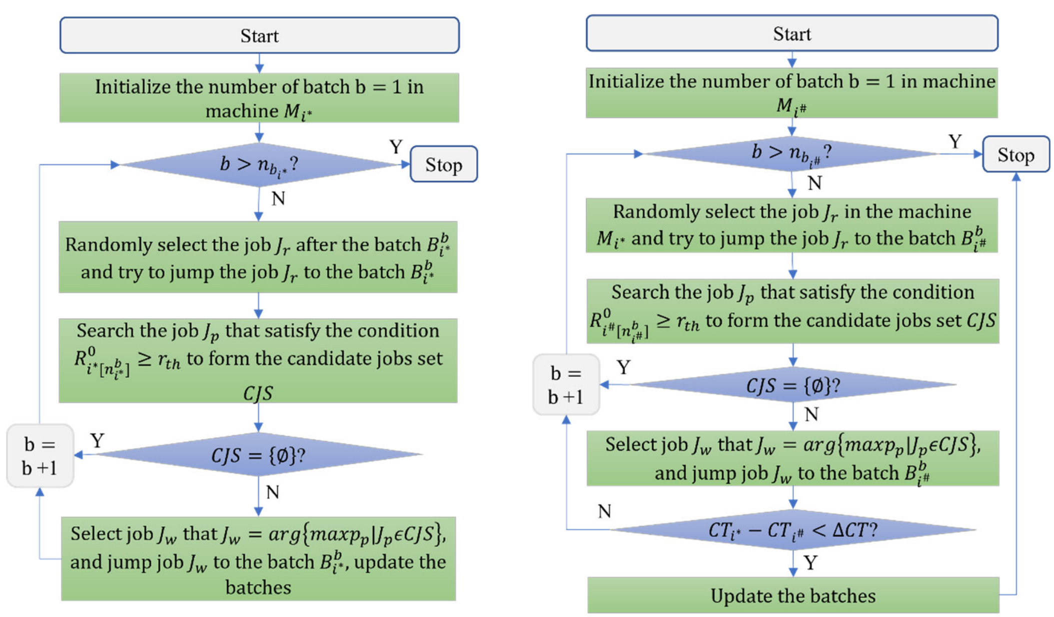
For a spider monkey individual SM_h, intra-machine or inter-machine neighbourhood search is applied according to the deviation rate of the makespan. It is calculated from the maximum and minimum completion times of the parallel machine $D_h^R = \frac{CT_h^{max} - CT_h^{min}}{CT_h^{max}}$. When $D_h^R \leq 0.1$, the intra-machine neighbourhood search operation is applied; otherwise, the inter-machine neighbourhood search operation is performed. If the objective function value of the neighbouring solution is better than that of the current solution, the current solution is updated. Otherwise, the current solution is retained.

Define n_{b_i} as the number of batches on machine M_i and n_i^b as the number of jobs in batch B_i^b. M_{i^*} and $M_{i^\#}$ represent the machines with the longest completion time and the shortest completion time, respectively. The jump and swap operations are performed sequentially for an individual SM_h, and a limit ΔCT is set for stopping the neighbourhood search between machines. The process of the jump and swap operations is shown in Fig. 7.

4. Parameter tuning and computational experiments

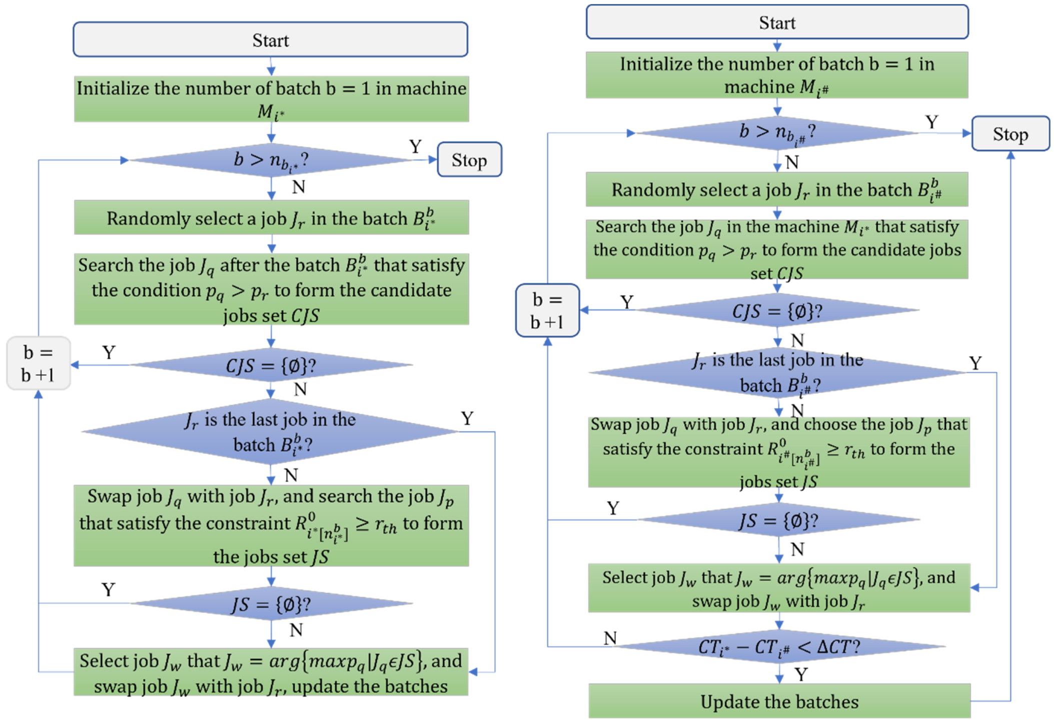
4.1. Data generation

In this section, we present the computational experiments on the performance of the proposed algorithm. To evaluate the effectiveness of the proposed HDSMO algorithm, HDSMO is compared with the DSMO algorithm and GA. The algorithms were programmed by using MATLAB R2020a software, and the MIP model was implemented on the LINGO 18.0 x64 platform. All programs were run on an Intel(R) Xeon(R) Gold 6242R @3.10 GHz CPU, 64.0 GB RAM workstation.



(a) Jump in a machine

(c) Jump between machines



(b) Swap in a machine

(d) Swap between machines

Fig.7. The main processes of the jump and swap operations

The machine parameters, job parameters, and maintenance activity parameters are utilized in the execution of the experiments. The parameters of the machine consist of the number of machines m, whereas the parameters of the job comprise the quantity of jobs n, and the time for processing p_{ij}; and the maintenance activity parameters include the threshold $r_{th} = 0.4$, maintenance time t^{PM}, and machine failure rate, λ. The instances and experimental parameters are shown in Table 1, as were also shown in reference (Avalos-Rosales et al., 2018). We used 4 machines with 20 jobs (n20m4) to represent the problem instances. For each combination of problem instance sizes, 10 instances were randomly generated.

Table 1
Instances and experimental parameters

Size	m	n	p_{ij}	λ	t^{PM}
Small	2,3	6,8,10,12	$U[1,100]$	0.007	10
Medium	3,4,5	15,20,25,30,35,40,50,60	$U[1,100]$	0.0035	20
Large	10,15,20	100,150,200,250	$U[1,100]$	0.0035	20

4.2. Parametric tuning

Proper adjustment of the parameters is crucial for optimizing the algorithm's performance, hence employing suitable techniques to tune these parameters is essential. A set of 15 typical examples are selected from the problems in Table 1, and the Taguchi method is employed to establish the algorithm's parameters for problems of varying scales (Yue et al., 2019). Taking the HDSMO algorithm as an example, the parameters of the algorithm include the population size N, the maximum number of iterations $Maxt$, crossover rates p_1 and p_2 and the inertia weight p_w. The parameters and levels of the HDSMO algorithm determined by the pre-experiment are shown in Table 2.

Table 2
The parameters and levels for the HDSMO algorithm

Parameters	Small	Medium	Large
N	30,50,80,100	100,200,300,350	200,300,450,500
$Maxt$	50,100,150,200	100,200,300,400	200,300,400,500
p_1	0.2,0.3,0.4,0.5	0.3,0.4,0.5,0.6	0.5,0.6,0.7,0.8
p_2	0.2,0.3,0.4,0.5	0.3,0.4,0.5,0.6	0.5,0.6,0.7,0.8
p_w	0.05,0.1,0.15,0.2	0.1,0.2,0.25,0.3	0.2,0.25,0.3,0.35

Taking the makespan as the response variable, each parameter combination experiment is run 10 times. The number of orthogonal experiments under 5 parameters and 4 levels is $L_{16}(4^5)$, and the total number of experiments required is 16*15*10=2400. The orthogonal experimental design was generated in Minitab, and the signal-to-noise (S/N) ratio analysis result of the HDSMO algorithm under the three instance sets is shown in Fig. 8.

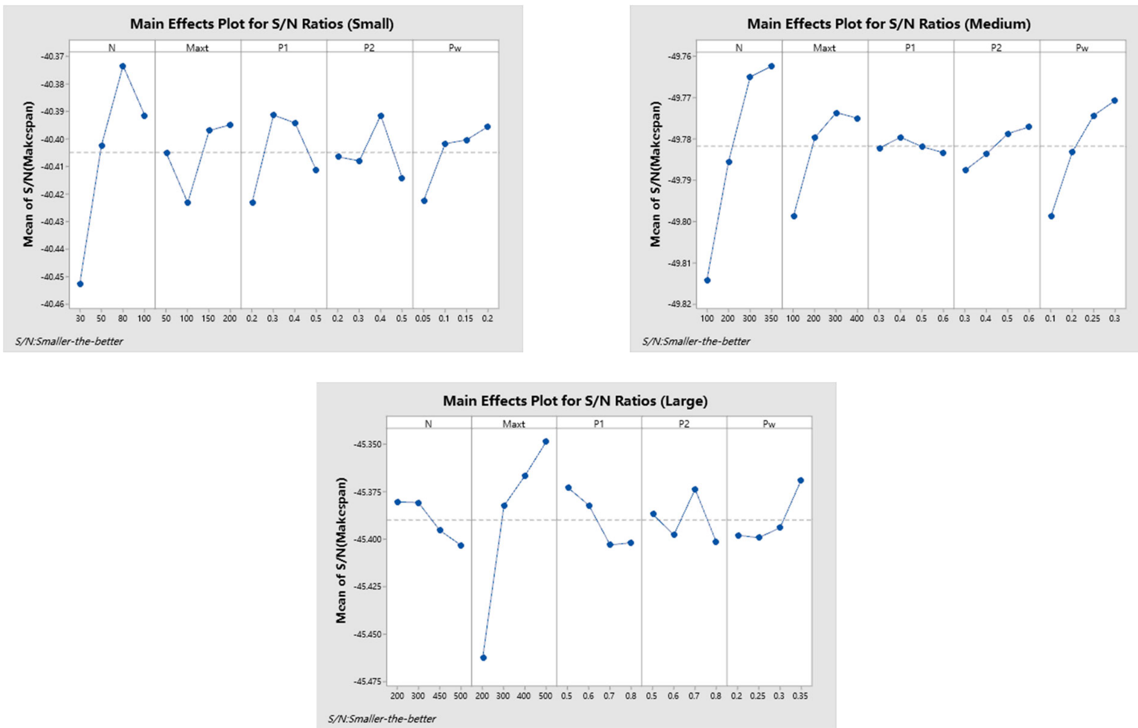


Fig. 8. The mean S/N results for the makespan of the HDSMO algorithm

For each parameter, the level with the largest S/N ratio is selected. According to the same method, the Table 3 illustrates the determined parameter values for various algorithms, obtained through experimental analysis of the parameters in the GA and DSMO algorithms.

Table 3
Parameter values for different algorithms

Parameter	GA			DSMO			HDSMO		
	Small	Medium	Large	Small	Medium	Large	Small	Medium	Large
N	100	350	500	100	300	450	80	350	200
$Maxt$	200	400	500	100	400	500	200	300	500
p_1/p_c	0.4	0.3	0.5	0.3	0.6	0.8	0.3	0.4	0.5
p_2/p_m	0.15	0.2	0.2	0.5	0.6	0.8	0.4	0.6	0.7
p_w	/	/	/	/	/	/	0.2	0.3	0.35

4.3. Computational experiments and discussion

The computational results are shown in Tables 4-6 for all the test instances. The objective value of C_{max}, and the relative percentage deviation (PD) is applied to compare the three evaluated methods, together with the average computation time (CT) in seconds. The PD values were calculated by $PD = \frac{C_{max}-C_{max}^*}{C_{max}^*}$. C_{max}^* is the optimal solution obtained from the MIP model of the small-scale problems and the best solution obtained by different algorithms for the medium- and large-scale problems. Due to the low efficiency of the MIP models, the medium- and large-scale problems only include the computation times of the heuristics and intelligent algorithms.

Table 4
The performance of the algorithms (Small)

n*m*	MIP			GA			DSMO			HDSMO		
	CT	C_{max}	PD	CT	C_{max}	PD	CT	C_{max}	PD	CT	C_{max}	PD
n6m2	35.97	**103.7**	**0.000**	1.74	103.7	**0.000**	**0.90**	103.7	**0.000**	3.23	103.7	**0.000**
n6m3	42.44	**69.0**	**0.000**	1.62	69.0	**0.000**	**1.03**	70.5	0.022	4.23	69.0	**0.000**
n8m2	8385.64	**140.7**	**0.000**	1.42	140.7	**0.000**	**0.98**	141.7	0.007	3.45	140.7	**0.000**
n8m3	9029.06	**95.5**	**0.000**	2.24	96.4	0.009	**1.14**	98.5	0.031	4.39	95.7	0.002
n10m2	12618.35	**192.6**	**0.000**	1.57	193.6	0.005	**1.19**	198.4	0.030	3.96	193.6	0.005
n10m3	16229.79	**95.1**	**0.000**	1.73	98.0	0.030	**1.32**	99.0	0.041	4.65	96.2	0.012
n12m2	18000	**232.0**	**0.000**	1.77	234.6	0.011	**1.37**	232.8	0.003	4.77	234.4	0.010
n12m3	12663.53	**110.0**	**0.000**	1.92	122.1	0.110	**1.50**	121.8	0.107	4.93	110.8	0.007
Average	9625.60	**129.8**	**0.000**	1.75	132.3	0.021	**1.18**	133.3	0.030	4.20	130.5	0.005

Table 5
The performance of the algorithms (Medium)

n*m*	GA			DSMO			HDSMO		
	CT	C_{max}	PD	CT	C_{max}	PD	CT	C_{max}	PD
n15m3	**12.013**	142.6	0.038	23.903	143.0	0.041	35.889	**137.4**	**0.000**
n15m4	**13.310**	104.2	0.113	24.662	103.6	0.107	38.062	**93.6**	**0.000**
n15m5	**14.655**	81.5	0.101	26.396	80.6	0.089	43.520	**74.0**	**0.000**
n20m3	**13.966**	200.6	0.109	30.210	186.2	0.029	40.427	**180.9**	**0.000**
n20m4	**14.109**	152.4	0.290	31.145	134.8	0.141	42.148	**118.1**	**0.000**
n20m5	**14.989**	122.3	0.353	32.093	112.8	0.248	44.811	**90.4**	**0.000**
n25m3	**15.691**	270.5	0.199	36.585	253.8	0.125	44.602	**225.6**	**0.000**
n25m4	**16.355**	190.0	0.412	37.994	168.9	0.255	45.163	**134.6**	**0.000**
n25m5	**17.809**	167.0	0.678	39.057	154.7	0.555	47.673	**99.5**	**0.000**
n30m3	**18.437**	352.4	0.309	42.712	338.9	0.259	51.462	**269.2**	**0.000**
n30m4	**19.499**	242.8	0.456	43.592	238.9	0.432	50.455	**166.8**	**0.000**
n30m5	**18.533**	206.1	0.748	45.173	200.6	0.701	52.502	**117.9**	**0.000**
n35m3	**19.060**	392.9	0.367	49.476	380.5	0.324	58.653	**287.4**	**0.000**
n35m4	**20.973**	299.0	0.633	49.605	274.7	0.500	55.618	**183.1**	**0.000**
n35m5	**20.987**	241.5	0.876	50.236	228.4	0.775	56.479	**128.7**	**0.000**
n40m3	**21.207**	479.1	0.336	54.224	452.2	0.261	73.937	**358.5**	**0.000**
n40m4	**22.250**	388.2	0.705	55.300	389.9	0.712	60.991	**227.7**	**0.000**
n40m5	**22.251**	304.6	0.977	56.555	299.3	0.942	60.889	**154.1**	**0.000**
n50m3	**25.193**	621.3	0.409	66.772	616.1	0.398	90.185	**440.8**	**0.000**
n50m4	**25.016**	482.4	0.867	67.643	487.8	0.888	72.982	**258.4**	**0.000**
n50m5	**24.789**	421.4	1.127	68.049	429.7	1.169	71.131	**198.1**	**0.000**
n60m3	**27.651**	823.6	0.484	78.053	763.8	0.376	106.277	**555.0**	**0.000**
n60m4	**31.557**	608.4	0.824	79.492	620.2	0.860	92.450	**333.5**	**0.000**
n60m5	**30.809**	505.4	1.403	79.655	502.9	1.391	80.324	**210.3**	**0.000**
Average	**20.046**	325.0	0.534	48.691	315.1	0.482	59.026	**210.2**	**0.000**

Table 6
The performance of the algorithms (Large)

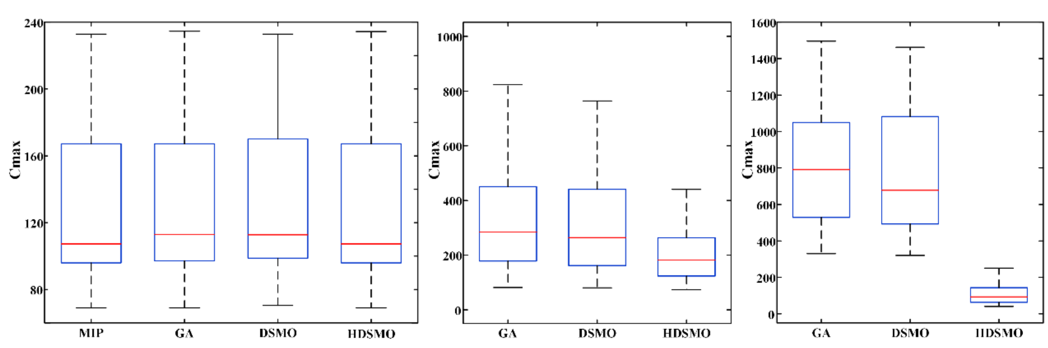
n*m*	GA			DSMO			HDSMO		
	CT	C_{max}	PD	CT	C_{max}	PD	CT	C_{max}	PD
n100m10	**121.056**	524.2	3.831	289.134	533.4	3.916	138.101	**108.5**	**0.000**
n100m15	**140.514**	416.1	5.700	304.562	386.7	5.227	158.355	**62.1**	**0.000**
n100m20	179.374	330.0	7.108	329.983	320.3	6.870	**169.284**	**40.7**	**0.000**
n150m10	**173.622**	850.0	4.296	420.129	806.6	4.026	182.417	**160.5**	**0.000**
n150m15	**190.879**	663.7	7.025	442.539	624.1	6.547	205.575	**82.7**	**0.000**
n150m20	**216.270**	533.6	9.011	461.788	492.9	8.248	224.662	**53.3**	**0.000**
n200m10	**215.577**	1172.6	4.678	546.221	1156.8	4.602	247.991	**206.5**	**0.000**
n200m15	**230.929**	891.1	7.618	569.715	865.2	7.368	241.927	**103.4**	**0.000**
n200m20	**255.377**	732.1	10.386	587.604	678.1	9.546	265.533	**64.3**	**0.000**
n250m10	**257.126**	1495.7	4.964	677.538	1462.2	4.830	292.197	**250.8**	**0.000**
n250m15	**221.977**	1142.5	8.075	693.807	1082.0	7.594	293.251	**125.9**	**0.000**
n250m20	**233.510**	956.0	11.529	717.169	849.1	10.128	308.877	**76.3**	**0.000**
Average	**203.018**	809.0	7.019	503.349	771.5	6.575	227.348	**111.2**	**0.000**

It can be concluded from Table 4 that the MIP model can obtain the optimal solutions of all the small-scale test problems, while CT greatly increases with the increase in the number of machines and jobs. HDSMO and GA can obtain the optimal solutions for 3 of 8 instances, and the average PD of HDSMO is smaller than GA. HDSMO outperforms DSMO and GA in terms of C_{max} and PD, while DSMO and GA outperform HDSMO in terms of CT. Since HDSMO, SMO and GA can almost obtain near-optimal solutions of the test problems, the difference in the C_{max} and PD values of the three algorithms is not large. Tables 5 & 6 show that HDSMO clearly outperforms DSMO and GA in terms of C_{max} and PD, and GA clearly outperforms DSMO in terms of CT for the medium- and large-scale problems. It can be inferred that HDSMO is the most promising solution method for solving medium- and large-scale problems, and it can obtain better C_{max} and PD within a relatively longer CT than GA, and much shorter CT than SMO. The reason lies in the fact that the HDSMO algorithm balances the exploration and exploitation capabilities of search through three improvements compared with DSMO. To verify whether there is a significant difference in the performance of the four evaluated algorithms, an ANOVA test on the mean was applied to compare the solution approaches, and the results are shown in Table 7. Since the significance is less than 0.05, it indicates that there was a statistically significant difference in the performance of the different solution methods.

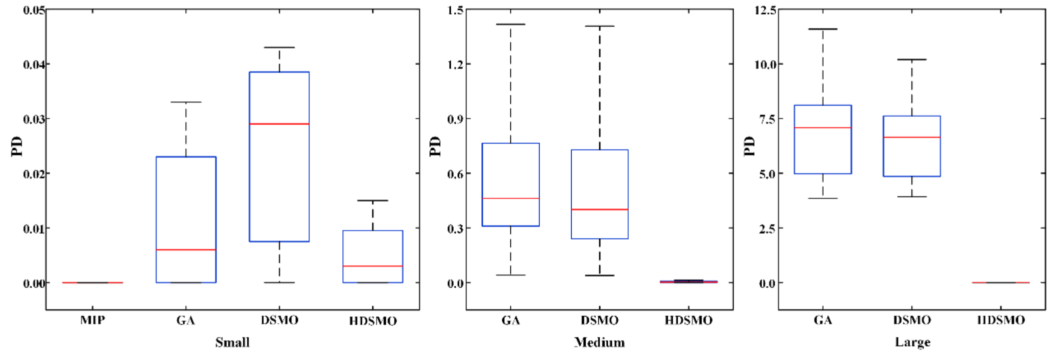
Table 7
Significance test of the PD values of different algorithms (ANOVA: α=0.05)

Algorithms	Sum of Squares	df	Mean Square	F	p value
Between Algorithms	1371.859	2	685.929	104.197	.000
Within Algorithm	8663.222	1316	6.583		
Total	10035.081	1318			

The boxplot diagram at the 95% confidence level for C_{max} and PD are shown in Fig. 9. MIP is just for the small problems. According to the ANOVA test and the boxplot diagram, we can find that the performance of HDSMO is significantly superior to that of the other two algorithms in the stability and near optimality of the solution, especially for the medium- and large-scale problems.



(a)C_{max}



(b) **PD**

Fig. 9. Boxplot diagram of different algorithms

As stated in Section 3.3.3, the neighbourhood search (NS) is designed to balance the exploration and exploitation capabilities of the algorithm. The convergence curves of the five algorithms, GA, DSMO, HDSMO and two hybrid approaches with NS, that is, GA+NS and DSMO+NS, are analyzed to further investigate the effectiveness of NS. The results of classical instances for different scale problems are shown in Fig.10.

It can be seen from the Fig.10 that HDSMO is clearly superior to the other solution approaches in terms of both the C_{max} and convergence speed for all the test problems, and the performance of DSMO and GA is relatively similar with respect to the quality of the solution for medium- and large-scale problems. In addition, DSMO+NS (or GA+NS) outperforms DSMO in terms of both the solution quality and convergence. Generally, it can be inferred that NS can balance the exploration and exploitation capabilities of the DSMO algorithm for the problem proposed in this paper, and HDSMO is superior to the other solution methods with respect to both the solution quality and convergence speed.

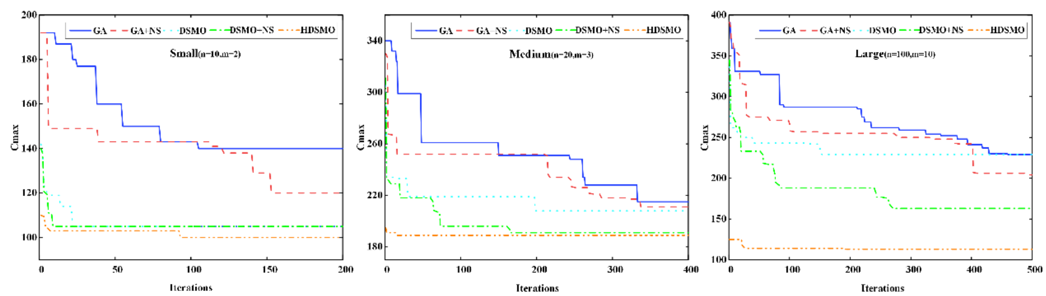


Fig.10. Convergence curves of the algorithms for the three scale problems

5. Conclusions

To jointly optimize production and maintenance scheduling for the unrelated parallel machines problem, in this paper, a MIP model of the problem is constructed, and *Lingo*$^{@}$ is used to solve the small-scale problem. According to the properties of the addressed problem and the “job-allocating and batch-grouping” decision-making method, a hybrid discrete SMO algorithm is proposed to solve medium- and large-scale problems. The experimental results show that the HDSMO algorithm outperforms the GA and DSMO algorithms in terms of the makespan, with slightly more computation time for the medium- and large-scale problems. In addition, the statistical analysis shows that HDSMO is robust to the size of the problems. Meanwhile, a convergence curves comparison of the different algorithms shows that HDSMO is a promising solution method for solving medium- and large-scale problems, and the neighbourhood search method proposed in this paper is effective in improving the convergence of the algorithm.

UPMSPs with maintenance are very common in real-life production environments, where the practical constraints and performance requirements are diverse and complex. Future research will focus on a hybrid DSMO algorithm study for UPMSPs with dynamic event constraints, such as job release times, urgent jobs, sequence-dependent setup times and multi-objective optimization problems. In addition, smart scheduling algorithms integrating the HDSMO and reinforcement learning methods will be investigated to ensure a quick and reasonable response for dynamic events.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (No.51705370).

Disclosure statement

No potential conflict of interest was reported by the authors.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

Avalos-Rosales, O., Angel-Bello, F., Álvarez, A., & Cardona-Valdés, Y. (2018). Including preventive maintenance activities in an unrelated parallel machine environment with dependent setup times. *Computers & Industrial Engineering*, *123*,364-377. https://doi.org/10.1016/j.cie.2018.07.006.

Azizoglu, M., & Kirca, O. (1999). Scheduling jobs on unrelated parallel machines to minimize regular total cost functions. *IIE Transactions*, *31*,153-159. https://doi.org/10.1023/A:1007516602473.

Bansal, J. C., Sharma, H., Jadon, S. S., & Clerc, M. (2014). Spider Monkey Optimization algorithm for numerical optimization. *Memetic Computing*, *6*,31-47. https://doi.org/10.1007/s12293-013-0128-0.

Beldar, P., Moghtader, M., Giret, A., & Ansaripoor, A.H. (2022). Non-identical parallel machines batch processing problem with release dates, due dates and variable maintenance activity to minimize total tardiness. *Computers & Industrial Engineering*, *168*,108135. https://doi.org/10.1016/j.cie.2022.108135.

Chen, Y., Huang, P., Huang, C., Huang, S., & Chou, F.-D. (2021). Makespan minimization for scheduling on two identical parallel machines with flexible maintenance and nonresumable jobs. *Journal of Industrial and Production Engineering*, *38*(4),271-284. https://doi.org/10.1080/21681015.2021.1883131.

Cheng, T.C.E., Hsu, C.-J, & Yang, D.-L. (2011). Unrelated parallel-machine scheduling with deteriorating maintenance activities. *Computers & Industrial Engineering*, *60*(4),602-605. https://doi.org/10.1016/j.cie.2010.12.017.

Cheng, T. C. E., & Sin, C.C.S. (1990). A state-of-the-art review of parallel-machine scheduling research. *European Journal of Operational Research*, *47*(3), 271-292. https://doi.org/10.1016/0377-2217(90)90215-W.

Cheruku, R., Edla, D. R., & Kuppili, V. (2017). SM-RuleMiner: Spider monkey based rule miner using novel fitness function for diabetes classification. *Computers in biology and medicine*, *81*, 79-92. https://doi.org/10.1016/j.compbiomed.2016.12.009.

Dang, Q.-V., van Diessen, T., Martagan, T., & Adan, I. (2021). A matheuristic for parallel machine scheduling with tool replacements. *European Journal of Operational Research*, *291*(2), 640-660. https://doi.org/10.1016/j.ejor.2020.09.050.

Fanjul-Peyro, L., Ruiz, R., & Perea, F. (2019). Reformulations and an exact algorithm for unrelated parallel machine scheduling problems with setup times. *Computers & Operations Research*, *101*,173-182. https://doi.org/10.1016/j.cor.2018.07.007.

Gara-Ali, A., Finke, G., & Espinouse, M.-L. (2016). Parallel-machine scheduling with maintenance: Praising the assignment problem. *European Journal of Operational Research*, *252*(1),90-97. https://doi.org/10.1016/j.ejor.2015.12.047.

Gupta, K., Deep, K., & Bansal, J. C. (2017). Spider monkey optimization algorithm for constrained optimization problems. *Soft Computing*, *21*,6933-6962. https://doi.org/10.1007/s00500-016-2419-0.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Kan, A.H.G.R. (1979). Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Operations Research*, *5*,287-326. https://doi.org/10.1016/S0167-5060(08)70356-X.

Hsu, C.-J., Ji, M., Guo, J.-Y., &Yang, D.-L. (2013). Unrelated parallel-machine scheduling problems with aging effects and deteriorating maintenance activities. *Information Sciences*, *253*,163-169.https://doi.org/10.1016/j.ins.2013.08.053.

Kim, Y. -H., & Kim, R. -S. (2020). Insertion of new idle time for unrelated parallel machine scheduling with job splitting and machine breakdowns. *Computers & Industrial Engineering*, *147*,106630. https://doi.org/10.1016/j.cie.2020.106630.

Lei, D., & He, S. (2022). An adaptive artificial bee colony for unrelated parallel machine scheduling with additional resource and maintenance. *Expert Systems with Applications*, *205*,117577. https://doi.org/10.1016/j.eswa.2022.117577.

Lei, D., & Liu, M. (2020). An artificial bee colony with division for distributed unrelated parallel machine scheduling with preventive maintenance. *Computers & Industrial Engineering*,*141*,106320. https://doi.org/10.1016/j.cie.2020.106320.

Lei, D., & Yang, H. (2022). Scheduling unrelated parallel machines with preventive maintenance and setup time: Multi-sub-colony artificial bee colony. *Applied Soft Computing*,*125*, 109154. https://doi.org/10.1016/j.asoc.2022.109154.

Lei, D., & Yi, T. (2021). A novel shuffled frog-leaping algorithm for unrelated parallel machine scheduling with deteriorating maintenance and setup time. *Symmetry*, *13*(9) ,1574. https://doi.org/10.3390/sym13091574.

Lei, D., Yuan, Y., & Cai, J. (2021). An improved artificial bee colony for multi-objective distributed unrelated parallel machine scheduling. *International Journal of Production Research*, *59*(17),5259-5271. https://doi.org/10.1080/00207543.2020.1775911.

Lei, D., Yuan, Y., Cai, J., & Bai, D. (2020). An imperialist competitive algorithm with memory for distributed unrelated parallel machines scheduling. *International Journal of Production Research*, *58*(2),597-614.https://doi.org/10.1080/00207543.2019.1598596.

Lin, Y. K., Pfund, M. E., & Fowler, J. W. (2011). Heuristics for minimizing regular performance measures in unrelated parallel machine scheduling problems. *Computers & Operations Research*,*38*(6),901-916.https://doi.org/10.1016/j.cor.2010.08.018.

Lin, S. W., Ying, K. C. (2015). A multi-point simulated annealing heuristic for solving multiple objective unrelated parallel machine scheduling problems. *International Journal of Production Research*, *53*(4),1065-1076.https://doi.org/10.1080/00207543.2014.942011.

Lin, S. W., Ying, K. C., Wu, W. J., & Chiang, Y.I. (2016). Multi-objective unrelated parallel machine scheduling: a Tabu-enhanced iterated Pareto greedy algorithm. *International Journal of Production Research*, *54*(4),1110-1121.

https://doi.org/10.1080/00207543.2015.1047981.
Lu, S., Liu, X., Pei, J., T. Thai, M., & M. Pardalos, P. (2018). A hybrid ABC-TS algorithm for the unrelated parallel-batching machines scheduling problem with deteriorating jobs and maintenance activity. *Applied Soft Computing*, *66*,168-182. https://doi.org/10.1016/j.asoc.2018.02.018.

Mumtaz, J., Guan, Z., Yue, L., Zhang, L., & He, C. (2020). Hybrid spider monkey optimisation algorithm for multi-level planning and scheduling problems of assembly lines. *International Journal of Production Research*, *58*(20),6252-6267. https://doi.org/10.1080/00207543.2019.1675917.

Nickabadi, A., Ebadzadeh, M. M., & Safabakhsh, R. (2011). A novel particle swarm optimization algorithm with adaptive inertia weight. *Applied soft computing*, *11*(4), 3658-3670. https://doi.org/10.1016/j.asoc.2011.01.037.

Pang, J, Tsai, Y. -C., & Chou, F.-D. (2021). Feature-extraction-based iterated algorithm to solve the unrelated parallel machine problem with periodic maintenance activities. *IEEE Access*, *9*, 139089-139108. https://doi.org/10.1109/ACCESS.2021.3118986.

Pfund, M., Fowler, J. W., & Gupta, J. N. D. (2004). A survey of algorithms for single and multi-objective unrelated parallel-machine deterministic scheduling problems. *Journal of the Chinese Institute of Industrial Engineers*, *21*(3), 230-241. https://doi.org/10.1080/10170660409509404.

Rocha, P. L., Ravetti, M. G., Mateus, G. R., & Pardalos, P. M. (2008). Exact algorithms for a scheduling problem with unrelated parallel machines and sequence and machine-dependent setup times. *Computers & Operations Research*, *35*(4),1250-1264. https://doi.org/10.1016/j.cor.2006.07.015.

Rodriguez, F. J., Lozano, M., Blum, C., & García-Martínez C. (2013). An iterated greedy algorithm for the large-scale unrelated parallel machines scheduling problem. *Computers & Operations Research*,*40*(7),1829-1841. https://doi.org/10.1016/j.cor.2013.01.018.

Sharma, A., Sharma, A., Panigrahi, B.K., Kiran, D., & Kumar, R. (2016). Ageist spider monkey optimization algorithm. *Swarm and Evolutionary Computation*, *28*,58-77. https://doi.org/10.1016/j.swevo.2016.01.002.

Sharma, A., Sharma, H., Bhargava, A., Sharma, N., & Bansal, J. C. (2017). Optimal placement and sizing of capacitor using Lima limaçon inspired spider monkey optimization algorithm. *Memetic Computing*, *9*,311-331. https://doi.org/10.1007/s12293-016-0208-z.

Tavana, M., Zarook, Y., & Santos-Arteaga, F. J. (2015). An integrated three-stage maintenance scheduling model for unrelated parallel machines with aging effect and multi-maintenance activities. *Computers & Industrial Engineering*, *83*,226-236. https://doi.org/10.1016/j.cie.2015.02.012.

Wang, H., & Alidaee, B. (2019). Effective heuristic for large-scale unrelated parallel machines scheduling problems. *Omega*, *83*,261-274. https://doi.org/10.1016/j.omega.2018.07.005.

Wang, X., Li, Z., Chen, Q., & Mao, N. (2020) Meta-heuristics for unrelated parallel machines scheduling with random rework to minimize expected total weighted tardiness. *Computers & Industrial Engineering*, *145*,106505. https://doi.org/10.1016/j.cie.2020.106505.

Wang, M., & Pan, G. (2019). A novel imperialist competitive algorithm with multi-elite individuals guidance for multi-object unrelated parallel machine scheduling problem. *IEEE Access*, *7*,121223-121235. https://doi.org/10.1109/ACCESS.2019.2937747.

Xia, X., Liao, W., Zhang, Y., & Peng, X. (2021). A discrete spider monkey optimization for the vehicle routing problem with stochastic demands. *Applied Soft Computing*, *111*,107676. https://doi.org/10.1016/j.asoc.2021.107676.

Yang, D.-L., Cheng, T. C. E., Yang, S.-J., & Hsu, C.-J. (2012). Unrelated parallel-machine scheduling with aging effects and multi-maintenance activities. *Computers and Operations Research*, *39*(7),1458-1464.https://doi.org/10.1016/j.cor.2011.08.017.

Yue, L., Guan, Z., Zhang, L., Ullah, S., & Cui, Y. (2019). Multi objective lotsizing and scheduling with material constraints in flexible parallel lines using a Pareto based guided artificial bee colony algorithm. *Computers & Industrial Engineering*, *128*,659-680. https://doi.org/10.1016/j.cie.2018.12.065.

Zhang, X., & Chen, L. (2022). A general variable neighborhood search algorithm for a parallel-machine scheduling problem considering machine health conditions and preventive maintenance. *Computers & Operations Research*, *143*,105738. https://doi.org/10.1016/j.cor.2022.105738.

Zhang, X., Liu, S. C., Lin, W. C., & Wu, C.C. (2020). Parallel-machine scheduling with linear deteriorating jobs and preventive maintenance activities under a potential machine disruption. *Computers & Industrial Engineering*, *145*,106482. https://doi.org/10.1016/j.cie.2020.106482.

© 2023 by the authors; licensee Growing Science, Canada. This is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC-BY) license (http://creativecommons.org/licenses/by/4.0/).