Engineering Optimization
Publication details, including instructions for authors and subscription information:
http://www.tandfonline.com/loi/geno20

Parallel machine scheduling with step-deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm

Peng Guo[a], Wenming Cheng[a] & Yi Wang[b]
[a] School of Mechanical Engineering, Southwest Jiaotong University, Chengdu, PR China
[b] Department of Mathematics, Auburn University at Montgomery, Montgomery, Alabama, USA
Published online: 09 Jan 2015.

To cite this article: Peng Guo, Wenming Cheng & Yi Wang (2015): Parallel machine scheduling with step-deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm, Engineering Optimization, DOI: 10.1080/0305215X.2014.982634

To link to this article: http://dx.doi.org/10.1080/0305215X.2014.982634

Engineering Optimization, 2015
http://dx.doi.org/10.1080/0305215X.2014.982634

Taylor & Francis
Taylor & Francis Group

Parallel machine scheduling with step-deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm

Peng Guo[a], Wenming Cheng[a] and Yi Wang[b]*

[a]*School of Mechanical Engineering, Southwest Jiaotong University, Chengdu, PR China;* [b]*Department of Mathematics, Auburn University at Montgomery, Montgomery, Alabama, USA*

(*Received 2 September 2013; accepted 16 October 2014*)

This article considers the parallel machine scheduling problem with step-deteriorating jobs and sequence-dependent setup times. The objective is to minimize the total tardiness by determining the allocation and sequence of jobs on identical parallel machines. In this problem, the processing time of each job is a step function dependent upon its starting time. An individual extended time is penalized when the starting time of a job is later than a specific deterioration date. The possibility of deterioration of a job makes the parallel machine scheduling problem more challenging than ordinary ones. A mixed integer programming model for the optimal solution is derived. Due to its NP-hard nature, a hybrid discrete cuckoo search algorithm is proposed to solve this problem. In order to generate a good initial swarm, a modified Biskup–Hermann–Gupta (BHG) heuristic called MBHG is incorporated into the population initialization. Several discrete operators are proposed in the random walk of Lévy flights and the crossover search. Moreover, a local search procedure based on variable neighbourhood descent is integrated into the algorithm as a hybrid strategy in order to improve the quality of elite solutions. Computational experiments are executed on two sets of randomly generated test instances. The results show that the proposed hybrid algorithm can yield better solutions in comparison with the commercial solver CPLEX® with a one hour time limit, the discrete cuckoo search algorithm and the existing variable neighbourhood search algorithm.

Keywords: parallel machine scheduling; step-deterioration; sequence-dependent setup time; hybrid cuckoo search; total tardiness

1. Introduction

Production scheduling as a key decision-making process has a great effect on the performance of manufacturing and service systems. There have been a great number of research works concerned with the classical scheduling problem. These studies assumed that the processing time of each job is constant throughout the scheduling period. However, this assumption may not be true in some real industrial settings. Examples can be found in steel production, equipment maintenance, medical emergency planning, *etc.*, where any delay or waiting in starting to process a job may incur extended time for its completion. Such kinds of job are called *deteriorating jobs*. The corresponding scheduling problem was firstly introduced by Gupta and Gupta (1988) and Browne and Yechiali (1990). Since then, many scheduling problems with deteriorating jobs have been extensively studied from various aspects. For details on this stream of research, Alidaee and

*Corresponding author. Email: ywang2@aum.edu

Womer (1999), Cheng, Ding, and Lin (2004) and Gawiejnowicz (2008) provided comprehensive surveys of different models and problems. More recent literature that has explored scheduling problems with deteriorating jobs include Toksari and Guener (2008), Wang, Jiang, and Wang (2009), Bahalke, Yolmeh, and Shahanaghi (2010), Cheng, Lee, and Wu (2011), Huang and Wang (2011), Bank *et al*. (2012), Jafari and Moslehi (2012), Yin *et al*. (2013), Lee, Yang, and Yang (2013), Yang (2013), *etc*.

Among various types of scheduling problems involving deteriorating jobs, those jobs with a *step-deteriorating* effect are considered in this article. A step-deteriorating job means that if a job fails to be processed prior to a pre-specified threshold, its processing time will be extended by adding an extra penalty time. The corresponding single machine scheduling problem may date back to Sundararaghavan and Kunnathur (1994), who firstly gave some solvable cases for minimizing the sum of the weighted completion times. Subsequently, Mosheiov (1995) studied the makespan minimization problem for both single- and multi-machine cases. He also suggested several heuristics for all these complex problems. Subsequently, Jeng and Lin (2004) considered the problem on a single machine with multiple deteriorating dates, and designed a branch-and-bound algorithm for deriving optimal solutions. Cheng and Ding (2001) proved that the total flowtime minimization problem is NP-complete. Jeng and Lin (2005) developed a branch-and-bound algorithm for the problem. Owing to the intractability of the problem, He, Wu, and Lee (2009) used weighted values of the basic processing times, the deteriorating thresholds and the penalties to derive a heuristic solution. Cheng *et al*. (2012) employed a variable neighbourhood search algorithm to minimize the total completion time in a parallel machines environment. Moreover, Layegh, Jolai, and Amalnik (2009) used a memetic algorithm to solve the single machine total weighted completion time problem. Guo, Cheng, and Wang (2014) used a general variable neighbourhood search to solve a single machine total tardiness problem. In particular, batch scheduling with step-deterioration has been researched (Leung, Ng, and Cheng 2008; Mor and Mosheiov 2012). In addition, a similar model which assumes that the processing time of a job is a piecewise linear function (rather than a step function) of the waiting time was introduced by Kunnathur and Gupta (1990). Other related works can be referred to (Kubiak and Van de Velde 1998; Cheng *et al*. 2003; Ji and Cheng 2007; Wu *et al*. 2009; Moslehi and Jafari 2010; Farahani and Hosseini 2013).

Sequence-dependent setup is commonly encountered in manufacturing environments (Allahverdi, Gupta, and Aldowaisan 1999; Allahverdi *et al*. 2008; Joo and Kim 2012). Research on the scheduling of deteriorating jobs with consideration of setup times is relatively uncommon compared with those ordinary problems with setup times. Single machine scheduling problems with past-sequence-dependent setup times and deteriorating jobs were studied by Zhao and Tang (2010), Cheng, Lee, and Wu (2011) and Lai, Lee, and Chen (2011). In their works, the actual processing time of a job is formulated as an increasing function of its scheduled position and the actual processing times of jobs already processed. In addition, Bahalke, Yolmeh, and Shahanaghi (2010) suggested a genetic algorithm and a tabu search for a single machine scheduling problem with the objective of minimizing the makespan. Moreover, the branch-and-bound technique based on some dominance properties has been adopted by Cheng *et al*. (2011) and Lee, Lin, and Shiau (2011) for minimizing the maximum tardiness and the number of late jobs, respectively. *To the best of the authors' knowledge, there is no research that concerns the scheduling problem with step-deteriorating jobs and sequence-dependent setup times*.

The parallel machine scheduling problem, as a generalization of the single machine problem, has been widely studied in the literature because it frequently appears in industrial production (Kaplan and Rabadi 2013; Chung, Liao, and Smith 2014). In this article, the parallel machine scheduling problem is studied in which the two effects of both step-deterioration and sequence-dependent setup times are considered. The objective of the parallel machine scheduling problem with step-deteriorating effect and setup times (PMSDST) is to minimize the total tardiness. Since

the minimization of the total tardiness on uniform parallel machines with sequence-dependent setup times is NP-hard (Armentano and De Franca Filho 2007), the PMSDST, as a more general problem, is also NP-hard. Though the PMSDST can be formulated as a mathematical programming model, the model is impossible to solve in a reasonable amount of computational time when the problem size increases. Recently, various novel population-based intelligence optimization algorithms (Yang and Deb 2010; Civicioglu 2012, 2013) have been designed to solve continuous problems and perform very well in obtaining near-optimal solutions. However, using these new swarm optimization algorithms to solve scheduling problems is very rare. In this article, a hybrid meta-heuristic algorithm based on the cuckoo search is introduced for the PMSDST under study to obtain optimal or near-optimal solutions. Computational results and comparisons to other existing algorithms demonstrate the effectiveness and efficiency of the proposed meta-heuristic algorithm.

The rest of the article is organized as follows. The next section describes the problem in more detail, and formulates a mixed integer programming model. Section 3 proposes the search heuristic suggested in the present study together with methods to generate initial and neighbourhood solutions. Then the computational results are reported and discussed in Section 4. Finally, Section 5 gives some conclusions along with remarks for future research.

2. Problem formulation

In the PMSDST, a set of n independent jobs has to be processed on m identical parallel machines. All machines are available at time zero, and each machine can handle at most one job at a time. Preemption, division or cancellations are not allowed. For job j, its actual processing time p_j is a nonlinear step function of its starting time s_j and the deteriorating date h_j. If the starting time s_j of job j is less than or equal to the corresponding threshold h_j, then job j only requires a basic processing time a_j. Otherwise, an extra penalty b_j is incurred. Each job j is given a due date d_j. In addition, a sequence-dependent setup time for a job is also considered in this problem. That is to say, there is a setup time δ_{ij} when job i immediately precedes job j on the same machine. Obviously, $\delta_{jj} = 0$. In addition, it is assumed that $\delta_{0j} = 0$, which indicates that no setup time is needed if job j is processed at the first position on a machine. Without loss of generality, all parameters are assumed to be integers. The tardiness T_j of a job j can be determined by $T_j = \max\{0, C_j - d_j\}$, where C_j denotes the completion time of job j. Finally, the goal of the scheduling problem considered here is to find a schedule that specifies an assignment of all jobs to various machines and the schedule of those jobs on all machines so that the total tardiness is minimized.

Then, a mixed integer programming (MIP) model is proposed to minimize the total tardiness for the PMSDST. The basic structure of the MIP model is motivated by a network model formulated by Radhakrishnan and Ventura (2000). In order to present constraints conveniently, two dummy jobs 0 and $n + 1$ are introduced on each machine and are scheduled at the head and tail of its job sequence, and their processing times are set to 0. Additionally, the following notation is defined.

u_{ij}^k binary decision variable equal to one if job i immediately precedes job j on machine k, and zero otherwise

$\sum_{j=1}^{n} T_j$ total tardiness

M large positive constant such that $M \to \infty$ as $n \to \infty$ ($M = \max_{j=1,\ldots,n} d_j + \sum_{j=1}^{n}(a_j + b_j)$ in this article)

Applying the above notation and variables, the PMSDST can be formulated as follows.

Objective function:

$$\min \quad Z = \sum_{j=1}^{n} T_j \tag{1}$$

Subject to:

$$p_j = \begin{cases} a_j, & s_j \leqslant h_j \\ a_j + b_j, & \text{otherwise} \end{cases}, \quad \forall j = 1, \ldots, n \tag{2}$$

$$\sum_{i=1}^{n} u_{0i}^{k} = 1, \quad \forall k = 1, \ldots, m \tag{3}$$

$$\sum_{i=1}^{n} u_{i(n+1)}^{k} = 1, \quad \forall k = 1, \ldots, m \tag{4}$$

$$\sum_{i=0}^{n} \sum_{k=1}^{m} u_{ij}^{k} = 1, \quad \forall j = 1, \ldots, n, i \neq j \tag{5}$$

$$\sum_{j=1}^{n+1} \sum_{k=1}^{m} u_{ij}^{k} = 1, \quad \forall i = 1, \ldots, n, i \neq j \tag{6}$$

$$s_j \geqslant \delta_{0j} + M(u_{0j}^{k} - 1), \quad \forall j = 1, \ldots, n, \ \forall k = 1, \ldots, m \tag{7}$$

$$s_j \geqslant C_i + \delta_{ij} + M(u_{ij}^{k} - 1), \quad \forall i, j = 1, \ldots, n, \ \forall k = 1, \ldots, m \tag{8}$$

$$C_j \geqslant s_j + p_j, \quad \forall j = 1, \ldots, n \tag{9}$$

$$T_j \geqslant C_j - d_j, \quad \forall j = 1, \ldots, n \tag{10}$$

$$u_{ij}^{k} \in \{0, 1\}, \quad \forall i, j = 1, \ldots, n, \ \forall k = 1, \ldots, m \tag{11}$$

$$s_j, T_j \geqslant 0, C_j > 0 \quad \forall j = 1, \ldots, n. \tag{12}$$

In the above formulation, Equation (1) is the objective function that minimizes the sum of the total tardiness. Constraints (2) define the actual processing time of a job under consideration of the step-deteriorating effect. Constraints (3) ensure that the dummy job 0 is positioned at the beginning of the sequence before all the real jobs on each machine. Constraints (4) guarantee that the dummy job $n + 1$ is processed just after all real jobs on each machine. Constraints (5) and (6) ensure the precedence relation of jobs assigned to the same machine. Constraints (7) state that the starting time for the first job of each machine must be equal to or greater than its initial setup time. For other jobs j, the starting time s_j is determined by constraints (8) and is at least the sum of the completion time of job i and the setup time from i to j. If i is not the predecessor of j, the subtraction of M makes constraints (8) non-restrictive. Constraints (9) calculate the completion time of each job. For each job j ($j = 1, \ldots, n$), the completion time C_j is equal to its starting time s_j plus its actual processing time p_j. Constraints (10) ensure that only the positive value of lateness can be considered as tardiness, that is, $T_j = \max\{0, C_j - d_j\}$. Constraints (11) declare u_{ij}^{k} as binary variables. Finally, constraints (12) state that the starting time, completion time and tardiness of job j are non-negative.

The above mathematical programming model is used to find optimal solution for the present problem. However, due to the NP-hardness of the proposed problem, it is difficult to solve

optimally for medium- and large-sized instances. This is the main characteristic of most complex scheduling problems, which makes it necessary to develop some meta-heuristic algorithms for solving the problem under study. In the next section, a hybrid cuckoo search algorithm is proposed for the intractable PMSDST.

3. Hybrid discrete cuckoo search algorithm

The cuckoo search (CS) algorithm is one of the newest swarm intelligence optimization algorithms introduced by Yang and Deb (2009, 2010), inspired by the intelligent breeding behaviour of some species of cuckoo. These species lay their eggs in the nests of host birds (of other species) and have amazing abilities such as being able to select nests with recently laid eggs which they remove so that the hatching probability of their own eggs is increased. On the other hand, some host birds are able to engage in direct contest with infringing cuckoos. For instance, if the alien eggs are discovered by the host bird, it may either throw out these eggs or abandon the nest completely. This natural phenomenon has led to the evolution of cuckoo eggs that mimic the appearance of local host birds' eggs.

For simplicity in the implementation of the algorithm, the following three idealized rules are used (Yang and Deb 2009).

(1) Each cuckoo lays one egg at a time, in a randomly chosen nest.
(2) The best nests with high quality eggs will carry over to the next generation.
(3) The number of available host nests is fixed, and the egg laid by a cuckoo is discovered by the host bird with a probability $\rho_a \in [0, 1]$. In this case, the host bird can either discard the egg or abandon the nest so as to build a new nest in a new location.

The last assumption can be approximated by a fraction ρ_a of the n nests being replaced by new ones.

The CS algorithm contains a population of $\mathcal{P}$ eggs. Egg $\imath \in \mathcal{P}$ is in nest $X_\imath$, which corresponds to a solution of the underlying problem. The word 'nest' and the word 'solution' are used interchangeably herein. In the initial phase, $\mathcal{P}$ solutions are generated randomly. Then all of these solutions except the one with the smallest objective value are replaced by new solutions generated by Lévy flights. Specifically, if t is the current iteration index, and let $\Phi^t := \{X_j^t : j \in \mathcal{P}\}$ be the set of all solutions at the tth iteration, then a candidate solution $\tilde{X}_\imath^{(t+1)}$ is generated by the equation

$$\tilde{X}_\imath^{(t+1)} = X_\imath^{(t)} + \alpha_0 \left(X_j^{(t)} - X_\imath^{(t)}\right) \cdot t^{-\lambda}, \tag{13}$$

where the constant parameter α_0 is related to the scale of the underlying problem, $X_j^{(t)}$ represents a randomly selected solution at the tth iteration and λ is a constant such that $1 < \lambda \leq 3$.

The vector $\alpha := \alpha_0(X_j^{(t)} - X_\imath^{(t)})$ introduces a difference from the current solution $X_\imath^{(t)}$. This difference mimics the scenario in which a cuckoo's egg is more difficult to recognize by a host if it is more similar to the host's egg. The term $t^{-\lambda}$ accounts for the heavy-tailed power law distribution Lévy(λ) given by

$$\text{Lévy}(\lambda) \sim t^{-\lambda}. \tag{14}$$

The distribution Lévy(λ) has infinite variance and mean when $1 < \lambda \leq 3$, thus Lévy flight essentially provides a random walk process in the hope of searching for a better solution.

If the host bird identifies a cuckoo's egg in its nest (with probability ρ_a), either the alien egg will be discarded by the host or the host will abandon the nest. In either case, the cuckoo's egg will fail to hatch. Thus the cuckoo has to explore a new nest (solution) in order to hatch its

egg successfully. This behaviour will be simulated by the crossover operation. Specifically, let rand_ι, rand be two uniformly distributed random numbers on the interval $[0, 1]$. A new candidate solution V_ι^{t+1} is obtained by the equation

$$V_\iota^{t+1} = \begin{cases} \tilde{X}_\iota^{t+1} + \text{rand} \cdot (\tilde{X}_{\gamma_1}^{t+1} - \tilde{X}_{\gamma_2}^{t+1}) & \text{if } \text{rand}_\iota > \rho_a \\ \tilde{X}_\iota^{t+1} & \text{otherwise,} \end{cases} \tag{15}$$

where γ_1 and γ_2 are two randomly chosen indices such that ι, γ_1 and γ_2 are pairwise distinct. If the candidate solution V_ι^{t+1} has a better quality than the old one X_ι^t, that is, with a smaller objective function value, then V_ι^{t+1} will be accepted as X_ι^{t+1}. Therefore, for $\iota \in \mathcal{P}$, the update process of X_ι at the $(t+1)$th iteration is defined by Equation (16):

$$X_\iota^{t+1} = \begin{cases} V_\iota^{t+1}, & \text{if } f(V_\iota^{t+1}) < f(X_\iota^t) \\ \tilde{X}_\iota^{t+1}, & \text{otherwise,} \end{cases} \tag{16}$$

where f is the objective function of the underlying optimization problem. Afterwards, the best solution among all solutions found so far up to the present iteration is recorded to memory. The same procedure repeats until a termination criterion is met.

The CS algorithm provides more robust and precise results than other meta-heuristic algorithms, such as particle swarm optimization, differential evolution, or artificial bee colony algorithms in solving the continuous optimization problems (Civicioglu and Besdok 2013). Therefore, the CS algorithm has aroused much interest and has been successfully applied to different kinds of optimization problems (Walton *et al*. 2011; Gandomi, Yang, and Alavi 2013; Valian *et al*. 2013; Yildiz 2013; Valian and Valian 2013; Kanagaraj *et al*. 2013). Recently, Ouaarab, Ahiod, and Yang (Online First 2013) proposed a discrete CS algorithm to solve the travelling salesman problem. Burnwal and Deb (2013) developed a CS-based approach for scheduling optimization of a flexible manufacturing system. Li and Yin (2013) designed a hybrid algorithm that combines CS and a fast local search to solve a permutation flow shop scheduling problem.

In the remainder of this section, a hybrid discrete CS algorithm (HDCS) is designed to solve the considered PMSDST. In order to maintain the remarkable characteristics of the standard CS, the Lévy flight equation (13) and the crossover operator (15) are redefined based on an integer encoding scheme. Moreover, a local search procedure based on variable neighbourhood descent is employed to refine elite solutions that are not discarded. In addition, a modified Biskup–Hermann–Gupta (BHG) heuristic called MBHG is incorporated into the random initialization to generate a population of initial solutions of certain quality. Furthermore, to avoid the algorithm being trapped into a local optimum, a restarting strategy is embedded into the search process. The details of the proposed hybrid discrete algorithm are elaborated below.

3.1. *Solution representation*

In a meta-heuristic, the solution performance is highly dependent upon the representation of a solution. The most commonly used solution representation for parallel machine scheduling problems is having an array of jobs for each machine that represent the processing order of jobs assigned to that machine. Nevertheless, in order to maintain the simplicity and the powerful search performance of the CS algorithm, a solution is represented by a single sequence of n jobs in this article. *With such a sequence, an actual schedule can be built simply by assigning the next unscheduled job to the earliest available machine until all jobs are scheduled*. When multiple machines are available for an unscheduled job, the machine with the smallest index will

Table 1. The job data of the example with six jobs and two machines.

Job (j)	1	2	3	4	5	6
Basic processing time a_j	78	17	97	93	62	53
Due date d_j	85	48	229	133	220	75
Deteriorating date h_j	70	4	62	19	58	39
Penalty b_j	18	33	1	17	40	31

Table 2. Sequence-dependent setup times in the example with six jobs and two machines.

δ_{ij}	1	2	3	4	5	6
1	0	9	9	5	4	6
2	5	0	8	2	2	5
3	2	6	0	5	4	7
4	3	5	10	0	3	9
5	3	10	6	9	0	5
6	5	8	4	8	4	0

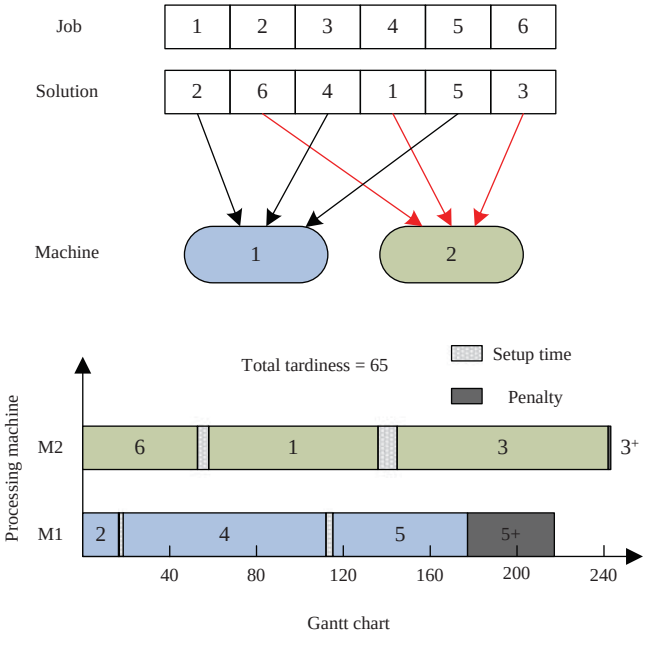

Figure 1. The decoding scheme and the Gantt chart of a solution.

be chosen. The obtained sequence of jobs will be called a solution vector or a nest vector. Once an actual schedule is constructed, the total tardiness can be easily calculated subsequently.

To illustrate how the nest vector is used to construct an actual schedule, below a simple example is given with *six* jobs and *two* machines. The detailed data for this problem are provided in Tables 1 and 2. Suppose a nest vector for the problem is given by $X = [2, 6, 4, 1, 5, 3]$. An actual schedule can be obtained through the aforementioned decoding scheme, as shown in Figure 1. Correspondingly, the total tardiness is 65. In Figure 1, jobs 2, 4 and 5 are assigned to machine 1 and jobs 6, 1 and 3 are assigned to machine 2.

3.2. *Population initialization*

In the original CS algorithm, the initial population of solutions is often filled with $\mathcal{P}$ n-dimensional vectors that are randomly generated. In order to ensure an initial population with certain quality and diversity, the BHG heuristic (Biskup, Herrmann, and Gupta 2008) is modified to generate an initial solution for our problem, whereas the remaining $\mathcal{P} - 1$ solutions are initialized with random job permutations.

Generally, a job j is likely to be scheduled in an earlier position if it has a smaller value of due date d_j and a smaller value of deteriorating date h_j for our problem. Therefore, in the modified BHG heuristic, a predefined list of jobs is obtained in ascending order of their weighted values $\omega d_j + (1-\omega)h_j$, where $0 < \omega < 1$. Moreover, in order to reduce computation time, the first m jobs from the predefined list are simply assigned simultaneously to the m machines, respectively, such that the jth job, $1 \leq m$, is assigned to the jth machine. The details of the modified BHG (MBHG) heuristic algorithm are described next.

(1) Obtain a list of all jobs in ascending order of the weighted value of due date and deteriorating date, *i.e.* $\omega d_j + (1-\omega)h_j \leqslant \omega d_{j+1} + (1-\omega)h_{j+1}$ for $j = 1, \ldots, n-1$.
(2) The first m jobs from the list are simultaneously assigned to the m machines, respectively, such that the jth job, $1 \leq m$, is assigned to the jth machine.
(3) Let $U = \{m+1, \ldots, n\}$ be the set of unscheduled jobs.
(4) Determine to which machine the job with the smallest index in U should be assigned. Starting from machine 1, take the job with the smallest index in U and insert it before and after each job that is already assigned to machine 1. The insertion into the schedule of machine 1 may start from the position after the last job and continue sequentially backward to the position before the first job assigned to machine 1. For each instance of inserting a job into the current schedule of machine 1, calculate a new value of the total tardiness. The same procedure is applied to machines $2, 3, \ldots, m$ afterwards. After all possible insertions into the schedule of each machine are done, the algorithm chooses the sequence with the lowest total tardiness which was calculated first. Remove the first job from the set U.
(5) Repeat Step (4) until U is empty.
(6) Output the final schedule on the m machines and its total tardiness.

It is found that the weight ω has a significant influence on the solution quality of the MBHG heuristic. When the weight is $\omega = 0.1$, the solution provided by the MBHG heuristic is 116 for the above mentioned instance listed in Table 1. When the weight is $\omega = 0.5$, the solution obtained by the MBHG heuristic is 65 for the same instance. It is hard to find the best value of weight ω. This motivates a search with different values of ω. In this study, the weight ω is varied from 0.1 to 0.9. The best solution with the least total tardiness among the solutions obtained from different weights is selected as the final output of the MBHG heuristic.

3.3. *CS-based search*

The search process of the original CS algorithm (Equations 13, 15 and 16) may fall into a local optimal solution, particularly in solving a discrete optimization problem. Therefore, in order to maintain diversity of the population, the following search strategies are adopted.

For the Lévy flight, a uniformly distributed random variable ψ is firstly generated in the interval (0, 1). Let X^t_{best} be the best solution found among all solutions in Φ^t. If $\psi > 0.5$, the generation of a new solution is directed by the current best solution X^t_{best}. Otherwise, the searching process is around a randomly selected solution X^t_j from the current population Φ^t. Parameter λ is critical in delivering a new solution, as observed from Equation (13). A smaller λ increases

the global exploration ability while a large λ tends to improve the local exploitation ability of solutions in the current search area. Therefore, to balance the two abilities, λ should be linearly increased from a relatively small value to a relatively large value in the entire process of the HDCS algorithm. By doing so, HDCS can possess good global search ability at the beginning phase of the search while having more local search ability near the end of the iterations. In this article, parameter λ is chose to vary through the formula

$$\lambda = \lambda_{\min} + \frac{\lambda_{\max} - \lambda_{\min}}{t_{\max}} t, \tag{17}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the initial value and the terminal value of λ, respectively, and $t_{\max}$ is the maximum number of iterations. In our algorithm, $\lambda_{\min} = 1.1$ and $\lambda_{\max} = 3$.

Based on the above modification, Equation (13) for producing a new solution $\tilde{X}_l^{t+1}$ is replaced by the following equation:

$$\tilde{X}_l^{t+1} = \begin{cases} X_l^t + \alpha_0 \cdot (X_l^t - X_{\text{best}}^t) \cdot t^{-\lambda}, & \psi > 0.5 \\ X_l^t + \alpha_0 \cdot (X_l^t - X_j^t) \cdot t^{-\lambda}, & \text{otherwise.} \end{cases} \tag{18}$$

In this article, α_0 is set to one.

Since the standard CS algorithm is originally designed for solving a continuous optimization problem, the operations such as subtraction, multiplication and addition involved in Lévy flight must be redefined for a discrete scheduling problem. These operations will be redefined by some neighbourhood search strategies to suite a discrete problem. Next, the discrete operators used in Equation (18) are described in detail.

Subtraction

The present authors' algorithm involves one subtraction operation between two solutions, say, the minuend X_1 and the subtrahend X_2. Since the solution is a job permutation, a comparison strategy is carried out for producing a new solution chain X'. If the given elements of two solutions are identical, the corresponding element of the new chain is set equal to the number zero. Otherwise, the element of the new chain is copied from the corresponding one of X_1. Figure 2 shows the redefined subtraction operation.

Multiplication

The multiplication operator carries out the product of the scalar $\sigma := t^{-\lambda}$ and a solution chain X' obtained by the subtraction operation. Let X'' denote the output of the multiplication. In the redefined manner of the multiplication operation, a uniformly distributed random number between zero and one is generated for each element in the chain X'. The random number is then compared with σ. If the random number is more than or equal to σ, the element in the offspring chain X'' is set equal to the element of the same position in the chain X'; otherwise,

X_1	6	4	1	5	3	2
X_2	1	4	2	6	3	5
X'	6	0	1	5	0	2

Figure 2. The subtraction operation $X_1 - X_2$.

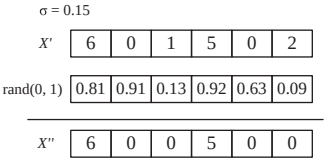

Figure 3. The multiplication operation.

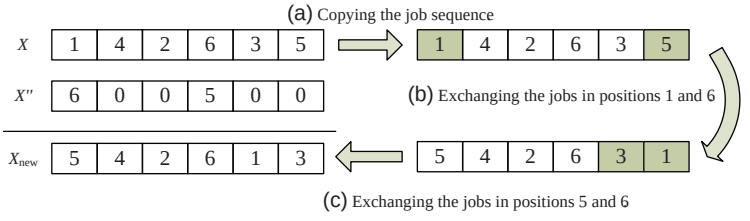

Figure 4. The addition operation.

its corresponding position is occupied by the number zero. Figure 3 represents the process of the multiplication operation on the scalar σ and a solution chain.

Addition

The addition operation is modified by a number of swap moves. Two objects manipulated in the operation are the current solution X and the chain X'' produced from the multiplication. Once the addition operation is finished, a new solution X_{new} is generated. Initially, X_{new} is set equal to the incumbent X. Then for each element $X''(i)$ of X'', if $X''(i)$ is more than 0, the job in position $X''(i)$ of X_{new} and the job in position $X_{\text{new}}(i)$ of X_{new} are swapped. Figure 4 illustrates the manner in which the addition performs.

As can be seen, the Lévy flight is redefined by the above three discrete operators. Likewise, the crossover operation described by Equation (15) is replaced by the *order crossover operator*. The order crossover (Davis 1985) has shown robust performance in the genetic algorithm. For each solution, a random number is generated and compared with the probability ρ_a to determine whether the crossover operator should be executed. If the random number is greater than ρ_a, the solution is selected as a parent for producing a new offspring. Another solution is randomly chosen from the population of solutions and regarded as a second parent. With the two parent solutions chosen, the order crossover operation is performed as below.

(1) Select randomly a subsequence from one parent, called the first parent X_1.
(2) Produce a proto-child by copying the subsequence into its corresponding position.
(3) Check all the jobs of the other parent, called the second parent X_2, from left to right. If a job is already assigned in the subsequence from X_1, then skip to the next job in X_2 that is not scheduled yet and assign it to the first available position of the new solution. The jobs taken from X_2 are the ones that the proto-child needs.

The above crossover operator will produce two offspring solutions because each parent may be chosen as the first parent. The incumbent solution will be updated by the better of the two offspring in the next generation. The procedure is illustrated in Figure 5.

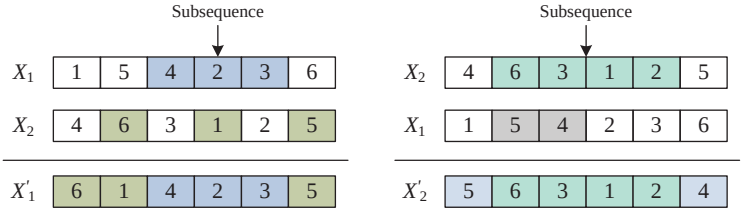

Figure 5. The order crossover operation.

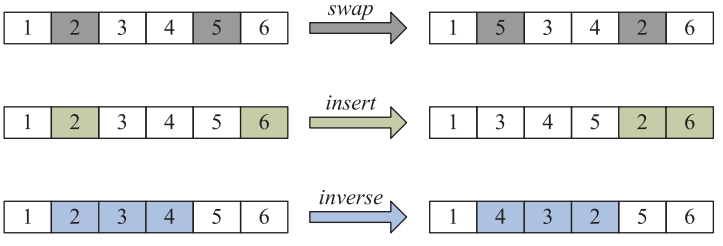

Figure 6. The *swap*, *insert* and *inverse* operations for local search.

3.4. *Local search*

To improve the performance, HDCS implements variable neighbourhood descent (VND) as a local search strategy for a set of elite solutions. The elite set is constructed by finding the best number of solutions τ from the population in each iteration. The preset value of τ can be chosen by the following equation:

$$\tau = \max\{3, \text{round}[\mathcal{P} \times (1 - \rho_a)]\}. \tag{19}$$

Equation (19) ensures that there are at least three elite solutions. If there are too few elite solutions, the local search strategy cannot improve the performance of the algorithm effectively. On the other hand, too many elite solutions may consume too much computational time. Each elite solution indicates a promising region to search for the optimal solution. Hence, the VND local search increases the possibility of obtaining an optimal solution from diverse areas of the solution space. In our implementation, three neighbourhood structures based on *swap*, *insert* and *inverse* operations are used. The swap operation is done by randomly selecting the *i*th and *j*th jobs from an elite solution and then swapping them directly. The insert operation is done by choosing job *i* at random from a solution and inserting it into the position immediately preceding a randomly chosen job *j* of the solution. The inverse operation is done by inverting the subsection between two randomly chosen jobs of a solution. Each of the three neighbourhood operations is repeated *n* times when it is executed. An example of each operation is shown in Figure 6.

The proposed variable neighbourhood descent is applied to enhance the quality of an elite solution. Each elite solution is regarded as a seed permutation. The search process repeats on each neighbourhood operation for the best local optimum until no improvement appears. If the local optimum is better than the seed solution, the seed solution is updated and the search continues in the neighbourhood; otherwise, the search moves to the next neighbourhood and continues to search there. The pseudo-code of the suggested local search is summarized in Algorithm 1.

3.5. *Restarting strategy and stopping criterion*

To maintain the diversity of the population of solutions, it is common to use a restarting strategy in evolutionary algorithms (Ruiz, Maroto, and Alcaraz 2006). In the present proposed algorithm,

Algorithm 1 Local search

Require: Elite solutions for the HDCS algorithm
Ensure: Improved elite solutions for the HDCS algorithm
for each solution X in the elite set **do**
κ=1;
while $\kappa \leqslant 3$ **do**
if $\kappa == 1$ **then**
Apply *swap* operations n times to X to obtain a local optimum X';
else if $\kappa == 2$ **then**
Apply *insert* operations n times to X to obtain a local optimum X';
else if $\kappa == 3$ **then**
Apply *inverse* operations n times to X to obtain a local optimum X';
end if
if X' is better than X **then**
$X \leftarrow X'$;
Continue the kth neighbourhood search operation;
else
$\kappa = \kappa + 1$;
end if
end while
end for

partial solutions adopt a restarting mechanism at each iteration. Firstly, the population is sorted in ascending order of objective function values. Then the first 90% of solutions from the sorted list are retained in the population. The remaining 10% of solutions will be replaced with newly generated nests by random job permutations.

Regarding the stopping criterion, two commonly used termination conditions are employed in the authors' algorithm proposed herein. That is to say, if the number of iterations t exceeds the maximum number of iterations $t_{\max}$ or the best solution X_{best} is not improved upon in t_{nip} successive iterations, the procedure is terminated; otherwise, a new iteration will start and the iteration counter t is increased by one. When the algorithm terminates, the best solution found X_{best} and its total tardiness are output.

3.6. *Outline of the HDCS algorithm*

The search framework of the proposed HDCS algorithm for solving the PMSDST is shown in Figure 7. The algorithm starts with an initial population of $\mathcal{P}$ solutions, among which one solution is generated by the MBHG heuristic, and the remainder are generated by random permutations of n jobs. Then the solution X_{best} with the smallest objective value is identified by calculating the function value of each solution. Subsequently, the algorithm iterates search for a better solution until a stopping condition is met. At each iteration, the first τ elite solutions are determined according to Equation (19) by their smaller objective function values. The remaining solutions are regarded as *normal solutions*. For each of the normal solutions, modified Lévy flight by Equation (18) is performed to search for a new solution either around X_{best} achieved in the previous iteration or a randomly selected neighbour. The elite solutions are improved by a local search based on variable neighbourhood descent. Then, a fraction ρ_a of the solutions are replaced by the new candidate solutions produced by performing the order crossover operation. To avoid premature convergence, the restarting strategy is utilized for the 10% worst solutions. All the

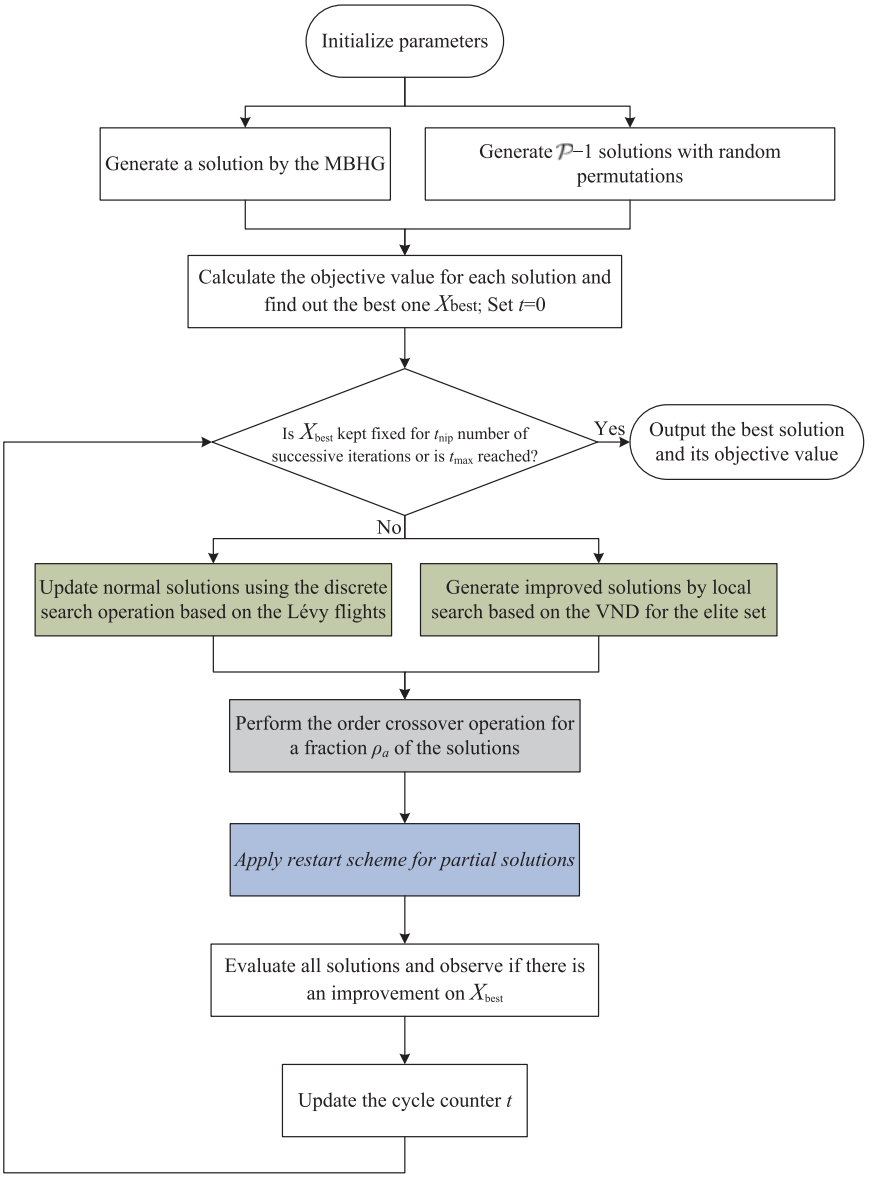

Figure 7. The flowchart for the proposed hybrid cuckoo search algorithm.

solutions are evaluated for their objective values and X_{best} is updated if a new better solution is found. The algorithm is then continued to the next iteration. When the algorithm is terminated, the best solution found by the algorithm and its objective value are output for the problem under study.

4. Computational experiments

There is no study on the parallel machine scheduling problem with step-deterioration jobs and setup times in the existing literature. Therefore, our article seems to be the first study of this problem. In order to evaluate the effectiveness of the proposed algorithm, the performance of

the HDCS is compared with the discrete cuckoo search algorithm without local search strategy (DCS) and the variable neighbourhood search proposed by Cheng *et al.* (2012) (VNS). The three algorithms are coded in MATLAB® 7.11 and executed on an Intel® Pentium® dual-core 2.6 GHz PC with 2 GB RAM under a Windows® XP environment. The performance of each algorithm is measured by the relative percentage deviation (RPD) calculated by the formula

$$\mathrm{RPD}(\%) = \frac{\mathrm{Alg}_{\mathrm{sol}} - \mathrm{Min}_{\mathrm{sol}}}{\mathrm{Min}_{\mathrm{sol}}} \times 100 \tag{20}$$

for each test instance, where $\mathrm{Alg}_{\mathrm{sol}}$ is the objective function value obtained for a given algorithm and $\mathrm{Min}_{\mathrm{sol}}$ is the best solution of all experiments for each test problem.

4.1. *Data generation of test problems*

In order to analyse the performance of our algorithm for the scheduling problem under consideration, different sizes of test instances are needed. In this section, two sets of instances are defined. For small-sized instances, the following combinations of the number n of jobs and the number m of machines are tested: $n = \{8, 10, 12, 14\}$ and $m = \{2, 3, 4\}$. For large instances, the combinations are $n = \{30, 40, 50, 60\}$ and $m = \{4, 6, 8\}$. In all cases, the following rules are used to generate job data.

The basic processing time a_j of job j is generated from a discrete uniform distribution on the interval $[1, 100]$. The penalty b_j is assumed to be drawn from the uniform distribution $U[1, 100 \times \xi]$, where $\xi = 0.5$ is chosen in the present study. Set $\beta = \sum_{j=1}^{j=n} a_j/m$. The deteriorating dates are generated randomly by a discrete uniform distribution on each of the intervals $\mathrm{H}_1 := [1, 0.5 \times \beta]$, $\mathrm{H}_2 := [0.5 \times \beta, \beta]$ and $\mathrm{H}_3 := [1, \beta]$. The setup times are produced according to a uniform distribution in the range $[1, 10]$. Moreover, the due dates of jobs are randomly generated from a uniform distribution on the interval $[1, \bar{C}_{\max}]$, where $\bar{C}_{\max}$ is the value of the maximal completion time obtained by the following rule: first of all, a job permutation is achieved by arranging the jobs in non-decreasing order of the ratio a_j/b_j. Then, a corresponding schedule with no idle times can be constructed by assigning unscheduled jobs to the earliest available machine. Once a schedule is built, the maximal completion time $\bar{C}_{\max}$ is calculated. For each possible combination, 10 random replicates are generated. Therefore, there are 360 small instances and 360 large instances to test the algorithms.

4.2. *Parameter calibration*

Parameter selection can significantly affect the quality of a algorithm. In this study, suitable parameter values were selected taking into consideration both solution quality and computational efficiency. To determine appropriate values of parameters, extensive computational tests were performed on 27 instances, including both small-sized and large-sized ones. These instances are generated from the above defined rule with the following parameter values: $\mathcal{P} = 15, 20, 25, 30, 35, 40$; $\rho_a = 0.6, 0.7, 0.8, 0.9$; $t_{\max} = 100, 200, 300, 400$; $t_{\mathrm{nip}} = 30, 50, 70$. The preliminary tests using these parameter values show that the following set of parameter values seems to provide the best performance within a reasonable computational time: $\mathcal{P} = 30$, $\rho_a = 0.8$, $t_{\max} = 200$ and $t_{\mathrm{nip}} = 50$. Hence, this set of parameter values will be adopted for all subsequent experiments in this study.

Table 3. Average relative percentage deviation (RPD) of small instances for the algorithms.

Instance		MIP	VNS			DCS			HDCS		
			RPD_B	RPD_A	RPD_W	RPD_B	RPD_A	RPD_W	RPD_B	RPD_A	RPD_W
H_1	8×2	0.00	0.00	0.22	0.31	0.31	1.63	4.79	0.00	0.00	0.00
	8×3	0.00	0.00	0.00	0.00	0.00	1.50	5.72	0.00	0.00	0.00
	8×4	0.00	0.00	0.00	0.00	0.00	0.72	1.21	0.00	0.00	0.00
	10×2	0.00	0.44	0.96	1.99	6.43	11.13	14.54	0.05	0.05	0.05
	10×3	0.00	0.00	0.13	0.45	1.14	5.41	7.35	0.00	0.01	0.08
	10×4	0.00	0.00	0.34	1.50	0.58	3.83	5.79	0.00	0.00	0.00
	12×2	0.36	0.00	0.72	2.32	15.05	18.06	19.87	0.00	0.00	0.00
	12×3	0.00	0.14	1.76	5.83	14.26	17.11	18.97	0.00	0.09	0.22
	12×4	0.00	0.24	2.58	5.18	56.32	58.86	60.87	0.00	0.02	0.03
	14×2	12.34	0.00	2.74	8.02	53.13	55.09	55.80	0.00	0.41	0.92
	14×3	1.13	0.12	1.44	3.13	19.91	23.26	24.60	0.00	0.26	0.57
	14×4	1.51	0.02	1.14	2.71	19.60	20.75	21.83	0.00	0.02	0.11
Average		1.28	0.08	1.00	2.62	15.56	18.11	20.11	0.00	0.07	0.17
H_2	8×2	0.00	0.00	0.29	0.59	0.11	0.58	0.75	0.00	0.00	0.00
	8×3	0.00	0.00	0.00	0.00	0.00	0.04	0.18	0.00	0.00	0.00
	8×4	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
	10×2	0.00	0.00	0.23	0.65	1.39	1.64	1.99	0.00	0.00	0.00
	10×3	0.00	0.00	0.15	0.33	7.00	8.24	9.96	0.00	0.00	0.00
	10×4	0.00	0.00	0.04	0.06	1.03	4.51	6.83	0.00	0.00	0.00
	12×2	0.87	0.00	2.73	10.07	25.28	26.65	27.42	0.00	0.09	0.23
	12×3	0.23	0.18	1.97	4.18	8.00	9.11	9.66	0.00	0.39	1.05
	12×4	0.00	0.00	1.02	1.86	4.87	6.94	8.76	0.00	0.10	0.33
	14×2	4.58	0.03	1.15	3.13	31.56	36.69	38.33	0.00	0.58	2.29
	14×3	0.28	0.00	1.17	3.13	30.89	36.37	38.96	0.09	0.49	1.63
	14×4	0.33	0.85	2.31	4.69	16.57	17.47	18.01	0.00	0.96	2.08
Average		0.52	0.09	0.92	2.39	10.56	12.35	13.40	0.01	0.22	0.63
H_3	8×2	0.00	0.00	0.00	0.00	0.00	0.97	1.70	0.00	0.00	0.00
	8×3	0.00	0.00	0.00	0.00	0.00	0.37	0.93	0.00	0.00	0.00
	8×4	0.00	0.00	0.00	0.00	0.00	0.07	0.28	0.00	0.00	0.00
	10×2	0.00	0.00	0.42	0.79	7.86	10.43	11.33	0.00	0.00	0.00
	10×3	0.00	0.00	0.82	2.07	3.93	5.92	9.85	0.00	0.00	0.00
	10×4	0.00	0.00	0.08	0.25	1.54	2.08	2.77	0.00	0.00	0.00
	12×2	1.00	0.35	3.78	5.76	54.08	61.27	65.25	0.00	0.04	0.14
	12×3	0.20	0.20	3.61	8.81	21.39	26.33	29.44	0.00	0.00	0.00
	12×4	0.11	0.04	1.18	3.09	6.84	10.42	12.23	0.04	0.35	1.06
	14×2	0.61	2.69	5.18	8.25	38.65	46.73	52.13	0.00	1.48	3.14
	14×3	1.38	0.00	2.70	5.56	26.37	34.83	39.53	0.00	0.10	0.52
	14×4	0.29	0.67	1.98	4.04	18.38	23.82	27.01	0.00	0.60	0.95
Average		0.30	0.33	1.65	3.22	14.92	18.60	21.04	0.00	0.21	0.48

4.3. *Results and discussion*

In this section, the comparative evaluations of the proposed HDCS, the VNS, the DCS, and a commercial solver CPLEX® 12.5 based on the branch-and-cut algorithm are performed on the same benchmark of instances explained in Section 4.1. Due to the intractability of the studied problem, the CPLEX® 12.5 is only used to solve the MIP model with small instances. A 3600 sec time limit is imposed. When solved by CPLEX®, a particular run is simply terminated and the best current integer solution is returned if the optimal solution has not been found or verified in that amount of time.

Owing to the stochastic nature of the evaluated meta-heuristics, each algorithm is run *five times* for a given instance to reach reliable results. The computational results of the best (RPD_B), average (RPD_A) and worst (RPD_W) deviation of each combination of n (number of jobs) and m (number of machines) obtained by VNS, DCS and HDCS are summarized in Tables 3 and 4.

Table 4. Computation results of the algorithms (in seconds) for large instances.

Instance		VNS				DCS				HDCS			
		RPD_B	RPD_A	RPD_W	Time (s)	RPD_B	RPD_A	RPD_W	Time (s)	RPD_B	RPD_A	RPD_W	Time (s)
H_1	30×4	1.49	8.50	19.09	41.63	49.61	49.63	49.63	5.16	**0.90**	**4.65**	**8.50**	50.38
	30×6	**0.24**	6.98	15.99	39.69	43.71	43.99	44.06	5.27	1.57	**3.58**	**6.22**	51.10
	30×8	**0.89**	8.05	16.09	39.12	45.99	46.34	46.50	6.48	1.60	**4.06**	**6.58**	46.58
	40×4	**0.86**	10.31	24.61	95.31	65.89	67.90	68.64	7.40	1.98	**6.09**	**10.23**	112.20
	40×6	**1.43**	13.75	25.76	91.41	75.04	80.10	81.40	7.92	3.79	**9.16**	**14.80**	110.94
	40×8	2.48	10.42	19.29	78.66	50.46	51.35	51.57	7.32	**1.06**	**5.43**	**9.75**	102.04
	50×4	1.99	23.64	29.63	164.51	94.65	96.52	97.01	8.76	**1.42**	**7.40**	**15.21**	221.95
	50×6	**3.62**	19.39	42.70	155.02	53.94	58.40	66.08	8.68	3.75	**11.96**	**20.18**	217.16
	50×8	3.52	16.27	30.23	151.72	76.00	76.25	76.31	8.44	**1.79**	**9.22**	**16.07**	183.92
	60×4	6.85	18.67	30.55	349.18	71.67	76.47	79.71	10.48	**1.64**	**10.92**	**18.80**	382.93
	60×6	2.63	12.71	21.50	307.53	63.89	63.89	63.89	10.44	**2.53**	**7.67**	**13.38**	315.90
	60×8	3.05	11.21	19.78	316.57	57.48	57.48	57.48	10.20	**0.60**	**6.37**	**12.23**	337.98
Average		2.42	13.33	24.60	152.53	62.36	64.03	65.19	8.05	1.89	7.21	12.66	177.76
H_2	30×4	1.70	8.81	16.44	43.43	35.19	35.64	35.79	5.45	**0.96**	**5.20**	**9.70**	46.19
	30×6	0.94	6.12	13.54	41.03	25.84	27.36	28.15	5.44	**0.40**	**2.51**	**4.50**	45.30
	30×8	1.46	7.58	15.95	36.84	28.31	29.78	30.52	6.68	**0.60**	**2.39**	**3.88**	42.12
	40×4	**0.81**	10.33	22.68	97.64	38.18	38.32	38.37	7.00	2.49	**7.81**	**12.19**	104.59
	40×6	**0.87**	10.16	23.08	90.96	37.43	38.43	39.01	6.88	2.05	**5.67**	**10.00**	91.51
	40×8	**0.71**	8.66	22.33	82.68	49.37	50.00	50.15	6.64	1.66	**5.66**	**9.81**	98.96
	50×4	2.83	17.59	38.13	156.74	60.14	61.22	61.15	9.00	**1.34**	**7.95**	**14.98**	184.93
	50×6	**1.96**	11.16	21.57	168.50	26.82	26.82	26.82	8.60	1.97	**6.39**	**11.90**	179.63
	50×8	1.99	15.06	27.47	166.81	38.17	39.24	40.06	8.84	**1.96**	**7.77**	**13.31**	168.26
	60×4	3.04	10.77	18.16	310.91	32.29	32.29	32.29	10.44	**0.97**	**6.79**	**12.50**	316.43
	60×6	4.55	15.73	28.03	288.22	44.30	44.30	44.30	10.36	**1.07**	**6.62**	**11.41**	298.84
	60×8	3.81	8.96	19.10	262.49	20.31	20.31	20.31	10.24	**1.61**	**5.04**	**8.68**	270.85
Average		2.06	10.91	22.21	145.52	36.36	36.98	37.24	7.96	1.42	5.82	10.24	153.97
H_3	30×4	2.23	8.59	16.09	42.96	53.71	54.57	54.94	5.82	**0.25**	**3.50**	**5.76**	50.39
	30×6	**1.27**	9.44	17.66	41.98	51.20	52.43	53.04	6.12	2.08	**4.31**	**7.04**	46.47
	30×8	1.69	7.24	15.42	36.66	50.18	51.67	52.44	6.16	**0.35**	**1.86**	**3.15**	41.30
	40×4	1.92	12.24	22.62	97.18	57.73	57.95	58.01	6.96	**0.99**	**5.89**	**11.00**	115.97
	40×6	**0.92**	9.67	19.79	88.09	47.61	48.74	49.02	6.92	1.88	**5.67**	**10.13**	103.95
	40×8	**0.63**	8.63	18.36	84.11	66.75	67.53	67.75	6.72	1.20	**5.62**	**10.45**	98.52
	50×4	**3.12**	16.66	34.57	165.67	59.92	62.83	67.96	8.68	3.66	**10.39**	**17.57**	199.58
	50×6	**1.01**	10.53	22.87	152.81	45.53	45.72	45.77	8.52	1.68	**6.67**	**11.08**	189.81
	50×8	**0.72**	13.99	29.39	148.59	81.57	82.41	82.62	8.52	3.48	**10.43**	**18.53**	190.57
	60×4	4.31	12.80	20.14	312.29	69.43	69.52	69.54	10.48	**1.40**	**8.26**	**13.26**	335.21
	60×6	4.05	15.74	28.93	310.69	62.67	62.67	62.67	10.36	**1.83**	**7.78**	**12.62**	332.11
	60×8	5.32	17.55	30.32	268.94	73.55	73.55	73.55	10.20	**2.06**	**10.88**	**18.89**	299.49
Average		2.27	11.92	23.01	145.83	59.99	60.80	61.44	7.95	1.74	6.77	11.62	166.95

Recall that, for each combination of n and m in our experiment, 10 random replicates are generated and for each instance the algorithm is run five times. Thus, corresponding to each combination of n and m, each value of RPD_B and RPD_W is found in the following way: firstly the best (respectively, the worst) RPD value is found among the five runs of an algorithm applied to a particular instance, then the average value of the 10 best (respectively, the worst) RPD values of those 10 random instances is recorded as RPD_B (respectively, RPD_W) in Tables 3 and 4. Likewise, the value of RPD_A is obtained in the following way: firstly the average RPD value is found for the five runs of an algorithm applied to a particular instance, then the average value of those 10 average RPD values of the 10 random instances is recorded as RPD_A in Tables 3 and 4.

Table 3 reports the experimental results of the small instances. Column 1 shows the combination of n and m for each instance. Columns 2–5 report the computational results of the four methods. If CPLEX® can yield the optimal solution, the RPD value is calculated over the optimal total tardiness. The CPLEX® solver is able to deliver optimal solutions for all instances with

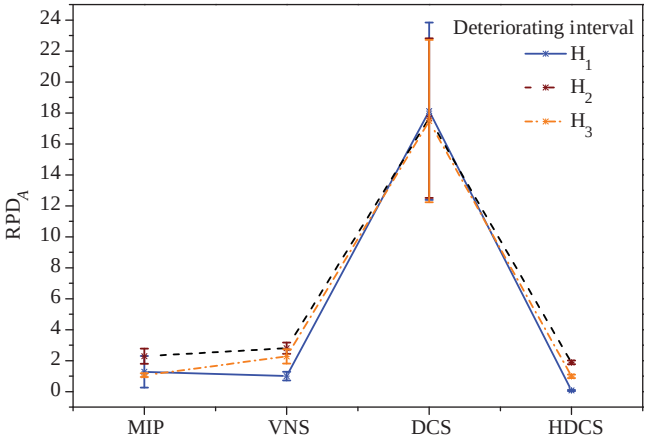

Figure 8. Average RPD_A means plot and the Tukey HSD intervals (at the 95% confidence level) for the tested algorithms (small instances).

Table 5. Computation times of the algorithms (in seconds) for small instances.

	H_1				H_2				H_3			
Instance	MIP	VNS	DCS	HDCS	MIP	VNS	DCS	HDCS	MIP	VNS	DCS	HDCS
8×2	5.22	1.06	0.52	1.56	30.91	1.08	0.55	1.56	16.12	1.09	0.63	1.55
8×3	2.01	0.99	0.58	1.50	1.17	1.00	0.51	1.49	2.83	0.99	0.56	1.46
8×4	0.33	0.96	0.49	1.38	0.20	0.93	0.49	1.37	0.31	0.95	0.41	1.38
10×2	$1{,}188.28^{1}$	1.94	0.83	2.62	$2{,}140.39^{4}$	1.95	0.62	2.60	$1{,}590.56^{2}$	1.97	0.67	2.72
10×3	209.54	1.80	0.81	2.60	423.81	1.77	0.64	2.46	97.56	1.75	0.74	2.44
10×4	14.91	1.70	0.78	2.28	6.29	1.62	0.59	2.25	48.63	1.70	0.71	2.32
12×2	$3{,}559.68^{9}$	3.00	1.01	4.34	$3{,}259.20^{9}$	2.80	0.70	4.15	$3{,}415.89^{9}$	3.02	0.80	4.58
12×3	$2{,}496.76^{6}$	2.87	0.95	4.02	$3{,}101.88^{8}$	2.93	0.75	4.20	$2{,}328.34^{5}$	2.86	0.83	4.00
12×4	$1{,}873.83^{5}$	2.78	0.89	3.91	$1{,}160.38^{3}$	2.76	0.72	3.61	$1{,}826.88^{4}$	2.69	0.80	3.80
14×2	$3{,}600.00^{10}$	4.73	1.01	6.74	$3{,}600.00^{10}$	4.53	0.90	6.82	$3{,}258.91^{9}$	4.81	1.00	6.59
14×3	$3{,}600.00^{10}$	4.59	0.85	6.88	$2{,}791.57^{7}$	4.62	0.82	6.61	$2{,}862.13^{7}$	4.41	0.91	6.28
14×4	$2{,}576.57^{6}$	4.10	0.96	5.69	$2{,}755.64^{7}$	4.24	0.95	6.24	$3{,}039.72^{7}$	4.26	0.86	6.23
Average	1,593.93	2.54	0.81	3.63	1,605.95	2.52	0.69	3.61	1,540.66	2.54	0.74	3.61

Note: Superior numbers indicate the number of instances out of 10 that cannot be solved optimally by CPLEX® within the 3600 sec time limit.

eight jobs. For the 10-job instances, CPLEX® is able to solve all the instances optimally with three and four machines, but only 23 of 30 instances with two machines. Regarding the 12- and 14-job cases, CPLEX® only gives optimal solutions for a small portion of instances. In total, CPLEX® can yield optimal solutions for 223 of the 360 small instances.

The results revealed by Table 3 are exciting. From Table 3, it is found that HDCS can deliver the best RPD_B values except for the instances 10×2 with H_1 and 14×3 with H_2. Even for these two instances, the RPD_B values given by HDCS are only slightly larger than the corresponding best values given by one of the other three algorithms. The VNS is efficient in solving a parallel machine flow time problem with step-deterioration (Cheng *et al.* 2012); however, it does not perform as well as HDCS for the PMSDST problem. This is because the consideration of setup times and the total tardiness criterion in the PMSDST problem makes the VNS more easily trapped into a local optimal solution.

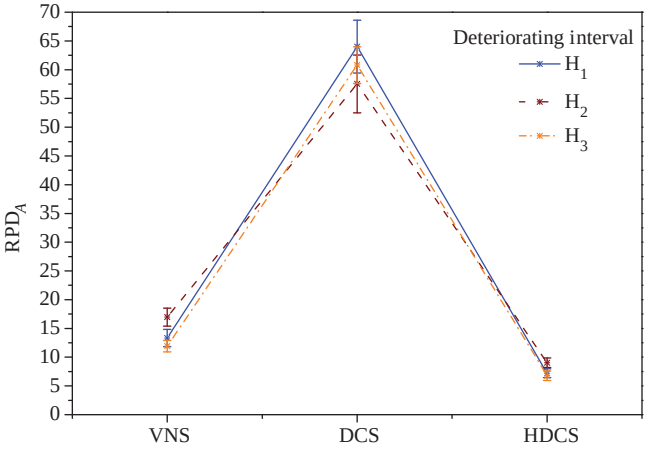

Figure 9. Average RPD_A means plot and the Tukey HSD intervals (at the 95% confidence level) for the tested algorithms (large instances).

In order to analyse the results further, the one-way analysis of variance (ANOVA) is adopted to check whether the observed difference in the RPD_A values for different algorithms are statistically significant when one of the deteriorating intervals (H_1, H_2 and H_3) is applied. The means plots along with the Tukey honestly significant difference (HSD) intervals at the 95% confidence level are given in Figure 8 for each algorithm applied to each of the three different deteriorating intervals. The means obtained by the four algorithms for the same deteriorating interval (one of H_1, H_2 and H_3) are connected by a line segment. If the Tukey HSD intervals of two algorithms for the same deteriorating interval have overlapping, then the performances of the two algorithms are not statistically significantly different. It is clearly seen from Figure 8 that the proposed HDCS is statistically better than other methods in that its Tukey HSD intervals of the RPD values are lower than those of other methods when applied to each of the three intervals H_1, H_2 and H_3.

The computation times of the four methods for each combination of n (number of jobs) and m (number of machines) are also presented in Table 5. They are obtained in a similar manner as for obtaining the RPD_A values. When the size of an instance increases, the computation time of CPLEX® grows significantly. Once the number of jobs exceeds 10, CPLEX® completely exhausts the given time limit for most instances. DCS consumes the least time compared with the other algorithms, but its performance is the worst of all the algorithms. On the other hand, the computation times of VNS and HDCS are very modest.

In order to evaluate the algorithms when the size of the problem is increased, experiments are also performed using large-sized instances. Since CPLEX® already consumes a lot of computational time in solving small-sized instances, it is excluded in this case. The results of large-sized instances obtained by all the meta-heuristic algorithms are listed in Table 4. In the table, the best RPD value for a specific instance delivered by an algorithm is in boldface type. It is found that the average results (RPD_A) of HDCS are *always* better than those of DCS and VNS. However, there are 15 instances in which the quality of the best solutions (RPD_B) obtained by HDCS is poorer than that of VNS. This might be due to the fact that more neighbourhood structures are used in VNS. Nevertheless, the worst solutions (RPD_W) delivered by HDCS are significantly better than those of the DCS and the VNS.

In order to validate the statistical significance of the observed difference in solution quality by different algorithms, one-way ANOVA is used again as in the previous scenario. Figure 9 shows the means plot with the Tukey HSD intervals at the 95% confidence level for each algorithm

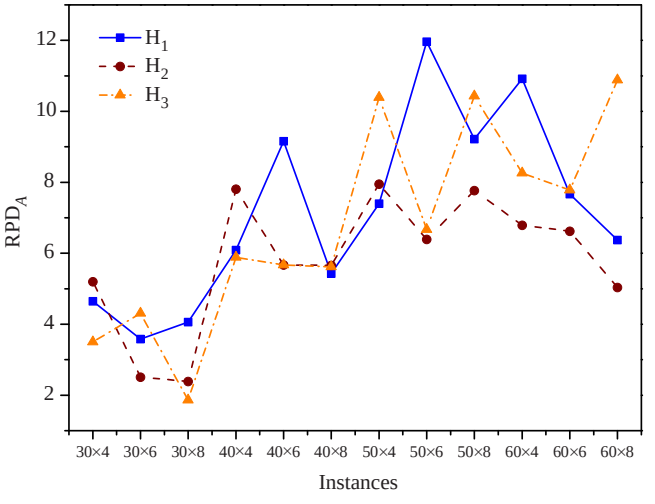

Figure 10. The effect of different deteriorating intervals on the performance of the HDCS algorithm.

under different deteriorating intervals. It is clearly seen that HDCS is statistically better than the other two algorithms, as HDCS consistently and statistically possesses lower Tukey intervals compared to the other two algorithms for each of the three deteriorating intervals H_1, H_2 and H_3.

In Table 4, from the rows labelled 'Average', it can be concluded that the average time taken by the DCS algorithm is the shortest, while the average time needed for the HDCS algorithm is the longest and the time consumed by VNS lies between that of the other two algorithms. This is due to the fact that more than one elite solution is being improved by the variable neighbourhood descent in HDCS. But the computational time of HDCS is acceptable for obtaining a good quality solution to the underlying problem.

Finally, the effect of deteriorating intervals on the performance of the proposed algorithm is analysed. As shown in Figure 10, the performance of HDCS somewhat depends on the deteriorating interval, but it is not too significant. When the deteriorating dates are generated from H_3, the mean RPD values delivered by HDCS are the smallest.

5. Conclusions

In this article, an identical parallel machines scheduling problem with step-deteriorating jobs and sequence-dependent setup times was considered. The objective of this problem was to find a schedule for minimizing the total tardiness. To solve this problem, a mixed-integer programming model was presented to deliver exact solutions for relatively small-sized instances. Due to the intractability of the problem, a hybrid discrete cuckoo search algorithm was proposed to obtain a near-optimal solution. In the proposed algorithm, the operators of the standard CS algorithm are redefined by some discrete operations. To improve the quality of the solution, variable neighbourhood descent was used as a local search approach to refine a set of chosen elite solutions. Moreover, the MBHG heuristic was incorporated into the generation of initial solutions. A set of small to large instances were produced to test the performance of the proposed algorithm. The computational results show that the hybrid discrete cuckoo search algorithm in general provides good average results for test problems, especially in large-sized instances. Indeed, the HDCS algorithm outperforms the CPLEX® solver, and the VNS and DCS algorithms in terms of average RPD values, shown in the rows labelled 'Average' in Tables 3 and 4. Further studies may

focus on applying the proposed algorithm to solve other scheduling problems, such as unrelated parallel machine scheduling problems, the flowshop problem, and their stochastic versions. In addition, some multi-objective scheduling problems may be considered in future work.

Funding

This work was supported by the National Natural Science Foundation of China [grant number 51405403], [grant number 51175442], [grant number 51205328]; the Fundamental Research Funds for the Central Universities [grant number 2682014BR019].

References

Alidaee, B., and N. K. Womer. 1999. "Scheduling with Time Dependent Processing Times: Review and Extensions." *Journal of the Operational Research Society* 50 (7): 711–720.

Allahverdi, A., J. N. D. Gupta, and T. Aldowaisan. 1999. "A Review of Scheduling Research Involving Setup Considerations." *Omega* 27 (2): 219–239.

Allahverdi, A., C. T. Ng, T. C. E. Cheng, and M. Y. Kovalyov. 2008. "A Survey of Scheduling Problems with Setup Times or Costs." *European Journal of Operational Research* 187 (3): 985–1032.

Armentano, V. A., and M. F. De Franca Filho. 2007. "Minimizing Total Tardiness in Parallel Machine Scheduling with Setup Times: An Adaptive Memory-Based GRASP Approach." *European Journal of Operational Research* 183 (1): 100–114.

Bahalke, U., A. M. Yolmeh, and K. Shahanaghi. 2010. "Meta-Heuristics to Solve Single-Machine Scheduling Problem with Sequence-Dependent Setup Time and Deteriorating Jobs." *International Journal of Advanced Manufacturing Technology* 50 (5-8): 749–759.

Bank, M., S. M. T. Fatemi Ghomi, F. Jolai, and J. Behnamian. 2012. "Application of Particle Swarm Optimization and Simulated Annealing Algorithms in Flow Shop Scheduling Problem under Linear Deterioration." *Advances in Engineering Software* 47 (1): 1–6.

Biskup, D., J. Herrmann, and J. N. D. Gupta. 2008. "Scheduling Identical Parallel Machines to Minimize Total Tardiness." *International Journal of Production Economics* 115 (1): 134–142.

Browne, S., and U. Yechiali. 1990. "Scheduling Deteriorating Jobs on a Single Processor." *Operations Research* 38 (3): 495–498.

Burnwal, S., and S. Deb. 2013. "Scheduling Optimization of Flexible Manufacturing System Using Cuckoo Search-Based Approach." *International Journal of Advanced Manufacturing Technology* 64 (5-8): 951–959.

Cheng, T. C. E., and Q. Ding. 2001. "Single Machine Scheduling with Step-Deteriorating Processing Times." *European Journal of Operational Research* 134 (3): 623–630.

Cheng, T. C. E., Q. Ding, M. Y. Kovalyov, A. Bachman, and A. Janiak. 2003. "Scheduling Jobs with Piecewise Linear Decreasing Processing Times." *Naval Research Logistics* 50 (6): 531–554.

Cheng, T. C. E., Q. Ding, and B. M. T. Lin. 2004. "A Concise Survey of Scheduling with Time-Dependent Processing Times." *European Journal of Operational Research* 152 (1): 1–13.

Cheng, W. M., P. Guo, Z. Q. Zhang, M. Zeng, and J. Liang. 2012. "Variable Neighborhood Search for Parallel Machines Scheduling Problem with Step Deteriorating Jobs." *Mathematical Problems in Engineering*, Article ID 928312. http://dx.doi.org/10.1155/2012/928312.

Cheng, T. C. E., C. J. Hsu, Y. C. Huang, and W. C. Lee. 2011. "Single-Machine Scheduling with Deteriorating Jobs and Setup Times to Minimize the Maximum Tardiness." *Computers & Operations Research* 38 (12): 1760–1765.

Cheng, T. C. E., W. C. Lee, and C. C. Wu. 2011. "Single-Machine Scheduling with Deteriorating Jobs and Past-Sequence-Dependent Setup Times." *Applied Mathematical Modelling* 35 (4): 1861–1867.

Chung, Tsui-Ping, Ching-Jong Liao, and Milton Smith. 2014. "A Canned Food Scheduling Problem with Batch Due Date." *Engineering Optimization* 46 (9): 1284–1294.

Civicioglu, Pinar. 2012. "Transforming Geocentric Cartesian Coordinates to Geodetic Coordinates by Using Differential Search Algorithm." *Computers & Geosciences* 46 (Sep.): 229–247.

Civicioglu, Pinar. 2013. "Backtracking Search Optimization Algorithm for Numerical Optimization Problems." *Applied Mathematics and Computation* 219 (15): 8121–8144.

Civicioglu, P., and E. Besdok. 2013. "A Conceptual Comparison of the Cuckoo-Search, Particle Swarm Optimization, Differential Evolution and Artificial Bee Colony Algorithms." *Artificial Intelligence Review* 39 (4): 315–346.

Davis, L. 1985. "Applying Adaptive Algorithms to Epistatic Domains." In *Proceedings of the 9th International Joint Conference on Artificial Intelligence*, Los Angeles, CA, 162–164. San Francisco, CA: Morgan Kaufmann.

Farahani, M. H., and L. Hosseini. 2013. "Minimizing Cycle Time in Single Machine Scheduling with Start Time-Dependent Processing Times." *International Journal of Advanced Manufacturing Technology* 64 (9-12): 1479–1486.

Gandomi, A. H., X. S. Yang, and A. H. Alavi. 2013. "Cuckoo Search Algorithm: A Metaheuristic Approach to Solve Structural Optimization Problems." *Engineering with Computers* 29 (1): 17–35.

Gawiejnowicz, S. 2008. *Time-Dependent Scheduling*. Berlin: Springer.

Guo, P., W. M. Cheng, and Y. Wang. 2014. “A General Variable Neighborhood Search for Single-Machine Total Tardiness Scheduling Problem with Step-Deteriorating Jobs.” *Journal of Industrial and Management Optimization* 10 (4): 1071–1090.
Gupta, J. N. D., and S. K. Gupta. 1988. “Single Facility Scheduling with Nonlinear Processing Times.” *Computers & Industrial Engineering* 14 (4): 387–393.
He, C. C., C. C. Wu, and W. C. Lee. 2009. “Branch-and-Bound and Weight-Combination Search Algorithms for the Total Completion Time Problem with Step-Deteriorating Jobs.” *Journal of the Operational Research Society* 60 (12): 1759–1766.
Huang, X., and M. Z. Wang. 2011. “Parallel Identical Machines Scheduling with Deteriorating Jobs and Total Absolute Differences Penalties.” *Applied Mathematical Modelling* 35 (3): 1349–1353.
Jafari, A., and G. Moslehi. 2012. “Scheduling Linear Deteriorating Jobs to Minimize the Number of Tardy Jobs.” *Journal of Global Optimization* 54 (2): 389–404.
Jeng, A. A. K., and B. M. T. Lin. 2004. “Makespan Minimization in Single-Machine Scheduling with Step-Deterioration of Processing Times.” *Journal of the Operational Research Society* 55 (3): 247–256.
Jeng, A. A. K., and B. M. T. Lin. 2005. “Minimizing the Total Completion Time in Single-Machine Scheduling with Step-Deteriorating Jobs.” *Computers & Operations Research* 32 (3): 521–536.
Ji, M., and T. C. E. Cheng. 2007. “An FPTAS for Scheduling Jobs with Piecewise Linear Decreasing Processing Times to Minimize Makespan.” *Information Processing Letters* 102 (2-3): 41–47.
Joo, Cheol Min, and Byung Soo Kim. 2012. “Parallel Machine Scheduling Problem with Ready Times, Due Times and Sequence-Dependent Setup Times Using Meta-Heuristic Algorithms.” *Engineering Optimization* 44 (9): 1021–1034.
Kanagaraj, G., S. G. Ponnambalam, N. Jawahar, and J. Mukund Nilakantan. 2013. “An Effective Hybrid Cuckoo Search and Genetic Algorithm for Constrained Engineering Design Optimization.” *Engineering Optimization* 46 (10): 1331–1351. http://dx.doi.org/10.1080/0305215X.2013.836640.
Kaplan, Sezgin, and Ghaith Rabadi. 2013. “Simulated Annealing and Metaheuristic for Randomized Priority Search Algorithms for the Aerial Refuelling Parallel Machine Scheduling Problem with Due Date-to-Deadline Windows and Release Times.” *Engineering Optimization* 45 (1): 67–87.
Kubiak, W., and S. Van de Velde. 1998. “Scheduling Deteriorating Jobs to Minimize Makespan.” *Naval Research Logistics* 45 (5): 511–523.
Kunnathur, A. S., and S. K. Gupta. 1990. “Minimizing the Makespan with Late Start Penalties Added to Processing Times in a Single Facility Scheduling Problem.” *European Journal of Operational Research* 47 (1): 56–64.
Lai, P. J., W. C. Lee, and H. H. Chen. 2011. “Scheduling with Deteriorating Jobs and Past-Sequence-Dependent Setup Times.” *International Journal of Advanced Manufacturing Technology* 54 (5-8): 737–741.
Layegh, J., F. Jolai, and M. S. Amalnik. 2009. “A Memetic Algorithm for Minimizing the Total Weighted Completion Time on a Single Machine Under Step-Deterioration.” *Advances in Engineering Software* 40 (10): 1074–1077.
Lee, W. C., J. B. Lin, and Y. R. Shiau. 2011. “Deteriorating Job Scheduling to Minimize the Number of Late Jobs with Setup Times.” *Computers & Industrial Engineering* 61 (3): 782–787.
Lee, Hsin-Tao, Dar-Li Yang, and Suh-Jenq Yang. 2013. “Multi-Machine Scheduling with Deterioration Effects and Maintenance Activities for Minimizing the Total Earliness and Tardiness Costs.” *International Journal of Advanced Manufacturing Technology* 66 (1-4): 547–554.
Leung, J. Y. T., C. T. Ng, and T. C. E. Cheng. 2008. “Minimizing Sum of Completion Times for Batch Scheduling of Jobs with Deteriorating Processing Times.” *European Journal of Operational Research* 187 (3): 1090–1099.
Li, X. T., and M. H. Yin. 2013. “A Hybrid Cuckoo Search via Lévy Flights for the Permutation Flow Shop Scheduling Problem.” *International Journal of Production Research* 51 (16): 4732–4754.
Mor, B., and G. Mosheiov. 2012. “Batch Scheduling with Step-Deteriorating Processing Times to Minimize Flowtime.” *Naval Research Logistics* 59 (8): 587–600.
Mosheiov, G. 1995. “Scheduling Jobs with Step-Deterioration; Minimizing Makespan on a Single- and Multi-Machine.” *Computers & Industrial Engineering* 28 (4): 869–879.
Moslehi, G., and A. Jafari. 2010. “Minimizing the Number of Tardy Jobs under Piecewise-Linear Deterioration.” *Computers & Industrial Engineering* 59 (4): 573–584.
Ouaarab, A., B. Ahiod, and X. S. Yang. Online First 2013. “Discrete Cuckoo Search Algorithm for the Travelling Salesman Problem.” *Neural Computing and Applications* 24 (7–8): 1659–1669.
Radhakrishnan, S., and J. A. Ventura. 2000. “Simulated Annealing for Parallel Machine Scheduling with Earliness–Tardiness Penalties and Sequence-Dependent Set-Up Times.” *International Journal of Production Research* 38 (10): 2233–2252.
Ruiz, R., C. Maroto, and J. Alcaraz. 2006. “Two New Robust Genetic Algorithms for the Flowshop Scheduling Problem.” *Omega* 34 (5): 461–476.
Sundararaghavan, P. S., and A. S. Kunnathur. 1994. “Single Machine Scheduling with Start Time Dependent Processing Times: Some Solvable Cases.” *European Journal of Operational Research* 78 (3): 394–403.
Toksari, M. D., and E. Guener. 2008. “Minimizing the Earliness/Tardiness Costs on Parallel Machine with Learning Effects and Deteriorating Jobs: A Mixed Nonlinear Integer Programming Approach.” *International Journal of Advanced Manufacturing Technology* 38 (7-8): 801–808.
Valian, E., S. Tavakoli, S. Mohanna, and A. Haghi. 2013. “Improved Cuckoo Search for Reliability Optimization Problems.” *Computers & Industrial Engineering* 64 (1): 459–468.
Valian, Ehsan, and Elham Valian. 2013. “A Cuckoo Search Algorithm by Lévy Flights for Solving Reliability Redundancy Allocation Problems.” *Engineering Optimization* 45 (11): 1273–1286.

Walton, S., O. Hassan, K. Morgan, and M. R. Brown. 2011. "Modified Cuckoo Search: A New Gradient Free Optimisation Algorithm." *Chaos, Solitons & Fractals* 44 (9): 710–718.

Wang, J. B., Y. Jiang, and G. Wang. 2009. "Single-Machine Scheduling with Past-Sequence-Dependent Setup Times and Effects of Deterioration and Learning." *International Journal of Advanced Manufacturing Technology* 41 (11-12): 1221–1226.

Wu, C. C., Y. R. Shiau, L. H. Lee, and W. C. Lee. 2009. "Scheduling Deteriorating Jobs to Minimize the Makespan on a Single Machine." *International Journal of Advanced Manufacturing Technology* 44 (11-12): 1230–1236.

Yang, Suh-Jenq. 2013. "Unrelated Parallel-Machine Scheduling with Deterioration Effects and Deteriorating Multi-Maintenance Activities for Minimizing the Total Completion Time." *Applied Mathematical Modelling* 37 (5): 2995–3005.

Yang, X. S., and S. Deb. 2009. "Cuckoo Search via Lévy Flights." In *Proceedings of the World Congress on Nature and Biologically Inspired Computing*, 210–214. IEEE Coimbatore, India: IEEE Publications.

Yang, X. S., and S. Deb. 2010. "Engineering Optimisation by Cuckoo Search." *International Journal of Mathematical Modelling and Numerical Optimisation* 1 (4): 330–343.

Yildiz, A. R. 2013. "Cuckoo Search Algorithm for the Selection of Optimal Machining Parameters in Milling Operations." *International Journal of Advanced Manufacturing Technology* 64 (1-4): 55–61.

Yin, Y. Q., T. C. E. Cheng, J. Y. Xu, S. R. Cheng, and C. C. Wu. 2013. "Single-Machine Scheduling with Past-Sequence-Dependent Delivery Times and a Linear Deterioration." *Journal of Industrial and Management Optimization* 9 (2): 323–339.

Zhao, C. L., and H. Y. Tang. 2010. "Single Machine Scheduling with Past-Sequence-Dependent Setup Times and Deteriorating Jobs." *Computers & Industrial Engineering* 59 (4): 663–666.