Computers & Industrial Engineering 126 (2018) 348–360

Contents lists available at ScienceDirect

Computers & Industrial Engineering

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

An improved cuckoo search algorithm for scheduling jobs on identical parallel machines

Dipak Laha[a], Jatinder N.D. Gupta[b,*]

[a] Department of Mechanical Engineering, Jadavpur University, Kolkata 700032, India
[b] College of Business, University of Alabama in Huntsville, Huntsville, AL, USA

ARTICLE INFO

Keywords:
Scheduling identical parallel machines
Makespan
Improved cuckoo search algorithm
Levy flights
Computational comparisons

ABSTRACT

In this paper, we propose an improved cuckoo search algorithm (ICSA) to minimize makespan for the identical parallel-machine scheduling problem. Starting with an initial population of schedules generated by using the longest processing time (LPT) rule and the job-interchange mechanism, we select the best schedule from this population and then execute the proposed ICSA. For the ICSA, we first propose a heuristic approach using a modulus operator to transform a continuous position in CSA into discrete schedule of jobs for generating a new cuckoo by Levy flights. Next, we present a heuristic procedure based on the pairwise exchange neighborhood to produce smart cuckoos in the proposed ICSA. We then conduct exhaustive computational experimentation on a large number of randomly generated well-known benchmark problems to show that the proposed ICSA produces better solutions than the six state-of-the-art existing algorithms.

1. Introduction

Parallel-machine scheduling (PMS) problems have long drawn the attention of researchers and practitioners since the significant work of McNaughton (1959). PMS research is motivated by its combinatorial complexity as well as its immense practical applications in many industries and real-life environment such as production lines, semiconductor manufacturing, shipping docks, petrol station, university, hospitals, and computer systems (Cheng & Sin, 1990; Su & Lien, 2009). For example, manufacturing sectors, especially involving job shop production systems, use multiple machines of each type to increase the routing flexibility of jobs as a way to decrease the throughput time of a batch of jobs by reducing the possibility of production holdups due to any bottlenecks.

In this paper, we consider the scheduling problem involving a set of n independent jobs for processing through a set of m identical parallel machines to minimize the maximum completion time of the latest completed job in the schedule, i.e., the makespan of the schedule of jobs. Following the standard three-field representation of scheduling problems, we represent this identical parallel-machine scheduling problem to minimize makespan as a $P||C_{max}$ problem where P represents the identical parallel machines and C_{max} is the makespan or maximum completion time of all n jobs. Cheng and Sin (1990) and Mokotoff (2001) provide a detailed survey of various optimization methods and applications of this scheduling problem.

Several studies develop various lower and upper bound strategies and exact optimization algorithms for the $P||C_{max}$ problem (Dell'Amico, Iori, Martello, & Monaci, 2008; Dell'Amico & Martello, 1995, 2005; Mokotoff, 2004). Dell'Amico and Martello (1995) presented an exact branch-and-bound algorithm for the $P||C_{max}$ problem. Results of computational experiments reveal that their algorithm produces optimal solutions in many instances of the large sized problems from a large set of benchmark instances. Dell'Amico et al. (2008) proposed a branch-and-price algorithm and a metaheuristic based on the scatter search procedure. They showed the effectiveness of their algorithms for optimally solving the existing test problem instances of the $P||C_{max}$ problem. Mokotoff (2004) described a simple integer programming formulation of the $P||C_{max}$ problem and proposed an exact cutting plane algorithm based on adding valid inequalities to the linear programming model and the solution of its relaxation model. The computational results for solving the small and large problem instances from the literature show that the cutting plane algorithm is superior to the branch-and-bound technique with respect to number of iterations and computational effort. However, Dell'Amico and Martello (2005) reported that their previous algorithm published in 1995 outperforms the cutting plane algorithm by Mokotoff (2004) and requires significantly less computational effort.

Shim and Kim (2008) developed a branch and bound algorithm to

* Corresponding author.
E-mail address: guptaj@uah.edu (J.N.D. Gupta).

https://doi.org/10.1016/j.cie.2018.09.016
Received 29 October 2017; Received in revised form 3 September 2018; Accepted 5 September 2018
Available online 08 September 2018
0360-8352/

solve the $P||C_{max}$ problem. Their proposed algorithm included new dominance properties and improved lower bounds. Haouari and Jemmali (2008) embedded some newly derived bounds in a branch-and-bound procedure to show that their exact algorithm is very effective for solving the identical parallel-machine scheduling problem. Currently, many researchers are focusing on using the decomposition-based method to solve large-scale PMS problems optimally. For instance, Tran, Araujo, and Beck (2016) applied Benders decomposition method for solving the PMS problem with setups. Chen and Powell (1999) solved PMS problems by column generation.

Since the $P||C_{max}$ problem is known to be NP-hard in the strong sense (Garey & Johnson, 1979), it is computationally infeasible to solve the $P||C_{max}$ problem using an exact optimization algorithm or through the execution of exhaustive enumeration of all possible schedules, especially for large-sized problems. Therefore, due to their polynomial time complexity, heuristic and metaheuristic-based approximation algorithms are normally preferred to generate optimal or near-optimal schedules, especially, for scheduling problems with large number of jobs.

In this paper, we present an improved cuckoo search algorithm (ICSA) to solve the $P||C_{max}$ problem. We propose a modulus operator based heuristic approach to convert a continuous position in CSA into a discrete schedule for obtaining a cuckoo by Levy flights. Next, based on a pairwise exchange neighborhood, we present a new construction heuristic procedure to generate smart cuckoos in the proposed method. Then we compare the proposed ICSA algorithm with six state-of-the-art procedures: simulated annealing (SA) of (Lee, Wu, & Chen, 2006), two variants of particle swam optimization (PSO) (Kashan & Karimi, 2009), discrete hybrid search (DHS) and hybrid version of DHS combined with local search (DHS_LS) (Chen, Pan, Wang, & Li, 2011) and CSA (Laha & Behera, 2015). The computational results with a large number of randomly generated well-known benchmark problem instances reveal that the proposed method produces better solutions compared to the existing algorithms.

The structure of the rest of this paper is as follows: Section 2 provides a brief review of various available heuristics for solving the $P||C_{max}$ problem. Section 3 formulates the $P||C_{max}$ problem and describes the lower bounds on its makespan. Section 4 presents a brief discussion of the cuckoo search algorithm. Section 5 describes the details of the proposed algorithm. Section 6 reports the results of exhaustive computational experimentation to test the effectiveness of the proposed ICSA. Finally, Section 7 summarizes our findings and suggests some fruitful directions for future research.

2. Literature review

Several existing construction-based heuristics can solve the $P||C_{max}$ problem. Some noteworthy construction heuristics include LPT (Graham, 1969), MULTIFIT (Coffman, Garey, & Johnson, 1978), COMBINE (Lee et al., 2006), pairwise interchange (Ghomi & Ghazvini, 1998) and LISTFIT (Gupta & Ruiz-Torres, 2001). Through empirical experimentation, Gupta and Ruiz-Torres (2001) show that LISTFIT outperforms the LPT, MULTIFIT, and COMBINE heuristics at the expense of its higher computational complexity. Ghomi and Ghazvini (1998) propose a pairwise interchange algorithm for this problem and show that for small-sized problems with uniform processing time distribution, their proposed algorithm outperforms the LPT and MULTIFIT algorithms.

Similar to the construction-based algorithms, several metaheuristics exist to minimize makespan on identical parallel machines. Some notable metaheuristics applied to this scheduling problem include the genetic algorithm (GA) (Min & Cheng, 1999), SA (Lee et al., 2006), DHS algorithm (Chen et al., 2011) and PSO (Kashan & Karimi, 2009). Lee et al. (2006) propose a SA heuristic considering the initial solution using LPT algorithm to solve the $P||C_{max}$ problem. Through an exhaustive computational experimentation based on standard scheduling benchmark problem sets, they show that SA significantly outperforms the existing effective construction methods. Kashan and Karimi (2009) present a discrete PSO (DPSO), and a hybrid version of PSO, termed HDPSO to solve identical PMS problem. Exhaustive computational results show the competitive performance of the DPSO over the SA of Lee et al. (2006) and the hybrid HDPSO algorithm outperforms both SA and DPSO. However, in terms of CPU times, SA is faster than both DPSO and HDPSO. Chen et al. (2011) propose a DHS algorithm to solve the $P||C_{max}$ problem. They also present a hybrid DHS_LS combining a local search strategy with the DHS. Simulation results reveal that the DHS_LS is very competitive in comparison with the SA (Lee et al., 2006), DPSO and HDPSO (Kashan & Karimi, 2009), with respect to both solution quality and execution times.

Yang and Deb (2009) propose the cuckoo search algorithm (CSA) as a structured randomized search optimization algorithm motivated by the combination of the holoparasite characteristics and Levy flight foraging configurations of some cuckoo species. The cuckoo search algorithm is enriched by the global random walks governed by the Levy flights distribution instead of standard isotropic random walks. Furthermore, since the Levy distribution following a power law in terms of step-length with a heavy-tailed probability distribution has an infinite mean and variance, the proposed algorithm can explore search space more efficiently and provides an opportunity to converge to global optima than other algorithms by standard Gaussian process (Yang & Deb, 2014). It is successful in solving various engineering and management problems including scheduling (Burnwal & Deb, 2013; Chandrasekharan & Simon, 2012; Guo, Cheng, & Wang, 2015; Laha & Behera, 2015, 2017; Li & Yin, 2013; Marichelvam, Prabaharan, & Yang, 2014a; Majumder & Laha, 2016), reliability optimization (Valian, Tavakoli, Mohanna, & Haghi, 2013; Valian & Valian, 2013), video target tracking (Walia & Kapoor, 2014), and travelling salesman routing (Ouaarab, Ahiod, & Yang, 2014). Abdel-Basset, Hessin, and Abdel-Fatah (2018), Expósito-Izquierdo and Expósito-Márquez (2017), Yang and Deb (2014) and Mohamad, Zain, and Bazin (2014) review the fundamental ideas and latest developments of cuckoo search as well as its applications.

In the field of scheduling, Marichelvam et al. (2014a) propose a CSA to minimize makespan in a hybrid flow shop. Li and Yin (2013) propose a cuckoo search-based memetic algorithm to solve permutation flow shop scheduling problem. They present a largest-rank-value-based rule to use Levy flights for obtaining a discrete job permutation. Marichelvam, Prabaharan, and Yang (2014b), Dasgupta and Das (2015), Komaki, Teymourinan, Kayvanfar, and Booyavi (2017) and Wang et al. (2017) discuss several modifications and extensions of cuckoo search for solving a variety of flowshop problems. Chandrasekharan and Simon (2012) develop a hybrid CSA integrated with fuzzy system to find optimal or near optimal schedules for the multi-objective unit commitment problem. They considered three objectives simultaneously, namely, fuel cost, emission, and reliability level of the problem. By applying the Levy flights for a discrete solution of the scheduling problem, Burnwal and Deb (2013) implement a CSA for scheduling a flexible manufacturing system to minimize both penalty cost caused by a delay in manufacturing and total elapsed time of a batch of jobs.

3. Problem definition and lower bounds

In the identical parallel-machine scheduling problem considered in this paper, a given set $N = \{1, 2, ..., n\}$ of n independent jobs available at time zero is to be assigned on a set $M = \{1, 2, ..., m\}$ of m identical parallel machines. The processing time (including the setup times required) for each job $i \in N$ is p_i. We assume that a machine processes one job at a time and once a job starts processing on either of the m machines, it must be completed without preemption. Let S_j be the subset of jobs assigned to machine M_j with its completion time $C(S_j) = \sum_{i \in S_j} p_i$. The makespan or the maximum completion time of a

schedule $S = (S_1, S_2, S_3, \ldots, S_m)$ of n jobs is given by:

$$MS = C_{\max}(S) = \max_{1 \leqslant j \leqslant m} C(S_j) \tag{1}$$

The problem considered in this paper is to determine m subsets of the given n jobs such that the MS given in Eq. (1) is minimum. By considering a preemptive version of the $P||C_{\max}$ problem, we can find an assignment of jobs to machines such that the completion time of all the jobs assigned to each machine is the same. Since the makespan of a non-preemptive schedule cannot be less than a preemptive schedule, a lower bound LB on MS is obtained as follows.

$$LB = \max\left\{\max_{1 \leqslant i \leqslant n} p_i ; \sum_{i=1}^{n} p_i/m\right\} \tag{2}$$

In the present study, we consider the assumption of non-preemptive schedule and the corresponding lower bound $LB1$ on the optimal value of MS, used in the identical parallel-machine scheduling literature is given as:

$$LB1 = \max\left\{\max_{1 \leqslant i \leqslant n} p_i ; \sum_{i=1}^{n} p_i/m\right\} \tag{3}$$

A better lower bound $LB2$ with respect to the optimal MS value given by Mokotoff (2001) can be determined from the following expression.

$$LB2 = \max\left\{\max_{1 \leqslant i \leqslant n} p_i ; \sum_{i=1}^{n} p_i/m; p_m + p_{m+1}\right\} \tag{4}$$

where, $p_1 \geqslant p_2 \geqslant \cdots \geqslant p_n$.

Total number of feasible solutions (N_2) for n jobs, 2 machines in the identical parallel-machine problem can be computed as:

$$N_2 = \sum_{i=1}^{n-1} C(n, i) = 2^n - 2 \tag{5}$$

where, $C(n, i) = \left(\frac{n!}{i!\,(n-i)!}\right)$.

Similarly, for n jobs, 3 machines and for n jobs, 4 machines problems, N_3 and N_4 can be computed as:

$$N_3 = \sum_{k=1}^{n-2} C(n, k) \sum_{p=1}^{n-k-1} C(n-k, p) \tag{6}$$

$$N_4 = \sum_{k=1}^{n-3} C(n, k) \left(\sum_{p=1}^{n-k-2} C(n-k, p) \sum_{r=1}^{r-k-p-1} C(n-k-p, r) \right) \tag{7}$$

Therefore, total number of feasible solutions (N_m) for n jobs, m machines in the identical parallel-machine problem is given by:

$$N_m = \sum_{k=1}^{n-(m-1)} C(n, k) \left(\sum_{p=1}^{n-k-(m-2)} C(n-k, p) \right) \ldots \left(\sum_{t=1}^{n-k-p-\ldots-(-1)} C(n-k-p-\ldots, t) \right) = O(2^n) \tag{8}$$

For example, using Eq. (5), for $n = 10$, $m = 2$, $N_2 = 1022$, and similarly, using Eq. (6), for $n = 6$, $m = 3$, N_3 will be 540. The MS of a schedule can be determined in $O(n)$ computational effort using Eq. (1). Therefore, the overall asymptotic time complexity of solving $P||C_{\max}$ problem through complete enumeration is $O(n*2^n)$ or $O(2^n)$, which is exponential and hence not practical to solve large-sized problems.

4. The cuckoo search algorithm

In this section, we briefly discuss the cuckoo search algorithm including the cuckoo behavior, Levy flights, and finally, the basic cuckoo search procedure.

4.1. Cuckoo behavior

Some cuckoo species rely on some host species to lay eggs in their host nests and grow their young cuckoos due to their holoparasite characteristics. The host species hatch the eggs of the cuckoos as their own eggs and brood the young chicks of the cuckoos. However, not all the eggs of the cuckoos in the host nests are brooded by the host species. The host species discard some eggs (Payne, Sorenson, & Klitz, 2005). Accordingly, the cuckoo species adopt some strategies to exploit the host birds by imitating the color and patterns of eggs in some chosen nests.

4.2. Levy flights

Apart from the holoparasite characteristics of the cuckoo species, the CSA developed by Yang and Deb (2009) depends on the foraging nature of some cuckoo species. The foraging pattern of the cuckoos is governed by an important factor, known as Levy flights (Brown, Liebovitch, & Glendon, 2007) that models a random walk in which the step-lengths follow a heavy-tailed probability distribution. Therefore, the foraging path used by some animals and insects follows a structured random walk with distributed step lengths of a heavy-tailed probability distribution characterized by Levy flights. Yang and Deb (2014) demonstrate that CSA with Levy flights based structured random walk instead of a pure random walk is more effective than many existing metaheuristics such as PSO, bee colony optimization, and differential evolution.

4.3. Cuckoo search procedure

In CSA, an egg in a nest is a solution and a cuckoo egg represents a new solution. A nest with the better quality of eggs chosen for new generation corresponds to the ability of the egg to give a new cuckoo. The CSA is constructed based on the following assumptions.

- Each cuckoo selects a host nest and lays an egg randomly.
- The best nest with the highest quality egg is chosen for the next generation.
- The number of nests is kept constant and a host bird can abandon a foreign egg with some probability $P_a \in [0, 1]$.

The new solution (nest or egg) $x_i(t + 1)$ at $(t + 1)$-th generation for cuckoo i using Levy flights is given as:

$$x_i(t + 1) = x_i(t) + \alpha Levy(\lambda) \tag{9}$$

where $\alpha > 0$ represents the step size scaling factor and depends on the scale of the problem of interest. In many instances, $\alpha = O(1)$ can be used, i.e., the value α is constant and it will have fixed, finite time complexity. The cuckoo i randomly selects the host nest to lay an egg using Levy flights based random walk. The levy flight in which the jumps or steps of a cuckoo is governed according to the following probability distribution:

$$Levy \sim u = t^{-\lambda},\ 1 < \lambda \leqslant 3 \tag{10}$$

The above Levy distribution follows a power law in terms of step-length with a heavy-tailed probability distribution. It has an infinite variance and infinite mean. The step-length λ splits the search domain into local and global search spaces and provides an opportunity for global search instead of getting stuck at a local minima. Since the cuckoo search uses global random walk through Levy flights, it is more efficient in exploring the search space to converge to global optimality (Yang & Deb, 2014).

Let $f(x)$, $x = (x_1, \ldots, x_d)^T$ be the objective function to be maximized. Then, the steps of the CSA given by Yang & Deb (2010) can be summarized as follows:

Algorithm 1. Cuckoo Search Algorithm

Input: A set $N = \{1, 2, ..., n\}$ of n jobs, a set $M = \{1, 2, ..., m\}$ of m identical parallel machines, processing time p_i for each job $i \in N$, MaxGeneration, and stopping criterion.
Generate an initial population of R host nests x_i, $(i = 1, ..., R)$.
while $t <$ MaxGeneration or stopping criterion **do**
Obtain a cuckoo randomly by Levy flights.
Evaluate its quality/fitness F_i.
Choose a nest (say, j) randomly from the population of R and compute its quality/fitness as F_j.
if $F_i > F_j$, **then**
Replace j with the new solution.
end if
A fraction (P_a) of worst nests from the population are abandoned and new ones are constructed.
Keep the best individuals (or nests with quality solutions) in the population.
Rank the solutions based on the fitness function values and find the current best.
end while
Output: The current best schedule of n jobs and its corresponding MS.

5. Improved cuckoo search algorithm

In this section, we describe the proposed improved cuckoo search algorithm (ICSA) for solving the $P||C_{\max}$ problem. The details of the ICSA consist of the encoding scheme for the problem, procedural steps of the improved ICSA, initial population, heuristic to generate schedule using Levy flights, and heuristic to generate smart cuckoos.

5.1. Encoding scheme

For the given problem, we consider that an egg in the nest is a solution (here, a schedule of jobs) denoted by an individual of the population. Similarly, a cuckoo egg represents a new solution candidate for a location or position in the population in the ICSA. In this paper, the cuckoo search algorithm uses an encoding scheme having a fixed number of jobs with respect to each machine since the preliminary investigations revealed that the variation of the length of jobs in the sets does not significantly improve the solution quality but requires excessive computational effort. Using the proposed encoding scheme, the generalized schedule of jobs used for the particular scheduling problem can be represented as shown in Eq. (11) below.

$$S_1 - S_2 - S_3 ... - S_m \tag{11}$$

where $S_1, S_2, S_3, ..., S_m$ are the subsets of jobs assigned to machines $M_1, M_2, M_3, ..., M_m$ respectively.

The schedule can vary only when jobs in two subsets of $(S_1, S_2, S_3, ..., S_m)$ interchange with each other. The number of jobs assigned to each machine are determined during the search process of the proposed algorithm and a particular scheme has been decided during the generation of the initial job sequence using LPT rule for initial population and remains unaltered throughout the execution of the algorithm. In the ICSA, our objective is to determine m subsets of $n$jobs such that the *MS* given in Eq. (1) is minimum.

5.2. Steps of the ICSA

The steps of the proposed ICSA to solve the $P||C_{\max}$ problem are as follows.

Algorithm 2. Improved Cuckoo Search Algorithm

Input: A set $N = \{1, 2, ..., n\}$ of n jobs, a set $M = \{1, 2, ..., m\}$ of m identical parallel machines, processing time p_i for each job $i \in N$, MaxGeneration, and stopping criterion.
1: Generate an initial population of R schedules (described in Section 5.2.1) with the corresponding solutions and consider these solutions as R host nests $x_i (i = 1, ..., R)$. Select the best schedule from the population and consider it as the current best schedule.
2: Determine the lower bound based on the MS objective from the Eq. (3).
3: If the current best solution is equal to the lower bound, the algorithm terminates and the optimal schedule is obtained.
4: **while** $t <$ MaxGeneration or stopping criterion **do**
5: Obtain a cuckoo (here, schedule) randomly by Levy flights using Procedure 1 (given in Section 5.2.2).
6: Evaluate its quality/fitness, F_i.
7: Choose a nest (say, j) from the population of R randomly and compute its quality/fitness as F_j.
8: **if** $F_i > F_j$, **then**
9: Replace j with the new solution.
10: **end if**
11: Abondon a fraction (P_a) of worst nests from the population and construct new ones by generating quality or smart cuckoos using Procedure 2 (described in Section 5.2.3).
12: Keep the best individuals (or nests with quality solutions) in the population.
13: Rank the solutions based on the fitness function values and find the current best.
14: **end while**
Output: The current best schedule of n jobs and its corresponding MS.

5.2.1. Initial population

We generate the initial schedule of jobs for processing on the m parallel machines using the LPT algorithm (Graham, 1969). We use the LPT algorithm because it produces good quality solution in negligible computational time. To construct each independent schedule for the remaining (R-1) schedules in the population, we randomly select one job from the subset of jobs of the machine with largest completion time and another job from the subset of jobs of all other machines from the initial schedule and interchange these two jobs.

5.2.2. Heuristic to generate a schedule of jobs using Levy flights

We now present a heuristic for generating a new schedule of jobs by converting the continuous position via Levy flights to a discrete job schedule. For this purpose, we consider $x_i(t)$ as a schedule of jobs at a particular iteration t. The pseudocode for generating schedule of jobs using Levy flights is given below:

Procedure 1. Generating a Schedule Using Levy Fights

1: ▷ This pseudocode modifies the array A [ ] to array B [ ], where initially, A [ ] is the current best schedule of n jobs at a particular iteration t. k is number of digits in A[ ]. B[ ] is the generated schedule of n jobs. Array indexing is assumed to start at 0.
Input: A, length
2: Compute $d = \lfloor \alpha Levy(\lambda) \rfloor$ from Eqs. (9) and (10).
3: $c = 0$.
4: **for** $i = n-1; i \geqslant 0; i--$ **do**
5: $k = A[i]$.length.

6: $r = d\%10^k$.
7: $d = d/10^k$.
8: B [i] = A [i] + r + c.
9: c= B [i]/n.
10: B [i]= B [i]%n.
11:**end for**
12: **for** $i = 0;i < n;i + +$ **do**
13: **for** $j = i + 1;j < n;j + +$ **do**
14: **if** B [i] = B [j] **then** (i.e., check if a job in the array is duplicate).
15: **for** $p = 0;p < n;p + +$ **do**
16: $q = 0$.
17: **while** $q < n$ && B [q]! = p **do** (i.e., find out the job missing in the array).
18: $q + +$
19: **end while**
20: **if** $q = n$ **then**
21: **if** B [i] = A [i] **then**
22: B [i] = p.
23: **else**
24: B [j] = p.
25: **end if**
26: **break**
27: **end if**
28: **end for**
29: **end if**
30: **end for**
31: **end for**
Output: Obtain the generated schedule B [ ].

Table 1
Schedule representation of cuckoo x_i^t (n = 8, t = 30).

Index	0	1	2	3	4	5	6	7	8	9	10	11
schedule of jobs	2	3	6	4	8	11	7	5	10	9	1	12
A[i] (zero based schedule)	1	2	5	3	7	10	6	4	9	8	0	11
k (A[i].length)	1	1	1	1	1	2	1	1	1	1	1	2
r (d mod 10^k)	0	0	0	0	0	0	5	3	3	3	3	33
d (d = d/10^k)	0	0	0	0	0	0	0	5	53	533	5333	53333
B[i] (B[i] = A[i] + r + c)	1	2	5	3	7	10	11	8	12	11	6	44
c (c = B[i]/n)	0	0	0	0	0	0	0	1	0	0	3	0
revised B[i] (B[i] mod n)	1	2	5	3	7	10	11	8	0	11	6	8
set of missing jobs in the revised B[i]								{4, 9}				
Final B[i]	1	2	5	3	7	10	11	8	0	4	6	9
Final schedule of jobs	2	3	6	4	8	11	12	9	1	5	7	10

Fig. 1 shows the corresponding schematic diagram. In the above procedure, for computing the makespan for a particular schedule of jobs, we need only combinations of jobs assigned in the respective sets.

To illustrate the above heuristic, we consider the twelve-job, three-machine ($n = 12, m = 3$) problem taking the initial job schedule as {[2-3-6-4]-[8-11-7-5]-[10-9-1-12]} with the corresponding zero based schedule A[ ]= {[1-2-5-3]-[7-10-6-4]-[9-8-0-11]} as the current schedule. We also assume the iteration number (t) =15, $\lambda = 2$ and $\alpha = n*10^8$. Therefore, $d = \lfloor \alpha Levy(\lambda) \rfloor = \lfloor 5333333.333 \rfloor = 5333333$. The implementation of this heuristic is presented in Table 1. The output schedule of jobs B [ ] using Levy flights is obtained as {[1-2-5-3]-[7-10-11-8]-[0-4-6-9]} or {[2-3-6-4]-[8-11-12-9]-[1-15-7-10]}. Tables 2 and 3 provide a detailed illustration of Table 1 for the discretized operator of the cuckoo search algorithm.

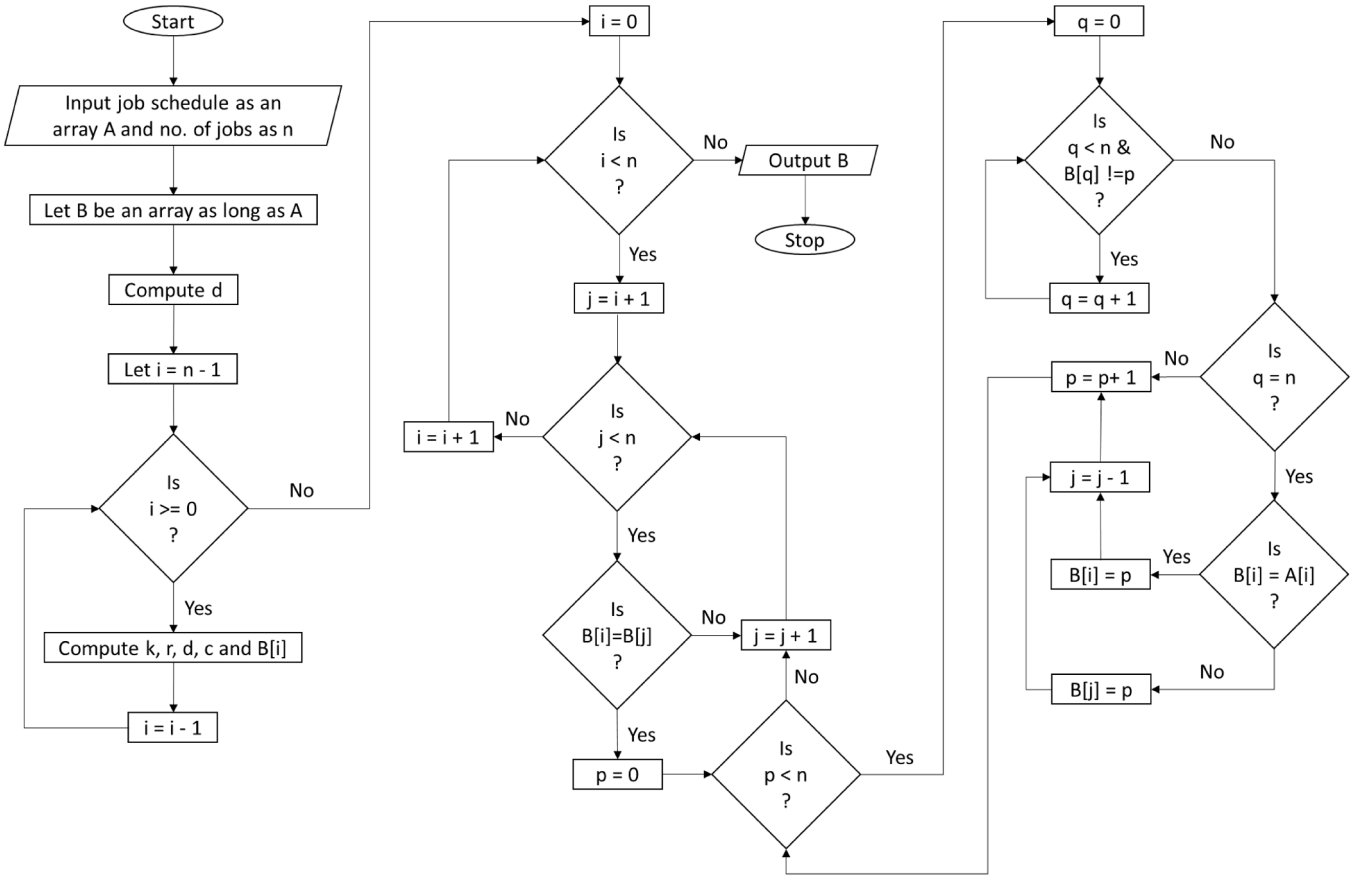

Fig. 1. The schematic diagram to illustrate the discretized operator of the cuckoo search algorithm.

Table 2
Illustration of Table 1 for the discretized operator of the cuckoo search algorithm – Part I.

i	A[i]	k (A[i].length)	$r(d \bmod 10^k)$	$d(d/10^k)$	B[i](A[i] + r + c)	c (B[i]/n)	Revised B[i] (B[i] mod n)
				5333333		0	
11	11	2	33	53333	11 + 33 + 0 = 44	3	8
10	0	1	3	5333	0 + 3+3 = 6	0	6
9	8	1	3	533	8 + 3 + 0 = 11	0	11
8	9	1	3	53	9 + 3 + 0 = 12	1	0
7	4	1	3	5	4 + 3 + 1 = 8	0	8
6	6	1	5	0	6 + 5 + 0 = 11	0	11
5	10	2	0	0	10 + 0 + 0 = 10	0	10
4	7	1	0	0	7 + 0 + 0 = 7	0	7
3	3	1	0	0	3 + 0 + 0 = 3	0	3
2	5	1	0	0	5 + 0 + 0 = 5	0	5
1	2	1	0	0	2 + 0 + 0 = 2	0	2
0	1	1	0	0	1 + 0 + 0 = 1	0	1

Table 3
Illustration of Table 1 for the discretized operator of the cuckoo search algorithm – Part II.

i	j	Duplicate (if any)	p (missing job)	q	Remarks
		$A[i]$ = {1 - 2 - 5 - 3 - 7 - 10 - 6 - 4 - 9 - 8 - 0 - 11}			
		Revised $B[i]$ = {1 - 2 - 5 - 3 - 7 - 10 - 11-8 - 0 - 11 - 6 - 8}			
0	1-12	–	–	–	–
1	2-12	–	–	–	–
2	3-12	–	–	–	–
3	4-12	–	–	–	–
4	5-12	–	–	–	–
5	6-12	–	–	–	–
6	7-9	B[6] = B[9]	4	12	Since $B[i] \neq A[i]$, $B[j] = p$, we replace
	9-12	–	–	–	job 11 by 4 in position 9 of $B[i]$.
7	8-11	B[7] = B[11]	9	12	Since $B[i] \neq A[i]$, $B[j] = p$, we replace
	11-12	–	–	–	job 8 by 9 in position 11 of $B[i]$.
8	9-12	–	–	–	–
9	10-12	–	–	–	–
10	11-12	–	–	–	–
11	12	–	–	–	–
		Final $B[i]$ = {1 - 2 - 5 - 3 - 7 - 10 - 11 - 8 - 0 - 4 - 6 - 9}			
		Final schedule of jobs is {2 - 3 - 6 - 4 - 8 - 11 - 12 - 9 - 1 - 5 - 7 - 10}.			

5.2.3. Heuristic to construct smart cuckoos in the ICSA

In the standard cuckoo search, new schedules are usually built by randomly generating the required schedules after the fraction (P_a) of worst nests from the population are abandoned. In the proposed ICSA, we present a heuristic procedure based on a pairwise exchange neighborhood to generate quality schedules, called smart cuckoos instead of generating random schedules. Thus, this proposed procedure attempts to reduce the completion time of the jobs assigned to the machine with the maximum workload among all the machines for a particular schedule. The pseudocode for generating quality schedules is given below:

Procedure 2. Constructing Smart Cuckoos

1: ▷ This pseudocode generates q distinct schedule of jobs by replacing q worst schedules from R schedules in the population MAXSET [][]. M_1 is the machine with $\max_{1 \leqslant j \leqslant m} C(S_j)$ and M [ ] is the set of remaining $(m-1)$ machines. S1 is the set of jobs assigned to M_1. S2 is the set of jobs assigned to M_2, where M_2 is a machine in M. C1 is the sum of processing times for all jobs in S1. C2 is the sum of processing times for all jobs in S2. S3 is the set of jobs assigned to all machines except M1. Array indexing is assumed to start at 0.

Input R with the corresponding makespan values in the population.
2: **Step 1**: Sort the R schedules of MAXSET in ascending order of their makespans.
3: **Step 2**: Select a schedule S [ ] from the first $(R-q)$ schedules of MAXSET as follows:
4: $i = 0$.
5: **while** 1 **do**
6: S = MAXSET[i];
7: **if** $\max_{i \in S1}(p_i) \leqslant \min_{i \in S3}(p_i)$ **then**
8: increment i.
9: **else**
10: go to Step 3.
11: **end if**
12: **if** $i \geqslant (R-q)$ **then**
13: exit.
14: **end if**
15: **end while**
16: **Step 3**: If two or more machines have the same maximum completion time in the S [ ], enter Step 4, otherwise go to Step 5.
17: **Step 4**: Let S4 and S5 be the sets of jobs in two machines with same maximum completion time. Obtain two jobs a and b ($a \in$ S4, $b \in$ S5) such that $p_a - p_b$ is minimum for any pair of jobs in S4 and S5. Interchange a and b and update S4 and S5.
18: **Step 5**: Consider two jobs u and v ($u \in$ S1, $v \in$ S2). S5 is set of $\{u, v, |p_u - p_v|\}$;
19: **for** $r = R-1; r \geqslant (R-q); r--$ **do**
20: MAXSET [r] = S;
21: ▷ Stores the jobs and their differences in completion time in S5.
22: **for** $i = 0; i < m-1; i++$ **do**
23: $M2 = M[i]$;
24: **for** $j = 0; j <$ S1.length; $j++$ **do**
25: S5 $[i][2] = 0$;
26: $u = S1[i]$;
27: **for** $k = 0; k <$ S2.length; $k++$ **do**
28: $v = S2[j]$;
29: **if** $((p_u - p_v) > 0)$ && $((p_u - p_v) <$ (C1 - C2)) && $(((p_u - p_v)$ - (C1 - LB)) $<$ $(|(S5[i][2] - (C1 - LB))|))$ **then**
30: $S5[i][0] = u$;
31: $S5[i][1] = v$;
32: $S5[i][2] = p_u - p_v$;
33: **end if**
34: **end for**
35: **end for**
36: **end for**
37: **end for**
38: ▷ sort S5

```
for i = 0;i < m−1;i + + do
    for j = 0;j < m−i−1;j + + do
        if | (C1-LB)- (S5[j][2])| > ((| (C1-LB)- (S5[j + 1][2]|))) then
            for k = 0;k ⩽ 2;k + +
                swap S5[j][k] and S5[j + 1][k];
            end for
        end if
    end for
end for
▷ Update the schedules to be replaced. If new q schedules are not found, there will be duplicates in MAXSET
for i = 0;i < q;i + + do
    if S5 [i][2] = 0 then
        break.
    end if
    for j = 0;j < n;j + + do
        if S[j]= S5[i][0] then
            MAXSET[R−i−1][j]= S5[j][1];
        else
            if S[j]= S5[i][1]
                MAXSET[R−i−1][j]=S5[j][0];
            end if
        end if
    end for
end for
Step 6: We generate new distinct schedules to replace the duplicate schedules, if any, in MAXSET.
while N1 < R do
    Remove duplicate schedules from MAXSET.
    Let the number of distinct schedules in MAXSET be N1 after removing duplicates.
    for i = R−1;i ⩾ N1;i−− do
        S = MAXSET[0];
        Select two jobs x and y (x ∈ S1, y ∈ S2) such that (p_x > p_y)
        swap x and y;
        MAXSET[i] = S.
    end for
end while
Output: q distinct schedules.
```

6. Computational experimentation

In this section, we investigate the comparative performance of the proposed algorithm with the best-known state-of-the-art procedures. For this purpose, we coded the proposed algorithm in C to run on a Pentium Dual Core, 2.80 GHZ with 3 GB RAM.

6.1. Test problems

To compare the relative performance of the proposed ICSA with SA by Lee et al. (2006), DPSO and HDPSO by Kashan and Karimi (2009), DHS and DHS_LS by Chen et al. (2011), and CSA by Laha and Behera (2015), we used the standard identical parallel-machine scheduling benchmark experimental framework originally proposed by Kedia (1971), Lee and Massey (1988) and Gupta and Ruiz-Torres (2001) and later adapted by Lee et al. (2006), Kashan and Karimi (2009) and Chen et al. (2011). Therefore, we selected the problem instances from the well-known scheduling benchmark problems as cited frequently by many authors in literature since the publication of Kedia (1971). Table 4 shows the details of this experimental framework. The jobs and machines vary in the range 6–100 and 2–10 respectively for all the experiments. For each combination of m and n of a problem in each experiment, we generated fifty independent problem instances. Hence, we considered a total number of 120 problem sizes resulting in total 6000 problem instances. As is typically used in the different experimentations, the probability distribution of processing times follows a discrete uniform distribution with different ranges, U(1, 20), U(1, 100), U(20, 50), U(50, 100), U(100, 200) and U(100, 800).

In this comparative study, we report the original results of the benchmark problems produced by the SA (Lee et al., 2006), PSO (Kashan & Karimi, 2009), DHS (Chen et al., 2011) and CSA (Laha & Behera, 2015) directly from their studies without coding these algorithms. Since a cuckoo search is a non-deterministic algorithm, it is not desirable to draw conclusions based on the results of a single run for each problem instance. Therefore, we report the best of three independent runs of the algorithm on a particular instance as the solution for each problem instance of a problem size. Since use of more than three independent runs of the proposed ICSA will most likely improve its performance, the reported results are the minimum possible improvements over the existing algorithms.

6.2. Setting the algorithm parameters

Similar to other metaheuristics, the parameters play an important role in the performance of the proposed ICSA algorithm. Therefore, we conducted several preliminary computational experiments to ascertain the best parameter values of the three control parameters of the proposed method: λ, R and P_a. Recall that P_a denotes the fraction of R nests that are to be replaced by new random nests, here, new random solutions without getting stuck in a local optimum. In our experimentation, the values of λ ($1 < \lambda \leqslant 3$) were 2 and 3. The population size R was 10, 15, or 20. The values of P_a were 0.1, 0.3 and 0.5. In order to tune the parameters of the cuckoo search algorithm, the number of jobs and the number of machines varied as: n = 10, 16, 25, 31 and m = 3, 5, 8, 10 respectively. We generate thirty independent problem instances for each problem size. The probability distribution of processing time follows a discrete uniform distribution U(1, 100).

Tables 5 and 6 show, for different problem sizes, the performance of the proposed method over a range of parameter values. It is evident from the results of Tables 5 and 6 that $P_a = 0.3$ results in diversity and fine-tuning of the solution quality. If the P_a value is smaller than 0.3, the quality of solution will not improve but the algorithm will require considerable number of additional generations. If P_a value is larger, for example, 0.5, the algorithm will converge to local optimum quickly. Thus, as a result of our preliminary computational experiments using the benchmark problem instances given in Table 4, we propose $\alpha = n * 10^8$ in Eq. (9) where, n is the number of jobs for the particular problem and $\lambda = 2$ ($1 < \lambda \leqslant 3$) in Eq. (10) to generate improved schedules considering different problem sizes. Similarly, from the results of Tables 5 and 6, the proposed algorithm performs best for the population size of 15. In order to generate a high quality schedule in a reasonable CPU time, we set the number of maximum generation (MaxGeneration) as 5000 for all the problem sizes of the given experimental framework. Therefore, the proposed ICSA terminates when the makespan equals its LB of the problem or the number of iterations exceeds MaxGeneration = 5000.

6.3. Metrics for algorithm performance

In order to compare their performance, we considered the metric for the performance measure as the average (mean) ratio (ARH1) of the MS over its LB1 obtained from Eq. (3). Thus, for a given problem size, ARH1 is defined as follows.

$$ARH1 = \frac{\sum_{i=1}^{K} \left(\frac{MS(H_i)}{LB1_i}\right)}{K} \tag{12}$$

Table 4
Summary of the experimental framework.

Experiment	m	n	p
E_1	3, 4, 5	2 m, 3 m, 5 m	U(1, 20), U(20, 50)
E_2	2, 3	10	U(100, 800)
E_2	2, 3, 4, 6, 8, 10	30, 50, 100	U(100, 800)
E_{31}	3, 5, 8, 10	3 m + 1, 3 m + 2, 4 m + 1, 4 m + 2, 5 m + 1, 5 m + 2	U(1, 100)
E_{32}	3, 5, 8, 10	3 m + 1, 3 m + 2, 4 m + 1, 4 m + 2, 5 m + 1, 5 m + 2	U(100, 200)
E_{33}	3, 5, 8, 10	3 m + 1, 3 m + 2, 4 m + 1, 4 m + 2, 5 m + 1, 5 m + 2	U(100, 800)
E_4	2	9	U(1, 20), U(20, 50), U(50, 100), U(100, 200), U(100, 800)
E_4	3	10	U(1, 20), U(20, 50), U(50, 100), U(100, 200), U(100, 800)

Table 5
Mean ratio values by the ICSA with $p \in u(1, 100)$, R and P_a with $\lambda = 2$ and MaxGeneration = 5000.

n × m	p	R = 10			R = 15			R = 20		
	$u(1, 100)$	$P_a = 0.1$	$P_a = 0.3$	$P_a = 0.5$	$P_a = 0.1$	$P_a = 0.3$	$P_a = 0.5$	$P_a = 0.1$	$P_a = 0.3$	$P_a = 0.5$
10 × 3		1.0087	1.0061	1.0056	1.0045	1.0056	1.0056	1.0056	1.0061	1.0061
16 × 5		1.0070	1.0038	1.0040	1.0073	1.0038	1.0043	1.0050	1.0038	1.0064
25 × 8		1.0061	1.0067	1.0072	1.0074	1.0041	1.0050	1.0055	1.0046	1.0047
31 × 10		1.0085	1.0079	1.0050	1.0071	1.0042	1.0052	1.0047	1.0039	1.0038

Similarly, considering the optimal solutions, the average ratio (AR1) of the optimal MS over its LB1 for a given problem size is given as

$$AR1 = \frac{\sum_{i=1}^{K} \left(\frac{MS(OPT_i)}{LB1_i} \right)}{K} \tag{13}$$

where K is the total number of problem instances for a given job and machine combination and $MS(H_i)$ is the MS value obtained by the heuristic H_i for the i-th problem instance and $MS(OPT_i)$ is the optimal MS value obtained by the complete enumeration of all possible schedules for the i-th problem instance. Similarly, considering the LB2 defined by Eq. (4), we use Eqs. (12) and (13) to calculate ARH2 and AR2 by changing $LB1_i$ to $LB2_i$.

6.4. Comparing with optimal solutions

In order to compare the solutions generated by the proposed ICSA with the optimal solutions, we considered small sized problem instances by varying $n = 7, 8, 9, 10, 11$ and $m = 2, 3, 4$. Fifty independent problem instances were taken for each combination of n and m with various processing time probability distributions U(1, 20), U(20, 50), U(50, 100), and U(100, 200). Hence, we considered a total number of 15 problem sizes resulting in a total 1350 problem instances. For computational convenience, we used complete enumeration to find the optimal makespan for each problem instance rather than any existing exact optimization algorithm for the $P||C_{max}$ problem. While comparing the performance of the ICSA with the optimal solutions, we consider the mean ratio values obtained from Eqs. (12) and (13) for the enumeration method and the proposed method. Table 7 shows the comparative evaluations of the complete enumeration and the proposed method. Table 7 also shows the average ratio of the ICSA makespan and the optimal makespan. These results in Table 7 show that the solutions obtained by the proposed ICSA are very close to the optimal solutions produced by the complete enumeration.[1]

6.5. Comparison with the existing algorithms

We compared the proposed ICSA with the SA (2006), DPSO (2009), HDPSO (2009) and CSA (2015). The proposed ICSA differs from the CSA (2015) with respect to the generation of initial population (described in Section 5.2.1), the heuristic to obtain new cuckoos using Levy flights (described in Section 5.2.2), and the heuristic to construct improved schedules replacing the generated duplicate ones in the population (described in Section 5.2.3). The detailed computational results of experimental frameworks E_1, E_2, E_3 and E_4 for the proposed ICSA and the four existing heuristics (SA (2006), DPSO (2009), HDPSO (2009) and CSA (2015)) are given in Tables 8–13 respectively. Since the other two existing algorithms (DHS (2011) and DHS_LS (2011)) do not report the computational results for each of the four experimental frameworks, their corresponding results are not reported in these tables.

Table 8 displays the comparative evaluation of the proposed ICSA and the existing methods for experimental framework E_1. The computational results in terms of the mean ratio performance measure given in Eq. (12) show that the proposed ICSA performs significantly better than the SA, DPSO, HDPSO and CSA. Among the existing algorithms, CSA performed next best, followed by HDPSO, DPSO and SA respectively. While comparing the CSA and the proposed ICSA in this experiment, it is seen that the mean ratio value obtained by the ICSA is either better than or same as that obtained by the CSA in all 18 problem sizes.

The comparative results of the algorithms depicted in Table 9 for the Experiment E_2 reveal that overall, the average mean ratio value obtained by the ICSA is the same as that of the HDPSO. In this experiment, it is seen that in most of the problems, the performance of the ICSA is either better than or same as the HDPSO except in two cases with $n = 10$, $m = 2$, and $n = 30$, $m = 10$ out of 20 problem sizes. Also, we observe that the CSA performs better than the DPSO and SA.

Tables 10–12 present the mean ratio values obtained from various algorithms for the Experiment E_{31}, E_{32}, and E_{33} respectively. Considering the results of Table 10, overall, there is no significant improvement of the ICSA over the HDPSO. However, ICSA is much superior to CSA, DPSO and SA. The results of Table 11 for the Experiment E_{32} depict that the ICSA significantly performs the best and SA is the worst performer among all the algorithms. Also, the HDPSO and the CSA are comparable and are better than the DPSO and SA. The results of Table 12 show that overall, the ICSA and the HDPSO are comparable, but, the ICSA takes much less computational time compared to that of the HDPSO. In regard to the relative performance of the other three algorithms, the proposed ICSA significantly outperforms the SA, DPSO, and ICSA.

[1] The use of commercial solvers like IBM ILOG CPLEX or Gurobi to optimally solve the mathematical formulation of the problem will not change these results.

Table 6
Mean ratio values by the ICSA with $p \in u(1, 100)$, R and P_a with $\lambda = 3$ and MaxGeneration = 5000.

n × m	p	R = 10			R = 15			R = 20		
	$u(1, 100)$	$P_a = 0.1$	$P_a = 0.3$	$P_a = 0.5$	$P_a = 0.1$	$P_a = 0.3$	$P_a = 0.5$	$P_a = 0.1$	$P_a = 0.3$	$P_a = 0.5$
10 × 3		1.0068	1.0056	1.0056	1.0078	1.0056	1.0068	1.0056	1.0061	1.0120
16 × 5		1.0059	1.0042	1.0040	1.0051	1.0045	1.0043	1.0044	1.0043	1.0057
25 × 8		1.0069	1.0074	1.0049	1.0074	1.0044	1.0064	1.0039	1.0047	1.0069
31 × 10		1.0074	1.0062	1.0061	1.0060	1.0057	1.0053	1.0074	1.0065	1.0044

Table 7
Mean ratio values by the complete enumeration method and the ICSA.

m	n	p	*LB1*		*LB2*		ICSA/OPT
			AR1	ARH1	AR2	ARH2	ARH1/AR1
2	9	U(1, 20)	1.000	1.000	1.000	1.000	1.000
		U(20, 50)	1.001	1.001	1.001	1.001	1.000
		U(50, 100)	1.003	1.003	1.003	1.003	1.000
	11	U(100, 200)	1.000	1.000	1.000	1.000	1.000
3	7	U(1, 20)	1.040	1.041	1.016	1.017	1.001
	8		1.020	1.023	1.014	1.016	1.003
	9		1.011	1.014	1.008	1.012	1.003
		U(20, 50)	1.010	1.011	1.010	1.011	1.001
	10	U(50, 100)	1.012	1.012	1.012	1.012	1.000
		U(100, 200)	1.015	1.018	1.013	1.018	1.003
	11		1.010	1.010	1.010	1.010	1.000
4	7	U(1, 20)	1.072	1.073	1.022	1.023	1.001
	8		1.068	1.071	1.036	1.039	1.003
	9		1.039	1.039	1.025	1.025	1.000
	8	U(20, 50)	1.050	1.052	1.038	1.040	1.002
	9		1.049	1.053	1.048	1.048	1.004
	7	U(100, 200)	1.135	1.135	1.046	1.046	1.000
	8		1.038	1.038	1.024	1.024	1.000
	Average		1.032	1.033	1.018	1.019	1.001

Table 13 presents the mean ratio values of the algorithms for the Experiment E_4. It is evident from the results of Table 13 that the ICSA performs best among the rest of the algorithms, the DPSO is the next best followed by the HDPSO, CSA and SA. Comparing with HDPSO and the ICSA, out of 12 problem sizes, ICSA produces better solutions eight times more than HDPSO, and the HDPSO outperforms ICSA for only one instance with $n = 10$, $m = 3$ with U(1, 20). Similar to the results of Tables 8–12, in this experimentation E_4, we observe that SA performs the worst among the algorithms.

Table 14 exhibits the overall summarized performance of the algorithms considering all experiments for all seven algorithms including the summarized performance of algorithms DHS and DHS_LS reported by Chen et al. (2011). Out of the six experiments, it is observed that overall, the average mean ratio value of the ICSA is the best among the existing algorithms. It can be seen from all the experimental results that the proposed ICSA significantly outperforms the SA, DPSO, and DHS and it is also better than the hybrid versions of PSO (HDPSO) and DHS (DHS_LS).

6.6. Computational times

To obtain a fair comparison of the efficiency of the proposed method with the existing algorithms, we noted the average CPU times (seconds) required by the proposed method for running randomly 100 times of each problem size from the set of benchmark problems. The detailed CPU times taken by the algorithms are given in Tables 3–11 where the original computational results of the average CPU times for the benchmark problems produced by the SA (Lee et al., 2006), PSO (Kashan & Karimi, 2009), DHS (Chen et al., 2011) and CSA (Laha & Behera, 2015) are reported directly from their studies without coding these algorithms. Table 15 summarizes the average CPU times for the algorithms considering each experiment.

The results of Table 15 show that the proposed ICSA requires an average of 0.18 s whereas it is 2.67 2.51, 2.03, 0.05, and 0.14 s for the SA, DPSO, HDPSO, DHS, DHS_LS, and CSA respectively. However, use of different CPU processors for running the algorithms such as Pentium 4, 2.66 GHz CPU processor for computing CPU times of SA, DPSO and HDPSO, Pentium 2 CPU 2.0 GHz processor for DHS and DHS_LS, and the Pentium Dual Core, 2.80 GHz CPU for the CSA and the proposed ICSA make it is not possible to directly and specifically compare the CPU times required to solve a problem instance by these algorithms.

Table 8
Results for experiment E_1 using U(1, 20) and U(20, 50).

m	n	p	SA (2006)		DPSO (2009)		HDPSO (2009)		CSA (2015)		ICSA	
			Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time
3	6	U(1, 20)	1.059	0.06	1.059	0.89	1.059	0.92	1.070	0.01	1.063	0.12
	9		1.009	0.05	1.009	0.55	1.009	0.53	1.009	0.01	1.005	0.06
	15		1.000	0.01	1.000	0.02	1.000	0.02	1.000	0	1.000	0
	6	U(20, 50)	1.057	0.07	1.057	1.17	1.057	1.23	1.049	0.01	1.049	0.11
	9		1.008	0.09	1.008	1.09	1.008	1.12	1.011	0.03	1.007	0.09
	15		1.001	0.10	1.000	0.13	1.000	0.09	1.000	0	1.000	0
4	8	U(1, 20)	1.069	0.08	1.069	1.56	1.069	1.53	1.067	0.05	1.047	0.14
	12		1.004	0.04	1.002	0.13	1.002	0.13	1.007	0.01	1.001	0.04
	20		1.000	0.06	1.000	0.03	1.000	0.02	1.000	0	1.000	0
	8	U(20, 50)	1.060	0.1	1.060	1.89	1.060	1.80	1.052	0.06	1.046	0.15
	12		1.011	0.18	1.009	1.42	1.009	1.50	1.009	0.03	1.008	0.13
	20		1.001	0.20	1.000	0.08	1.000	0.04	1.000	0	1.000	0
5	10	U(1, 20)	1.059	0.10	1.059	1.54	1.059	1.73	1.052	0.03	1.047	0.10
	15		1.010	0.11	1.008	0.45	1.008	0.45	1.005	0.01	1.001	0.03
	25		1.000	0.10	1.000	0.03	1.000	0.03	1.000	0	1.000	0.01
	10	U(20, 50)	1.059	0.13	1.059	2.07	1.059	2.28	1.052	0.04	1.045	0.15
	15		1.009	0.25	1.007	1.46	1.007	1.59	1.007	0.03	1.005	0.12
	25		1.001	0.30	1.000	0.24	1.000	0.04	1.000	0	1.000	0
	Average		1.023	0.11	1.023	0.81	1.023	0.83	1.022	0.02	1.018	0.07

Table 9
Results for experiment E_2 using U(100, 800).

m	n	SA (2006)		DPSO (2009)		HDPSO (2009)		CSA (2015)		ICSA	
		Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time
2	10	1.0009	0.15	1.0008	1.25	1.0008	1.37	1.0011	0.040	1.0013	0.18
3		1.0071	0.21	1.0063	1.97	1.0063	2.17	1.0065	0.053	1.0058	0.18
2	30	1.0000	0.21	1.0000	0.07	1.0000	0.06	1.0000	0.002	1.0000	0
3		1.0002	1.53	1.000	1.77	1.0000	0.32	1.0000	0.026	1.0000	0.01
4		1.0009	2.35	1.0004	3.11	1.0000	1.44	1.0001	0.064	1.0000	0.15
6		1.0027	2.13	1.0019	3.48	1.0004	4.81	1.0007	0.109	1.0003	0.29
8		1.0058	2.49	1.0050	3.64	1.0014	5.94	1.0025	0.011	1.0014	0.44
10		1.0109	2.08	1.0095	3.82	1.0035	6.75	1.0079	0.116	1.0049	0.43
2	50	1.0000	0.25	1.0000	0.09	1.0000	0.14	1.0000	0.002	1.0000	0
3		1.0001	4.11	1.0000	1.72	1.0000	0.18	1.0000	0.004	1.0000	0.01
4		1.0004	6.83	1.0001	4.18	1.0000	0.46	1.0000	0.023	1.0000	0.02
6		1.0014	7.89	1.0009	5.42	1.0000	3.07	1.0001	0.117	1.0000	0.22
8		1.0027	8.94	1.0020	5.72	1.0002	6.75	1.0003	0.152	1.0001	0.37
10		1.0031	7.91	1.0035	6.01	1.0005	10.17	1.0007	0.176	1.0004	0.60
2	100	1.0000	0.86	1.0000	0.10	1.0000	0.92	1.0000	0.002	1.0000	0
3		1.0000	12.19	1.0000	1.13	1.0000	0.44	1.0000	0.005	1.0000	0
4		1.0000	16.36	1.0000	6.79	1.0000	0.36	1.0000	0.009	1.0000	0.01
6		1.0002	32.10	1.0003	11.40	1.0000	0.49	1.0000	0.045	1.0000	0.10
8		1.0005	38.63	1.0007	12.19	1.0000	1.25	1.0000	0.108	1.0000	0.33
10		1.0000	40.01	1.0012	12.95	1.0000	3.65	1.0000	0.188	1.0000	0.48
Average		1.0018	9.36	1.0016	4.34	1.0007	2.53	1.0010	0.063	1.0007	0.19

Table 10
Results for experiment E_{31} using U(1, 100).

m	n	SA (2006)		DPSO (2009)		HDPSO (2009)		CSA (2015)		ICSA	
		Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time
3	10	1.0115	0.14	1.0107	1.37	1.0107	1.54	1.0102	0.037	1.0056	0.14
	11	1.0045	0.14	1.0037	0.92	1.0037	1.01	1.044	0.028	1.0032	0.11
	13	1.0019	0.15	1.0005	0.37	1.0003	0.23	1.0017	0.023	1.0007	0.06
	14	1.0009	0.15	1.0000	0.14	1.0000	0.06	1.0008	0.018	1.0000	0.03
	16	1.0004	0.16	1.0000	0.09	1.0000	0.04	1.0003	0.012	1.0000	0.03
	17	1.0005	0.14	1.0000	0.04	1.0000	0.04	1.0002	0.012	1.0000	0.01
5	16	1.0096	0.39	1.0046	1.64	1.0039	1.76	1.0056	0.048	1.0038	0.16
	17	1.0086	0.5	1.0036	1.55	1.0024	1.45	1.0055	0.051	1.0027	0.17
	21	1.0031	0.52	1.0006	0.84	1.0000	0.28	1.0012	0.028	1.0000	0.08
	22	1.0030	0.50	1.0010	0.96	1.0000	0.25	1.0013	0.032	1.0000	0.05
	26	1.0020	0.60	1.0004	0.71	1.0000	0.13	1.0002	0.022	1.0000	0.04
	27	1.0010	0.57	1.0003	0.63	1.0000	0.10	1.0002	0.013	1.0000	0.01
8	25	1.0127	1.00	1.0094	2.97	1.0041	2.02	1.0067	0.078	1.0041	0.25
	26	1.0095	0.98	1.0069	3.01	1.0016	1.68	1.0061	0.082	1.0022	0.26
	33	1.0030	1.27	1.0028	2.83	1.0002	0.61	1.0013	0.048	1.0004	0.12
	34	1.0033	1.46	1.0027	2.81	1.0001	0.64	1.0016	0.055	1.0000	0.12
	41	1.0021	1.62	1.0012	2.54	1.0000	0.37	1.0008	0.040	1.0000	0.12
	42	1.0018	1.64	1.0013	2.39	1.0000	0.29	1.0003	0.026	1.0000	0.08
10	31	1.0123	1.42	1.0120	3.82	1.0050	3.41	1.0069	0.089	1.0042	0.30
	32	1.0094	1.38	1.0089	3.63	1.0011	2.40	1.0065	0.092	1.0031	0.26
	41	1.0039	2.15	1.0037	3.91	1.0004	1.24	1.0024	0.081	1.000	0.25
	42	1.0045	2.58	1.0040	4.31	1.0005	1.59	1.0020	0.070	1.0009	0.23
	51	1.0030	3.14	1.0021	4.31	1.0000	0.53	1.0006	0.051	1.0000	0.15
	52	1.0026	3.08	1.0022	4.41	1.0000	0.37	1.0006	0.042	1.0000	0.12
Average		1.0048	1.07	1.0034	2.09	1.0014	0.91	1.0028	0.045	1.0013	0.13

Because of these differing computer capabilities (like speeds of computations, system configuration, memory, and number of processors), programming languages, and the differences in the programming expertise of various researchers, we can only draw very general conclusions. Therefore, based on the results in Table 15, we can only say that the CPU times of SA, DPSO and HDPSO are considerably more than those for the DHS, DHS_LS, CSA, and the proposed ICSA. In addition, it appears that there are insignificant differences in the CPU times required by heuristics DHS, DHS_LS, CSA, and the proposed ICSA. Since the times for single executions are so small (and the differences even smaller), it is unlikely that the CPU times will be a deciding factor among these four heuristics (please see Aldowaisan & Allahverdi (2003) for similar observations).

6.7. Discussion of results

The above computational results show that the proposed algorithm is relatively more effective than the six state-of-the-art algorithms. We think this is due to two special features of the proposed algorithm. First, the use global random walks governed by the Levy flights distribution that follows a power law in terms of step-length with a heavy-tailed probability distribution with an infinite mean and variance enables the proposed ICSA to explore search space more efficiently to possibly converge to global optima (Yang & Deb, 2014). Moreover, the proposed ICSA permits to take small steps when located in the neighborhood of the current best solution, resulting in reducing the unnecessary oscillatory motion near global optima and consequently, increases the

Table 11
Results for experiment E_{32} using U(100, 200).

m	n	SA (2006)		DPSO (2009)		HDPSO (2009)		CSA (2015)		ICSA	
		Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time
3	10	1.0122	0.22	1.0119	2.07	1.0119	2.23	1.0123	0.051	1.0087	0.24
	11	1.0137	0.28	1.0129	2.08	1.0125	2.32	1.0094	0.048	1.0074	0.18
	13	1.0024	0.34	1.0011	1.56	1.0010	1.36	1.0023	0.035	1.0006	0.11
	14	1.0021	0.38	1.0014	1.43	1.0007	0.75	1.0011	0.034	1.0007	0.11
	16	1.0010	0.50	1.0002	0.77	1.0000	0.18	1.0002	0.017	1.0000	0.04
	17	1.011	0.50	1.0002	0.89	1.0000	0.13	1.0001	0.015	1.0000	0.05
5	16	1.0130	0.63	1.0097	2.62	1.0087	3.18	1.0085	0.086	1.0050	0.43
	17	1.0141	0.80	1.0122	2.72	1.0096	3.35	1.0076	0.061	1.0049	0.21
	21	1.0043	1.07	1.0014	2.60	1.0000	0.62	1.0006	0.038	1.0000	0.12
	22	1.0042	1.22	1.0020	2.82	1.0002	0.98	1.0007	0.039	1.0002	0.12
	26	1.0035	1.39	1.0007	2.51	1.0000	0.30	1.0002	0.026	1.0000	0.03
	27	1.0027	1.82	1.0013	2.82	1.0000	0.35	1.0002	0.030	1.0000	0.05
8	25	1.0118	1.53	1.0071	3.36	1.0030	3.79	1.0031	0.084	1.0023	0.30
	26	1.0129	1.59	1.0077	3.44	1.0023	3.74	1.0033	0.087	1.0022	0.24
	33	1.0060	2.94	1.0027	3.77	1.0000	0.65	1.0007	0.063	1.0003	0.14
	34	1.0068	3.13	1.0032	3.97	1.0000	0.95	1.0005	0.060	1.0003	0.16
	41	1.0051	4.40	1.0019	4.29	1.0000	0.72	1.0003	0.052	1.0000	0.08
	42	1.0047	4.80	1.0022	4.65	1.0000	0.50	1.0002	0.044	1.0000	0.10
10	31	1.0134	2.34	1.0072	3.99	1.0015	3.58	1.0023	0.088	1.0016	0.35
	32	1.0137	2.76	1.0081	4.10	1.0016	4.06	1.0025	0.098	1.0017	0.32
	41	1.0091	4.60	1.0036	4.85	1.0002	1.54	1.0007	0.081	1.0002	0.27
	42	1.0085	4.86	1.0043	4.85	1.0000	1.03	1.0007	0.072	1.0003	0.26
	51	1.0057	8.36	1.0026	5.78	1.0000	0.54	1.0004	0.071	1.0002	0.18
	52	1.0065	8.05	1.0033	5.88	1.0000	0.97	1.0003	0.069	1.0000	0.20
Average		1.0079	2.43	1.0045	3.24	1.0022	1.57	1.0024	0.056	1.0015	0.18

Table 12
Results for experiment E_{33} using U(100, 800).

m	n	SA (2006)		DPSO (2009)		HDPSO (2009)		CSA (2015)		ICSA	
		Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time
3	10	1.0089	0.22	1.0078	2.12	1.0078	2.39	1.0081	0.050	1.0058	0.18
	11	1.0056	0.28	1.0047	2.11	1.0046	2.43	1.0051	0.052	1.0027	0.20
	13	1.0024	0.37	1.0013	2.20	1.0009	2.65	1.0015	0.057	1.0010	0.21
	14	1.0018	0.48	1.0006	1.96	1.0005	2.15	1.0010	0.054	1.0008	0.19
	16	1.0014	0.56	1.0003	2.00	1.0001	1.69	1.0005	0.053	1.0003	0.19
	17	1.0014	0.63	1.0002	1.77	1.0000	1.22	1.003	0.047	1.0002	0.18
5	16	1.0102	0.56	1.0072	2.63	1.0057	3.43	1.0080	0.066	1.0049	0.25
	17	1.0088	0.71	1.0047	2.70	1.0032	3.53	1.0046	0.069	1.0030	0.26
	21	1.0046	1.19	1.0020	2.90	1.0006	3.71	1.0016	0.078	1.0008	0.30
	22	1.0046	1.20	1.0019	2.97	1.0005	3.69	1.0014	0.084	1.0006	0.30
	26	1.0035	1.76	1.0013	3.17	1.0003	3.53	1.0005	0.087	1.0003	0.32
	27	1.0029	1.94	1.0012	3.16	1.0002	3.82	1.0005	0.090	1.0002	0.30
8	25	1.0119	1.58	1.0087	3.26	1.0032	5.17	1.0065	0.098	1.0045	0.37
	26	1.0096	1.90	1.0076	3.33	1.0026	5.32	1.0053	0.099	1.0026	0.37
	33	1.0058	3.48	1.0043	3.88	1.0011	6.35	1.0016	0.126	1.0007	0.46
	34	1.0064	3.50	1.0040	4.08	1.0009	6.62	1.0016	0.126	1.0007	0.46
	41	1.0036	6.03	1.0026	4.76	1.0004	7.31	1.0007	0.145	1.0003	0.48
	42	1.0041	6.10	1.0025	4.87	1.0004	7.56	1.0006	0.147	1.0003	0.51
10	31	1.0125	2.51	1.0092	3.87	1.0032	7.03	1.0066	0.119	1.0037	0.45
	32	1.0118	2.99	1.0082	3.98	1.0026	7.21	1.0049	0.123	1.0029	0.46
	41	1.0059	5.84	1.0050	5.00	1.0009	8.93	1.0017	0.148	1.0007	0.55
	42	1.0066	6.81	1.0051	5.09	1.0010	8.89	1.0014	0.153	1.0007	0.56
	51	1.0046	9.27	1.0035	6.12	1.0004	10.44	1.0007	0.174	1.0002	0.59
	52	1.0042	9.89	1.0035	6.19	1.0004	10.38	1.0007	0.181	1.0003	0.55
Average		1.0060	2.90	1.0041	3.50	1.0017	5.22	1.0027	0.101	1.0016	0.36

convergence speed by restricting the diversification. However, the algorithm has the risk of trapping its solution into local optima. This is overcome by randomly generating a considerable fraction of new solutions during *abandon worst nests and replace them with new ones* step of the ICSA. As a result, their locations should be far enough from the current best solution so that the system will not be trapped in a local optimum. In addition, the number of parameters to be tuned in the cuckoo search algorithm is less than other existing metaheuristics. Second, the use of a new proposed construction heuristic procedure, based on a pairwise exchange neighborhood, to generate smart cuckoos improves the proposed ICSA and hence produces lower makespan schedules.

7. Conclusions

We have presented a cuckoo search algorithm for the identical parallel-machine scheduling problem with the objective of minimizing makespan. In the proposed cuckoo search algorithm, we used a heuristic approach based on modulus operator to convert a continuous position in the cuckoo search algorithm into a discrete job schedule for

Table 13
Results for experiment E_4.

m	n	p	SA (2006)		DPSO (2009)		HDPSO (2009)		CSA (2015)		ICSA	
			Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time	Mean ratio	CPU time
2	9	U(1, 20)	1.001	0.00	1.000	0.01	1.000	0.01	1.000	0	1.000	0
		U(20, 50)	1.001	0.03	1.001	0.22	1.001	0.21	1.001	0.04	1.001	0.05
		U(50, 100)	1.004	0.11	1.004	1.07	1.004	1.07	1.004	0.02	1.003	0.14
		U(100, 200)	1.004	0.14	1.004	1.22	1.004	1.23	1.004	0.04	1.004	0.20
		U(100, 800)	1.002	0.14	1.002	1.33	1.002	1.46	1.002	0.01	1.001	0.23
3	10	U(1, 20)	1.001	0.02	1.001	0.14	1.001	0.14	1.001	0	1.002	0.02
		U(20, 50)	1.008	0.14	1.007	1.16	1.007	1.20	1.007	0.03	1.005	0.14
		U(50, 100)	1.010	0.20	1.009	1.61	1.009	1.65	1.010	0.05	1.006	0.22
		U(100, 200)	1.017	0.22	1.016	2.00	1.016	2.08	1.016	0.06	1.008	0.24
		U(100, 800)	1.009	0.22	1.009	1.95	1.009	2.22	1.009	0.05	1.006	0.18
Average			1.0057	0.12	1.0053	1.07	1.0053	1.13	1.0054	0.03	1.0036	0.14

Table 14
Summary of solution quality for the algorithms for different experiments.

Experiment	Mean ratio						
	SA (2006)	DPSO (2009)	HDPSO (2009)	DHS (2011)	DHS_LS (2011)	CSA (2015)	ICSA
E_1	1.0232	1.0226	1.0226	1.0215	1.0204	1.0216	1.0180
E_2	1.0018	1.0016	1.0007	1.0069	1.0008	1.0010	1.0007
E_{31}	1.0048	1.0034	1.0014	1.0148	1.0011	1.0028	1.0013
E_{32}	1.0079	1.0045	1.0022	1.0174	1.0022	1.0024	1.0015
E_{33}	1.0060	1.0041	1.0017	1.0169	1.0014	1.0027	1.0016
E_4	1.0057	1.0053	1.0053	1.0064	1.0061	1.0054	1.0036
Average	1.0082	1.0069	1.0057	1.0142	1.0053	1.0060	1.0044

Table 15
Summary of CPU times for the algorithms for different experiments.[a]

Experiment	CPU time (s)						
	SA (2006)	DPSO (2009)	HDPSO (2009)	DHS (2011)	DHS_LS (2011)	CSA (2015)	ICSA
E_1	0.11	0.81	0.83	0.01	0.04	0.02	0.07
E_2	9.36	4.34	2.53	0.09	0.16	0.06	0.19
E_{31}	1.07	2.09	0.91	0.05	0.04	0.05	0.13
E_{32}	2.43	3.24	1.57	0.058	0.07	0.06	0.18
E_{33}	2.90	3.50	5.22	0.06	0.46	0.10	0.36
E_4	0.12	1.07	1.13	0.01	0.08	0.03	0.14
Average	2.67	2.51	2.03	0.05	0.14	0.05	0.18

[a] Because of different computer capabilities (like speeds of computations, system configuration, memory, and number of processors), programming languages, and the differences in the programming expertise of various researchers, it is not possible to directly compare the CPU times, except to draw very general conclusions as discussed in the text below.

obtaining a cuckoo by Levy flights. Moreover, we developed a new construction heuristic to generate improved cuckoos in the proposed algorithm.

The computational results demonstrated that the proposed algorithm produces better solutions than those by the existing best-known algorithms. The cuckoo search algorithm proposed in this paper is faster than other competing algorithms. By implementing a new construction heuristic for generating smart cuckoos, the proposed algorithm not only performs better than the existing state-of-the-art-procedures but also require less computational effort than the existing algorithms.

Several research issues are worth exploring in the future. First, it would be useful to assume the variable job scheme instead of fixed job scheme during the search of new schedules by the algorithm even though it may increase the complexity of the algorithm. Second, it would be interesting to develop other approaches to convert a continuous position in cuckoo search into a discrete schedule to generate a cuckoo by Levy flights. Third, seeking better heuristics to generate smart cuckoos is a promising area for future work. Finally, extending this technique to other complex machine scheduling environments such as unrelated parallel machines, job shops, and hybrid flow shops to enhance the application of scheduling theory in scheduling literature.

Acknowledgement

We thank the reviewers whose constructive comments on an earlier draft considerably improve the quality and presentation of this paper.

References

Abdel-Basset, M., Hessin, A-N., & Abdel-Fatah, L. (2018). A comprehensive study of cuckoo-inspired algorithms. *Neural Computations and Applications, 29*, 345–361.
Aldowaisan, T., & Allahverdi, A. (2003). New heuristics for no-wait flowshops to minimize makespan. *Computers and Operations Research, 30*, 1219–1231.
Brown, C., Liebovitch, L. S., & Glendon, R. (2007). Levy flights in Dobe Ju/hoansi foraging patterns. *Human Ecology, 35*, 129–138.
Burnwal, S., & Deb, S. (2013). Scheduling optimization of flexible manufacturing system using cuckoo search-based approach. *International Journal of Advanced Manufacturing Technology, 64*, 951–959.
Chandrasekharan, K., & Simon, S. P. (2012). Multi-objective scheduling problem: Hybrid approach using fuzzy assisted cuckoo search algorithm. *Swarm and Evolutionary Computation, 5*, 1–16.
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*, 271–292.
Chen, J., Pan, Q. K., Wang, L., & Li, J. Q. (2011). A hybrid harmony search algorithm for identical parallel machines scheduling. *Engineering Optimization, 1*, 1–16.
Chen, Z-L., & Powell, W. B. (1999). Solving parallel machine scheduling problems by column generation. *INFORMS Journal on Computing, 11*, 78–94.
Coffman, E. G., Garey, M. R., & Johnson, D. S. (1978). An application of bin-packing to multi- processor scheduling. *SIAM Journal of Computing, 7*, 1–17.
Dasgupta, P., & Das, S. (2015). A discrete inter-species cuckoo search for flowshop problems. *Computers and Operations Research, 60*, 111–120.
Dell'Amico, M., Iori, M., Martello, S., & Monaci, M. (2008). Heuristic and exact algorithms for the identical parallel machine scheduling problem. *INFORMS Journal on Computing, 20*, 333–344.
Dell'Amico, M., & Martello, S. (1995). Optimal scheduling of tasks on identical parallel processors. *ORSA Journal on Computing, 7*, 191–200.
Dell'Amico, M., & Martello, S. (2005). A note on exact algorithm for the identical parallel machine scheduling problem. *European Journal of Operational Research, 160*, 576–578.
Expósito-Izquierdo, C., & Expósito-Márquez, A. (2017). A survey of the cuckoo search and

its applications in real-world optimization problems. *Handbook of research on soft computing and nature-inspired algorithms*. IGI Global.
Garey, M. R., & Johnson, J. S. (1979). *Computers and intractability: A guide to the theory of NP-Completeness.* San Francisco, CA: Freeman.
Ghomi, S. M. T., & Ghazvini, F. J. (1998). A pairwise interchange algorithm for parallel machine scheduling. *Production Planning and Control, 9*, 685–689.
Graham, R. L. (1969). Bounds on multiprocessor timing anomalies. *SIAM Journal of Applied Mathematics, 17*, 416–429.
Guo, P., Cheng, W., & Wang, Y. (2015). Parallel machine scheduling with step-deteriorating jobs and setup times by a hybrid discrete cuckoo search algorithm. *Engineering Optimization, 47*(11), 1564–1585.
Gupta, J. N. D., & Ruiz-Torres, J. (2001). A LISTFIT heuristic for minimizing makespan on identical parallel machines. *Production Planning and Control, 12*, 28–36.
Haouari, M., & Jemmali, M. (2008). Tight bounds for the identical parallel machine-scheduling problem: Part II. *International Transactions in Operational Research, 15*, 19–34.
Kashan, A. H., & Karimi, B. (2009). A discrete particle swarm optimization algorithm for scheduling parallel machines. *Computers and Industrial Engineering, 56*, 216–223.
Kedia, S. K. (1971) A job scheduling problem with parallel processors. Unpublished Report. Ann Arbor, MI: Department of Industrial and Operations Engineering, University of Michigan.
Komaki, G. M., Teymourinan, E., Kayvanfar, V., & Booyavi, Z. (2017). Improved discrete cuckoo optimization algorithm for the three-stage assembly flowshop scheduling problem. *Computers and Industrial Engineering, 105*, 158–173.
Laha, D., & Behera, D. K. (2015). An improved cuckoo search algorithm for parallel machine scheduling. *Lecture Notes in Computer Science, 8947*, 788–800.
Laha, D., & Behera, D. K. (2017). A comprehensive review and evaluation of LPT, MULTIFIT, COMBINE and LISTFIT for scheduling identical parallel machines. *International Journal of Information and Communication Technology, 11*, 151–165.
Lee, C. Y., & Massey, J. D. (1988). Multiprocessor scheduling: Combining LPT and MULTIFIT. *Discrete Applied Mathematics, 20*, 233–242.
Lee, W.-C., Wu, C.-C., & Chen, P. (2006). A simulated annealing approach to makespan minimization on identical parallel machines. *International Journal of Advanced Manufacturing Technology, 31*, 328–333.
Li, X., & Yin, M. A. (2013). A hybrid cuckoo search via Levy flights for the permutation flow shop scheduling problem. *International Journal of Production Research, 51*, 4732–4754.
Majumder, A., & Laha, D. (2016). A new cuckoo search algorithm for 2-machine robotic cell scheduling problem with sequence-dependent setup times. *Swarm and Evolutionary Computation, 28*, 131–143.
Marichelvam, M. K., Prabaharan, T., & Yang, X. S. (2014a). Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Applied Soft Computing, 19*, 93–101.
Marichelvam, M. K., Prabaharan, T., & Yang, X. S. (2014b). A discrete firefly algorithm for the multi-objective hybrid flowshop scheduling problems. *IEEE Transactions on Evolutionary Computation, 18*, 301–305.
McNaughton, R. (1959). Scheduling with deadlines and loss functions. *Management Science, 6*, 1–8.
Min, L., & Cheng, W. (1999). A genetic algorithm for minimizing the makespan in the case of scheduling identical parallel machines. *Artificial Intelligence in Engineering, 13*, 399–403.
Mohamad, A. B., Zain, A. M., & Bazin, N. E. (2014). Cuckoo search algorithm for optimization problems – A literature review and its applications. *Applied Artificial Intelligence, 28*, 419–448.
Mokotoff, E. (2001). Parallel machine scheduling problems: A survey. *Asia-Pacific Journal of Operational Research, 18*, 193–242.
Mokotoff, E. (2004). An exact algorithm for the identical parallel machine scheduling problem. *European Journal of Operational Research, 152*, 758–769.
Ouaarab, A., Ahiod, B., & Yang, X. S. (2014). Discrete cuckoo search algorithm for the travelling salesman. *Neural Computing and Applications, 24*, 1659–1669.
Payne, R. B., Sorenson, M. D., & Klitz, K. (2005). *The cuckoos.* Oxford University Press.
Shim, S-O., & Kim, Y. D. (2008). branch and bound algorithm for an identical parallel machine scheduling problem with a job splitting property. *Computers and Operations Research, 35*, 863–875.
Su, L. H., & Lien, C. Y. (2009). Scheduling parallel machines with resource-dependent processing times. *International Journal of Production Economics, 17*, 256–266.
Tran, T. T., Araujo, A., & Beck, J. C. (2016). Decomposition methods for the parallel machine scheduling problem with setups. *INFORMS Journal on Computing, 28*, 83–95.
Valian, E., Tavakoli, S., Mohanna, S., & Haghi, A. (2013). Improved cuckoo search for reliability optimization problems. *Computers and Industrial Engineering, 64*, 459–468.
Valian, E., & Valian, E. (2013). A cuckoo search algorithm by Levy flights for solving reliability redundancy allocation problems. *Engineering Optimization, 45*, 1273–1286.
Walia, G. S., & Kapoor, R. (2014). Intelligent video target using an evolutionary particle filter based upon improved cuckoo search. *Expert Systems with Applications, 41*, 6315–6326.
Wang, H., Wang, W., Sun, H., Cui, Z., Rahnamayan, S., & Zeng, S. (2017). A new cuckoo search algorithm with hybrid strategies for flow shop scheduling problems. *Soft Computing, 21*, 4297–4307.
Yang, X.-S., & Deb, S. (2009). Cuckoo search via Levy flights. *Proceedings of the world congress on nature & biologically inspired computing (NaBIC 2009, India)* (pp. 210–214). USA: IEEE Publications.
Yang, X-S., & Deb, S. (2014). Cuckoo search: Recent advances and applications. *Neural Computations and Applications, 24*, 169–174.