Contents lists available at GrowingScience

International Journal of Industrial Engineering Computations

homepage: www.GrowingScience.com/ijiec

A discrete Jaya algorithm for permutation flow-shop scheduling problem

Aseem K. Mishra[a*] and Divya Shrivastava[a]

[a]*Department of Mechanical Engineering, Shiv Nadar University, NH 91 Tehsil Dadri, Gautam Buddha Nagar Uttar Pradesh 201314, India*

C H R O N I C L E

Article history:
Received October 8 2019
Received in Revised Format December 28 2019
Accepted December 31 2019
Available online January 2 2020

Keywords:
Jaya algorithm
Permutation flow-shop scheduling problem
Makespan minimization

A B S T R A C T

Jaya algorithm has recently been proposed, which is simple and efficient meta-heuristic optimization technique and has received a great attention in the world of optimization. It has been successfully applied to some thermal, design and manufacturing associated optimization problems. This paper aims to analyze the performance of Jaya algorithm for permutation flow-shop scheduling problem which is a well-known NP-hard optimization problem. The objective is to minimize the makespan. First, to make Jaya algorithm adaptive to the problem, a random priority is allocated to each job in a permutation sequence. Second, a job priority vector is converted into job permutation vector by means of Largest Order Value (LOV) rule. An exhaustive comparative study along with statistical analysis is performed by comparing the results with public benchmarks and other competitive heuristics. The key feature of Jaya algorithm of simultaneous movement towards the best solution and going away from the worst solution enables it to avoid being trapped in the local optima. Furthermore, the uniqueness of Jaya algorithm compared with any other evolutionary based optimization technique is that it is totally independent of specific parameters. This substantially reduces the computation effort and numerical complexity. Computational results reveal that Jaya algorithm is efficient in most cases and has considerable potential for permutation flow-shop scheduling problems.

1. Introduction

Permutation flow-shop scheduling problem (PFSP) has been proven to be the most popular non-deterministic-polynomial-time (NP)-hard with an extensive engineering relevance (Hejazi & Saghafian, 2005; Rinnooy Kan, 1976). A typical PFSP consists of each job i (i ϵ {1, 2, 3…, n}) to be processed on each machine j (j ϵ {1, 2, 3…, m}) in the same sequence of machines. Each machine can process only one job at a time. The objective is to determine an optimum production schedule which would satisfy the desired objective function under some given constraints (time horizon, limited resources) in a most efficient way. Since the PFSP has a great engineering background, many researchers have proposed several algorithms which can be broadly classified into three categories: exact, heuristic and meta-heuristic. Exact methods such as total enumeration, linear programming (Tseng & Stafford, 2008) and branch and bound (Madhushini & Rajendran, 2011) can obtain exact solutions. However, due to computational complexity, they cannot be applied for medium and large size problems. Heuristic

* Corresponding author
E-mail: am712@snu.edu.in (A. K. Mishra)

2020 Growing Science Ltd.
doi: 10.5267/j.ijiec.2019.12.001

algorithms (Nawaz et al., 1983) employ the problem-oriented specific knowledge and some constructive operations to solve the problems. They usually can obtain a nearly optimal solution in a reasonable computational time, while the solution qualities are not satisfactory. However, Nawaz–Enscore–Ham (NEH) (Nawaz et al., 1983) is one of the most effective heuristic and can provide comparable results with meta-heuristics. Meta-heuristic, on the other hand, starts from previously generated solutions and try to improve these solutions by using some strategies with domain-dependent knowledge. Some of the efficient meta-heuristics include genetic algorithm (GA) (Reeves & Yamada, 1998), simulated annealing algorithm (SA) (Osman & Potts, 1989) , particle swarm optimization algorithm (PSO) (Kuo et al., 2009), ant colony optimization (ACO) (Rajendran & Ziegler, 2004), artificial bee colony algorithm (ABC) (Pan et al., 2011), iterated greedy algorithm (IG) (Ruiz & Stützle, 2007), teaching learning-based optimization algorithm (TLBO) (Baykasoğlu et al., 2014) etc.

Recently it has been found out that mutation of two or more of these meta-heuristics known as hybrid heuristics can be more efficient than being applied in an isolated manner. As a consequence, a lot of hybrid heuristic based algorithms have been investigated by researchers in the past few years and reported that they can improve the performance especially when dealing with real-world and large-scale problems (Engin & Güçlü, 2018; Kuo et al., 2009; Lin et al., 2015; Ruiz et al., 2019; Zhang et al., 2019; Zhao et al., 2018). The performance of all types of approaches discussed above can be judged by three main factors: solution efficiency, computational efficiency, and ease of application. It should be noted that although hybrid heuristics produce most efficient results in less computation time, they become utter complicated due to their hybrid nature as the complexity vary dramatically according to their structure and parameters which in turn makes it problematic to understand and implement.

A new meta-heuristic optimization technique named as Jaya algorithm recently proposed by Venkata Rao (2016, 2019) is novel, simple and efficient algorithm to solve both constrained and unconstrained optimization problems. It has been successfully applied by many researchers in various fields of engineering and sciences (Rao, 2019). For instance, Rao and Saroj (2017) applied Jaya algorithm for the economic optimization of the shell-tube heat exchanger and compared the results with the earlier attempts of the same problem using GA, PSO and CSO (civilized swarm optimization). Computational results show the superiority of Jaya over other meta-heuristics. In the thermal area, Jaya finds another application in the optimization of the dimensions of a micro-channel heat sink under two objective functions viz. thermal resistance and pumping power (Rao et al., 2016). Experimental results show better performance when compared with TLBO and MOEA (multi-objective evolutionary algorithm). Gao et al. (2016) proposed Jaya algorithm for solving real-life case study of traffic light scheduling problem. Few improvement strategies and a feature based search operator is embedded to obtain the improved optimization results. Jaya showed better results as compared to harmony search (HS) and water cycle algorithm (WCA). Zhang et al. (2016) applied Jaya along with fractional Fourier entropy methodology for identification of tea category under sensitivity measurement.

Jaya also finds its applications in materials and manufacturing optimization processes. One such is the optimization of surface roughness in the surface grinding process (Rao et al., 2016). It is found that Jaya yields better results than reported by previous literature using GA, SA, PSO, ABC, ACO, HS and TLBO. A similar problem was addressed by Rao and Rai (2017) in the optimization of process parameters of the submerged arc welding process and results were compared with conventional meta-heuristics. Another application of Jaya exists in the optimization of control parameters for machining (turning) of carbon fiber-reinforced polymer (CFRP) (Abhishek et al., 2017). Jaya algorithm was combined with non-linear regression modeling and fuzzy inference system to optimize the process control parameters including feed rate, spindle speed and depth of cut. These parameters were optimized under three objectives viz. material removal rate (MRR), the average surface roughness (R_a) and net cutting force. Computational results revealed that Jaya has considerable potential in the context of machining performance optimization problems. Few applications of Jaya can also be found in design related problems (Du et al. 2017; Rao & More, 2017). Moreover, Jaya has also received successful applications in the optimization

of power transmission and electrical related problems. Singh et al. (Singh et al., 2017) performed design optimization of proportional-integral-derivative (PID) controller for automatic generation control (AGC) using Jaya algorithm. The superiority of Jaya is justified by comparing with particle swarm optimization (PSO), teaching-learning based optimization (TLBO) and differential evolution (DE) algorithms. Huang et al. (2017) applied Jaya algorithm for solving maximum power point tracking (MPPT) for photovoltaic (PV) systems by incorporating natural cubic spline based prediction model (S-Jaya). Simulation results proved the faster convergence of Jaya algorithm for providing higher tracking efficiency.

Based on the review of publically available literature, it seems that Jaya has potential in solving engineering design and non-linear optimization problems. Yet it appears from the literature that it has very limited applications to scheduling problems such as flow shops and job shops. For instance Gao et al. (2016) proposed Jaya algorithm for solving flexible job shop scheduling problems with an objective to minimize the maximum machine workload. The superiority of Jaya is presented by comparing with existing methods. A similar problem was addressed by Guo et al. (2017) considering flexible job-shop rescheduling problem. Buddala and Mahapatra (2017) applied TLBO and Jaya algorithm for optimization of flexible flow shop scheduling problem with objective of makespan minimization. Computational results validate the efficiency of Jaya as compared with other meta-heuristics. Radhika et al. (2016) applied Jaya algorithm for optimization of master production scheduling problem with multi-objective. Jaya produced better results when compared with conventional techniques such as genetic algorithm (GA) and differential evolution (DE). Recently, Mishra and Shrivastava (2018) applied TLBO and Jaya algorithm for flow shop scheduling problem with an objective to minimize the sum of work-in-process (WIP) inventory holding and tardiness costs. Computational results reveal the effectiveness when compared with the existing heuristics. However, the results were not compared with public benchmarks.

It is a well-known fact that apart from design and non-linear optimization problems most of the developed meta-heuristics are supposed to provide effective solutions for complex combinatorial optimization problems as well especially when dealing with flow shop scheduling problems. Secondly, the specialty of Jaya algorithm is that unlike any other optimization technique such as GA, SA and PSO it is a parameterless algorithm (free from algorithm-specific parameters) and contain only two common control parameters (population size and a number of generations) which make it very easy to understand and implement. Based on this motivation, we apply Jaya algorithm to permutation flow-shop scheduling problem. The performance of Jaya algorithm on flow shop can also give an idea for its possible scope in solving other scheduling problems. The objective is to minimize the maximum completion time i.e. makespan. In order to make Jaya adaptive to solve flow shop, a random priority is assigned to each job in permutation sequence. Then the largest order value (LOV) rule is utilized to convert job priority vector to job permutation vector. Jaya algorithm is compared with many benchmark problems and efficient heuristics available in the literature. The methodology, application and effectiveness of Jaya algorithm for PFSP are explained in the proceeding sections.

The rest of the paper is organized as follows: In section 2 we describe the objective function and the implementation of Jaya algorithm to PFSP with an illustrative example. Section 3 shows the parameter estimation, comparison of results and discussions followed by section 4 with conclusions and future extensions.

2. Jaya algorithm and objective function

2.1 A brief introduction of Jaya algorithm

Jaya algorithm is based on the principle that the solution to the given problem should move towards the best-known solution and goes away from the worst solution. The steps involved in the application of Jaya algorithm are briefly summarized as follows:-

- Initialize number of design variables, population size and termination criterion

- Identify the worst and the best solution in the population
- Modify the design variable of other solutions based on the best and the worst solution according to equation (1)
- Compare the updated solution from the previous solution. If the updated solution is better, replace it otherwise keep the older one.
- Report the optimum solution

$$x'_{i,k,l} = x_{i,k,l} + r_{1,i,l} * (x_{i,k,lbest} - |x_{i,k,l}|) - r_{2,i,l} * (x_{i,k,lworst} - |x_{i,k,l}|) \quad (1)$$

where,

$x_{i,k,l}$	the value of an i^{th} variable in k^{th} population during l^{th} iteration
$x_{i,k,lbest}$	the value of an i^{th} variable in the population having the best solution
$x_{i,k,lworst}$	the value of an i^{th} variable in the population having the worst solution
$x'_{i,k,l}$	updated value of $x_{i,k,l}$
$r_{1,i,l}$, $r_{2,i,l}$	random numbers for the i^{th} variable during the l^{th} iteration for best and worst solutions respectively in the range [0, 1]

2.2 Description of the objective function for PFSP

A typical flow-shop scheduling problem consists of job $i \in \{1,2,3 \dots, n\}$ to be processed on machine $j \in \{1,2,3 \dots, m\}$. Each job must visit each machine in the same sequence. The objective is to obtain an optimum production schedule of jobs which would minimize the total completion time (makespan). Let $C_{i,j,k}$ be the completion time of job i on machine j in schedule k. Therefore the completion time can be defined as:

$$C_{1,j,k} = p_{1,j,k} + p_{1,j-1,k} \quad (1)$$
$$C_{i,1,k} = p_{i,1,k} + p_{i-1,1,k} \quad (2)$$
$$C_{i,j,k} = p_{i,j,k,} + max\{ C_{i,j-1,k} + C_{i-1,j,k}\} \quad (3)$$
$$\pi_k = C_{n,m,k} \quad (4)$$
$$\pi^* = min\{ \pi_1, \pi_2, \pi_3, \dots, \pi_l\} \quad (5)$$

In the above equations, $p_{i,j,k,}$ represents the processing time of job i on machine j in schedule k. π_k is the maximum completion time (makespan) of k^{th} schedule and π^* represents the schedule with minimum makespan.

2.3 Implementation of Jaya algorithm to PFSP

The implementation of Jaya algorithm for permutation flow-shop scheduling problem (*PFSP*) is described in the following steps.

Step 1: - Initialize population size i.e. number of schedules (*NP*), decision variables i.e. number of jobs (n), and termination criterion i.e. generation number (*GEN*)

Step 2: - Create a job priority vector, $\Psi_{k,l} = \{ \psi_{1,k,l} , \psi_{2,k,l} , \dots , \psi_{i,k,l} , \dots , \psi_{n,k,l}\}$, where $\Psi_{k,l}$ is an n-dimensional vector which represents the sequence of jobs in the k^{th} schedule at l^{th} iteration and $\psi_{i,k,l}$ is the priority assigned to an i^{th} job in the k^{th} schedule at l^{th} iteration. The priority is randomly generated with uniform random number distribution as per the Eq. (7).

$$\psi_{i,k,l}=1+rand(0,1)\times(n-1) \quad (7)$$

Step 3:- Convert the job priority vector ($\Psi_{k,l}$) into job permutation vector ($\Theta_{k,l}$) by Largest Order Value (LOV) rule, i.e. jobs are arranged in non-increasing order of their prioritys ($\psi_{i,k,l}$). Therefore $\Theta_{k,l} = \{\theta_{j,1,k,l}, \theta_{j,2,k,l}, \ldots, \theta_{j,i,k,l}, \ldots, \theta_{j,n,k,l}\}$, where $\theta_{j,i,k,l}$ is j^{th} job placed at i^{th} position in k^{th} schedule at l^{th} iteration.

Step 4:- Calculate the makespan ($\pi_{max,k,l}$) for each permutation sequence ($\Theta_{k,l}$). Compare the $\pi_{max,k,l}$ value of each schedule with the corresponding $\pi_{max,k,l-1}$ value obtained in the previous iteration and update the better solution (In case of the very first iteration, directly go to step 5).

Step 5: Identify the schedule with minimum makespan ($\Theta_{k,l,min}$) and maximum makespan ($\Theta_{k,l,max}$) and their corresponding priority vectors ($\Psi_{k,lmin}$ & $\Psi_{k,lmax}$) respectively.

Step 6: - Update the prioritys of all priority vectors based on the prioritys of $\Psi_{k,lmin}$ & $\Psi_{k,lmax}$ as per Eq. (8)

$$\psi'_{i,k,l}= \psi_{i,k,l}+ r_{1,i,l} * (\psi_{i,k,lmin} - |\psi_{i,k,l}|) - r_{2,i,l} * (\psi_{i,k,lmax} - |\psi_{i,k,l}|) \quad (8)$$

where,

$\psi_{i,k,l}$	the priority of i^{th} job in k^{th} schedule during l^{th} iteration
$\psi_{i,k,lmin}$	the priority of the i^{th} job in the schedule having minimum makespan ($\pi_{max,k,lmin}$)
$\psi_{i,k,lmax}$	the priority of the i^{th} job in the schedule having maximum makespan ($\pi_{max,k,lmax}$)
$\psi'_{i,k,l}$	updated value of $\psi_{i,k,l}$
$r_{1,i,l}$, $r_{2,i,l}$	random numbers for the i^{th} job during the l^{th} iteration for minimum and maximum solutions respectively in the range [0, 1]

Step 7: Convert the updated priority vector into job permutation vector and identify the makespan of the new schedules obtained (as per step 3 and 4).

Step 8: - Compare the makespan values of new schedule with the corresponding previous schedule and update the better solution (replace or retain the schedule with minimum makespan).

Step 9: - Repeat the above steps until the termination criteria are satisfied.

Step 10: - Report the optimum solution (optimum schedule with minimum makespan)

$i \in \{1,2,3,\ldots,n\}, k \in \{1,2,3,\ldots NP\}, l \in \{1,2,3,\ldots,GEN\}$

The flow chart of Jaya algorithm for *PFSP* is depicted in Fig. 1.

2.4 Illustrative example

Let us consider an eight job three machine problem. The processing times of jobs are given in Table 1. Let the size of the population be 5. The solution steps for calculating makespan is represented in Fig. 2. In step 1 the initial job priority vector is created using equation 7. Now each priority vector is arranged in a non-increasing order of their priorities and the corresponding job permutation vector **($\Theta_{k,l}$)** is retrieved as shown in steps 2 and 3. For each job permutation sequence, makespan ($\pi_{max,k,l}$) is calculated. Step 4 identifies the schedules with minimum and maximum makespans ($\Theta_{k,l,min}$ & $\Theta_{k,l,max}$) and corresponding job priority vectors ($\Psi_{k,lmin}$ & $\Psi_{k,lmax}$) .

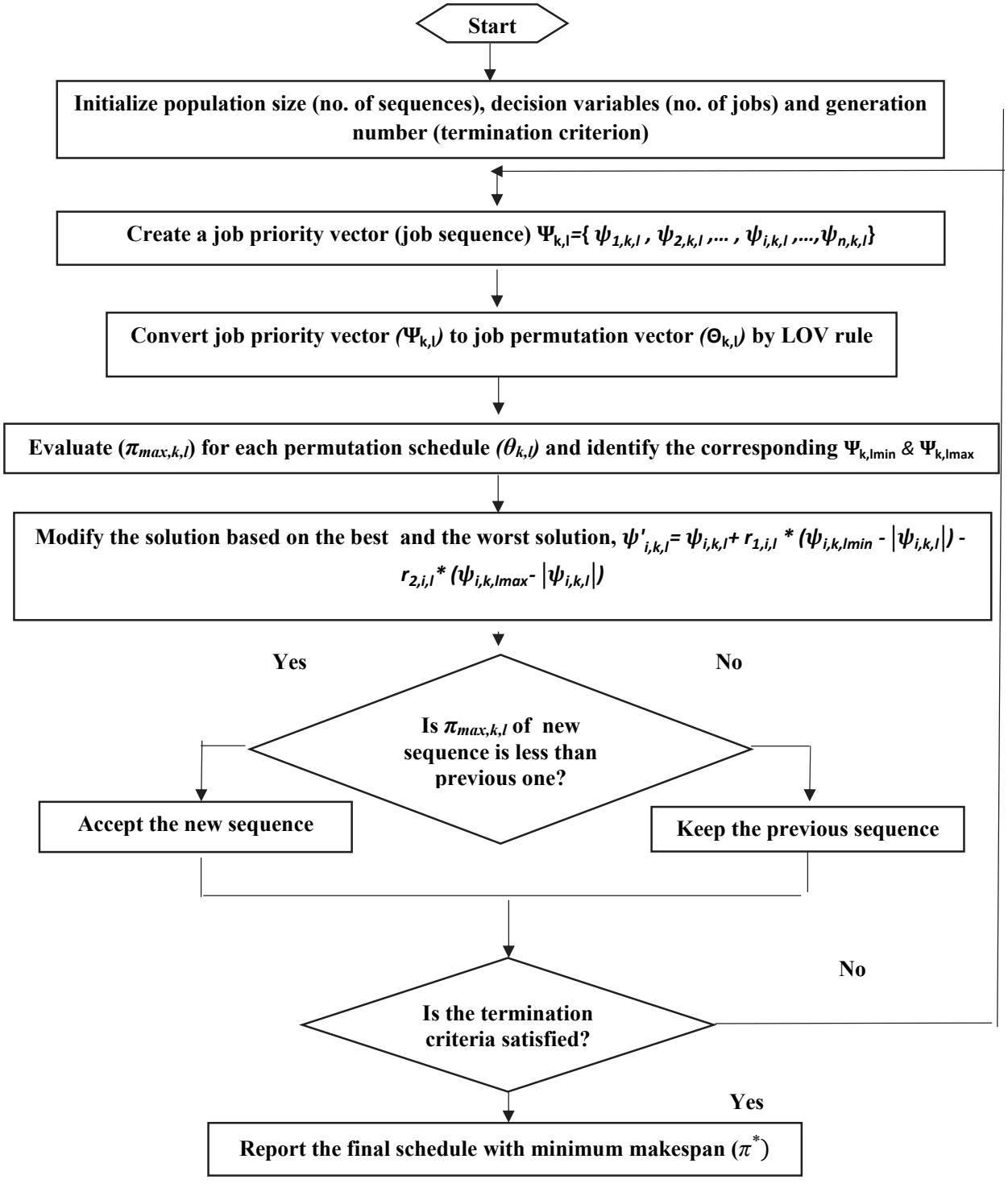


Fig. 1. Flowchart of Jaya algorithm for solving *PFSP*

Table 1
Processing times of jobs

Jobs/ Machines	Processing time of job (in min)							
	J1	J2	J3	J4	J5	J6	J7	J8
M1	5	74	67	97	87	10	69	69
M2	76	21	48	36	86	42	32	12
M3	74	83	6	71	64	20	99	54

As shown in Fig. 2, sequences 1 and 2 have minimum and maximum makespan respectively. Therefore, the priorities of priority vectors 1 and 2 will be assigned as $\psi_{i,k,lmin}$ and $\psi_{i,k,lmax}$ respectively. Now the priorities of each priority vector are updated (Step 4) using equation 8 and the corresponding new job permutation vector is obtained (Step 5). The makespan of each schedule in new job permutation vector is compared with the corresponding value in old one and the better solution is retained (Step 6). This completes one iteration. In the next iteration, compare the current solution with the previous solution and update the better solutions accordingly. After the termination criteria is satisfied, the near-optimal solution is reached.

3. Computational results and comparisons

In this section, we determine the estimation of parameters and evaluate the performance of Jaya algorithm by comparing the results with some classical and some efficient heuristics for the *PFSP* problem. Moreover, Jaya is coded in Matlab 8.6.0 and experimental results are analyzed on a 3.25 GHz i5-4570 processor. The performance of Jaya algorithm, is evaluated with three standard benchmarks viz. Carlier's benchmarks (Carlier, 1978) , Reeves and Yamada's benchmarks (Reeves & Yamada, 1998) and Taillard's benchmarks (Taillard, 1990) . These benchmarks have been widely used by many researchers to demonstrate the performance of *PFSSP* problems. Secondly, perhaps Jaya algorithm is not yet applied for *PFSP* in an isolated or any hybridized form, therefore we attempt to select some classical algorithms and some recent heuristics to evaluate our results. Moreover, we do not recalculate or reproduce the results obtained by other algorithms and accept the actual results from the literature.

In order to measure the effectiveness and accuracy of Jaya algorithm, ten independent runs are carried out for each problem set and three performance measures as shown in equations (9)-(11) are calculated. S^* represents the best solution obtained by the aforementioned benchmarks, S_{best} and S_{worst} denotes the best and the worst solutions respectively obtained by the algorithms. S_i denotes the solution obtained at an i^{th} run. *BRE* is the best percentage relative error to S^*, *ARE* is the average percentage relative error to S^*, *WRE* represents the worst percentage relative error to S^* and k is the number of runs.

$$BRE = \frac{(S_{best} - S^*)}{S^*} \times 100 \tag{9}$$

$$ARE = \left(\sum_{i=1}^{k} \frac{(S_i - S^*)}{S^*} \times 100\right)/k \tag{10}$$

$$WRE = \frac{(S_{worst} - S^*)}{S^*} \times 100 \tag{11}$$

3.1 Estimation of parameters

Jaya algorithm, as aforementioned, consists of only two common control parameters: population size (*NP*) and generation number (*GEN*). The value range of these parameters are set as:

$NP \in \{50,100,150,200 \text{ and } 250\}$, $GEN \in \{500, 1000, 1500, 2000, \text{and } 2500\}$. Hence, a total of 25 combinations of {*NP*, *GEN*} are taken and for each set, 20 independent runs are carried out.

To evaluate the experimental results, two problem sizes of 50 jobs × 20 machines and 100 jobs × 20 machines are taken from Taillard's problem set (Taillard, 1990) and experimental results are analyzed by Analysis of Variance (ANOVA) method. The results were found significant with p-value < 0.05 (95% confidence interval) for both the parameters. Fig. 3 and Fig. 4 show the mean plots of *ARE* values for *NP* and *GEN* for the two problem sizes respectively. In Fig. 3 and Fig. 4 it is seen that for both the problem sizes, the optimum possible set could be {*NP*, *GEN*} = {200, 1500}.

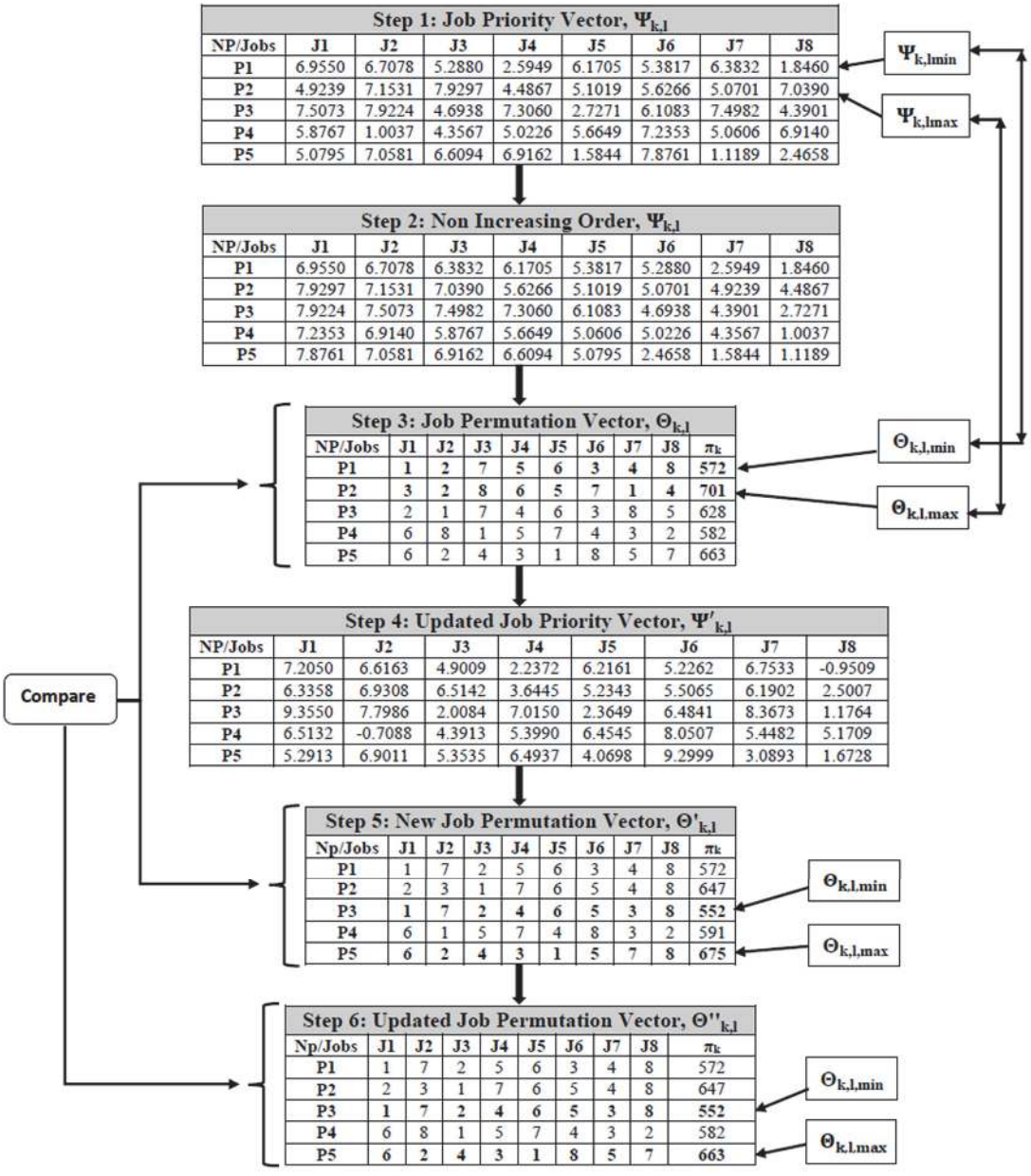
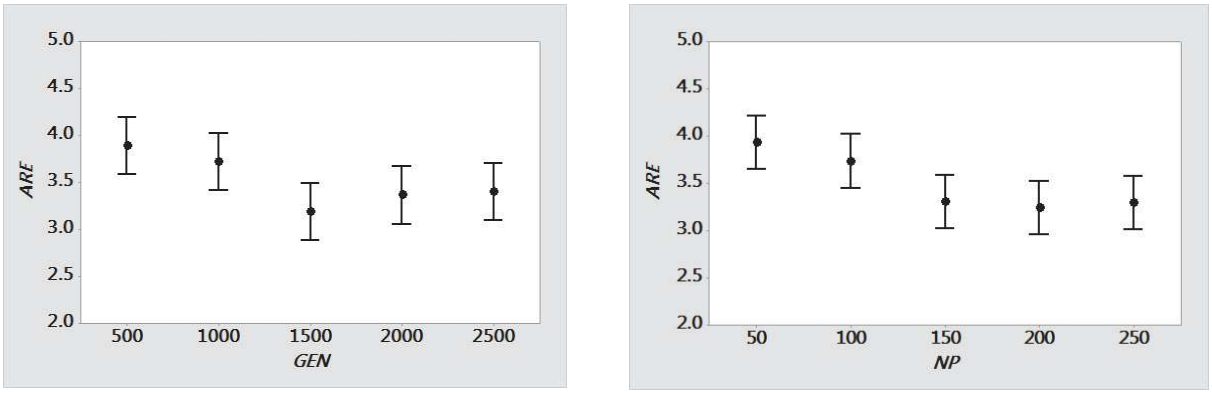


Fig. 2. An example to demonstrate the application of Jaya algorithm for *PFSP*

3.2 Comparisons with various algorithms

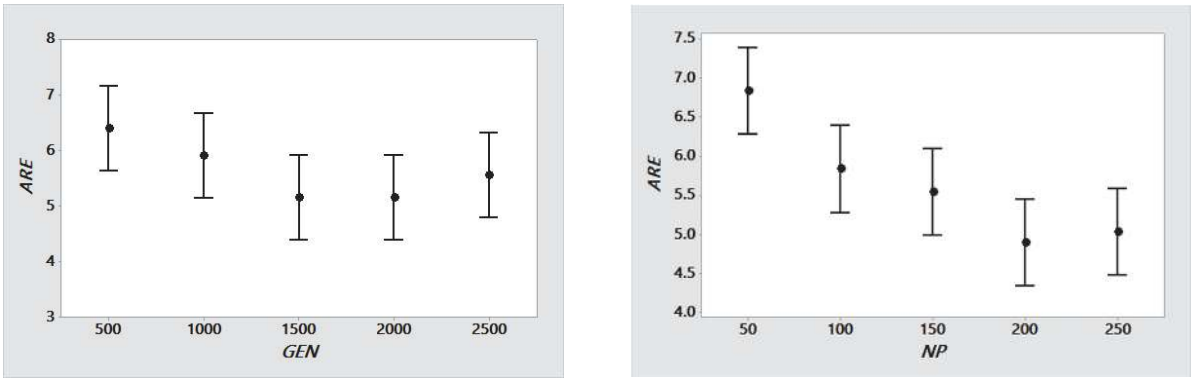
In this section, we evaluate the performance of Jaya algorithm with Carlier's benchmark set (Carlier, 1978) and Reeves and Yamada's benchmark set (Reeves & Yamada, 1998). The efficiency of Jaya algorithm is compared with the following efficient meta-heuristics available in literature:

- Hybrid Genetic Algorithm (HGA) by (Wang & Zheng, 2003).
- Hybrid Differential Evolution (HDE) by (Qian et al., 2008).
- Hybrid Particle Swarm Optimization (HSPO) by (Liu, Wang, & Jin, 2008).
- Teaching Learning Based Optimization (TLBO) by (Baykasoğlu et al., 2014).
- Hybrid Backtracking Search Algorithm (HBSA) by (Lin et al., 2015).



(a) (b)

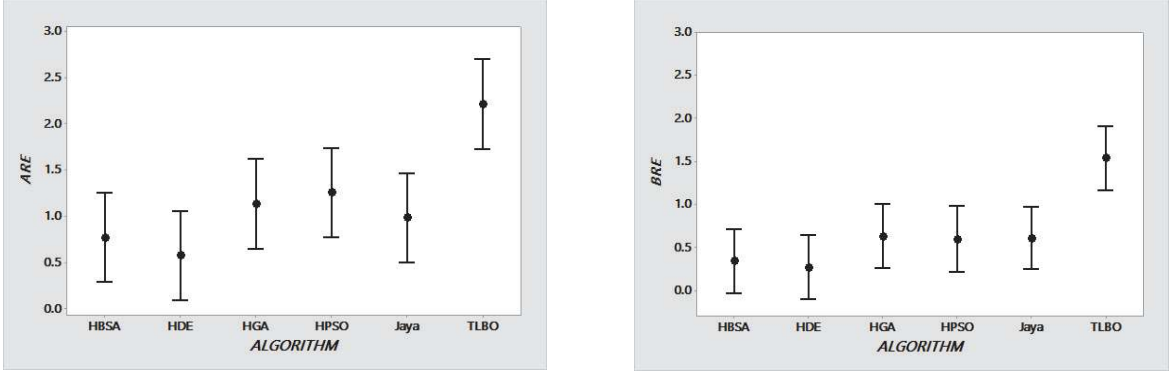
Fig. 3. *ARE* mean plots of parameters for 50×20 problem size (a) *GEN* factor (b) *NP* factor (95% confidence interval)



(a) (b)

Fig. 4. ARE mean plots of parameters for 100×20 problem size (a) GEN factor (b) NP factor (95% confidence interval)

The *BRE, ARE* and *WRE* values of all these algorithms shown in Table 2 and Table 3. From tables 2 and 3 it can be seen that Jaya algorithm has produced efficient results for all these benchmarks. Fig. 5 depicts the mean plots of *ARE* and *BRE* values of various algorithms under 95% confidence interval. In Fig. 5(a) we see that in terms of *ARE,* Jaya outperforms TLBO, HPSO and HGA and fairly comparable with HDE and HBSA. Similarly, the *BRE* value (Fig. 5(b)) of Jaya is less than TLBO and equally proportionate with other algorithms. This gives us an idea Jaya seems to be efficient and have considerable potential to solve permutation flow shop scheduling problem.



(a) (b)

Fig 5. Mean plots of performance measures for various algorithms (a) *ARE* values (b) *BRE* values (95% confidence interval)

Table 2

Comparisons of HGA, HDE, HPSO and Jaya. **Bold** indicate the best solution obtained among various algorithms

Instance	$n \times m$	S^*	HGA			HDE			HPSO		Jaya		
			BRE	ARE	WRE	BRE	ARE	WRE	BRE	ARE	BRE	ARE	WRE
Car1	11×5	7038	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Car2	13×4	7166	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Car3	12×5	7312	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.79	**0**	**0**	**0**
Car4	14×4	8003	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Car5	10×6	7720	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.59	**0**	0.106	0.233
Car6	8×9	8505	**0**	0.04	0.76	**0**	**0**	**0**	**0**	0.61	**0**	**0**	**0**
Car7	7×7	6590	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Car8	8×8	8366	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.03	**0**	**0**	**0**
Rec01	20×5	1247	**0**	0.14	0.16	**0**	0.144	0.160	**0**	0.41	**0**	0.111	0.261
Rec03	20×5	1109	**0**	0.09	0.18	**0**	**0**	**0**	0.18	0.30	**0**	0.123	0.382
Rec05	20×5	1242	**0**	0.29	1.13	0.242	0.242	0.242	0.24	0.29	0.142	0.195	0.425
Rec07	20×10	1566	**0**	0.69	1.15	**0**	0.230	1.149	0.70	1.66	**0**	0.881	1.213
Rec09	20×10	1537	**0**	0.64	2.41	**0**	**0**	**0**	**0**	1.54	**0**	0.678	1.431
Rec11	20×10	1431	**0**	1.10	2.59	**0**	**0**	**0**	**0**	1.20	**0**	0.743	1.607
Rec13	20×15	1930	0.36	1.68	3.06	0.104	0.301	0.518	0.21	1.25	0.207	1.026	2.383
Rec15	20×15	1950	0.56	1.12	2.00	**0**	0.308	0.923	0.67	1.36	0.41	1.062	2.615
Rec17	20×15	1902	0.95	2.32	3.73	**0**	1.178	2.471	**0**	2.33	0.894	1.378	2.629
Rec19	30×10	2093	0.62	1.32	2.25	**0.287**	0.559	0.860	0.67	1.35	0.956	1.358	2.293
Rec21	30×10	2017	1.44	1.57	1.64	**0.198**	1.413	1.636	1.44	1.61	1.286	1.406	4.958
Rec23	30×10	2011	**0.40**	0.87	1.69	0.448	0.482	0.497	0.90	1.84	1.442	2.083	4.724
Rec25	30×15	2513	1.27	2.54	3.98	0.478	1.492	2.308	1.11	2.42	1.592	2.207	3.82
Rec27	30×15	2373	1.10	1.83	4.00	0.843	1.285	2.191	0.55	1.83	1.559	1.717	3.245
Rec29	30×15	2287	1.40	1.70	4.20	**0.306**	0.791	1.443	1.01	3.05	1.618	1.983	3.585
Rec31	50×10	3045	0.43	1.34	2.5	0.296	0.824	1.839	1.38	2.34	1.905	2.286	3.021
Rec33	50×10	3114	**0**	0.78	0.83	**0**	0.434	0.835	**0**	0.78	**0**	0.257	1.445
Rec35	50×10	3277	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0.01	**0**	0.108	0.593
Rec37	75×20	4951	3.75	4.90	6.18	1.818	2.727	3.878	2.26	3.03	2.686	3.254	4.141
Rec39	75×20	5087	2.20	2.79	4.48	**0.983**	1.541	1.985	1.47	2.11	1.868	2.227	3.126
Rec41	75×20	4960	3.64	4.92	5.91	1.673	2.629	3.306	2.74	3.48	2.548	3.127	5.141

Table 3

Comparisons of HBSA, TLBO and Jaya. **Bold** indicate the best solution obtained among various algorithms

Instance	$n \times m$	S^*	HBSA			TLBO		Jaya		
			BRE	ARE	WRE	BRE	ARE	BRE	ARE	WRE
Car1	11×5	7038	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Car2	13×4	7166	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Car3	12×5	7312	**0**	0.060	1.190	**0**	0.324	**0**	**0**	**0**
Car4	14×4	8003	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Car5	10×6	7720	**0**	**0**	**0**	**0**	0.593	**0**	0.106	0.233
Car6	8×9	8505	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Car7	7×7	6590	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Car8	8×8	8366	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**
Rec01	20×5	1247	**0**	0.14	0.16	**0**	0.160	**0**	0.111	0.261
Rec03	20×5	1109	**0**	0.08	0.18	**0**	0	**0**	0.123	0.382
Rec05	20×5	1242	0.24	0.24	0.24	0.242	0.242	0.142	0.195	0.425
Rec07	20×10	1566	**0**	0.46	1.15	0.325	0.911	**0**	0.881	1.213
Rec09	20×10	1537	**0**	0.07	0.65	0.978	1.607	**0**	0.678	1.431
Rec11	20×10	1431	**0**	**0**	**0**	1.327	1.887	**0**	0.743	1.607
Rec13	20×15	1930	0.10	0.53	1.14	0.725	1.347	0.207	1.026	2.383
Rec15	20×15	1950	0.05	0.64	1.18	0.872	2.205	0.41	1.062	2.615
Rec17	20×15	1902	**0**	0.1	2.16	2.050	3.785	0.894	1.378	2.629
Rec19	30×10	2093	**0**	1.00	2.16	2.102	3.440	0.956	1.358	2.293
Rec21	30×10	2017	0.29	0.81	1.29	1.636	2.677	1.286	1.406	4.958
Rec23	30×10	2011	0.69	1.50	2.83	1.542	2.984	1.442	2.083	4.724
Rec25	30×15	2513	0.94	1.95	3.08	3.104	4.497	1.592	2.207	3.82
Rec27	30×15	2373	1.47	2.54	3.78	2.950	4.130	1.559	1.717	3.245
Rec29	30×15	2287	1.01	1.96	2.91	4.416	6.209	1.618	1.983	3.585
Rec31	50×10	3045	0.43	1.91	2.66	4.696	5.813	1.905	2.286	3.021
Rec33	50×10	3114	**0**	0.59	1.28	2.055	3.308	**0**	0.257	1.445
Rec35	50×10	3277	**0**	**0**	**0**	0.335	0.458	**0**	0.108	0.593
Rec37	75×20	4951	1.92	2.93	4.20	5.777	6.461	2.686	3.254	4.141
Rec39	75×20	5087	0.90	1.88	3.38	4.030	4.993	1.868	2.227	3.126
Rec41	75×20	4960	1.69	2.72	3.55	5.161	5.907	2.548	3.127	5.141

3.3 Comparisons with Taillard's benchmarks

The performance of Jaya is also compared with well-known Taillard's benchmarks (Taillard, 1990) . Hybrid Particle Swarm Optimization (HPSO) by (Kuo et al., 2009) and TLBO by (Baykasoğlu et al., 2014) are considered for comparing results with Jaya algorithm. Table 4 shows the minimum, maximum and average values of makespan obtained by various algorithms. Table 5 presents the *ARE* values obtained by HPSO, TLBO and Jaya. As we can see that the ARE values of Jaya are better than TLBO and comparable with HPSO. Above all, we come to our final outcome that for all three well-known benchmarks, the results obtained by Jaya algorithm for permutation flow-shop scheduling problem seems plausible when compared with other efficient heuristics and hence this algorithm can be considered as amenable and exhaustive to discrete combinatorial optimization problems.

Table 4
Comparisons of HPSO, TLBO and Jaya. **Bold** indicate the best solution obtained among various algorithms

Instance	$n \times m$	S^*	HPSO			TLBO			Jaya		
			Min	Max	Avg.	Min	Max	Avg.	Min	Max	Avg.
Ta001	20×5	1278	**1278**	**1278**	**1278**	**1278**	1297	1287.2	**1278**	1285	1281.5
Ta011	20×10	1582	**1582**	1596	1587.3	1586	1618	1606	1584	1623	1609
Ta021	20×20	2297	**2297**	2315	2307	2325	2370	2345.7	2311	2347	2338.6
Ta031	50×5	2724	**2724**	**2724**	**2724**	**2724**	2741	2729.4	**2724**	2736	2729
Ta041	50×10	2991	3034	3063	3053.6	3120	3169	3141	3060	3108	3088.1
Ta051	50×20	3771	3923	3966	3944.3	3986	4095	4029.7	3981	4082	4052.5
Ta061	100×5	5493	**5493**	**5493**	**5493**	**5493**	5527	5499.4	**5493**	5505	5500.3
Ta071	100×10	5770	None	None	None	5887	5997	5928.7	5850	5964	5938.1
Ta081	100×20	6286	None	None	None	6549	6726	6617.8	6470	6609	6571.6
Ta091	200×10	10868	None	None	None	10979	11079	11033	11094	11165	11135
Ta101	200×20	11294	None	None	None	11855	12024	11940	12079	12146	12119.4
Ta111	500×20	26189	None	None	None	27377	27565	27492	27937	28073	28000.5

Table 5
Comparisons of *ARE* values of HPSO, TLBO and Jaya. **Bold** indicate the best solution obtained among various algorithms

Instance	Jaya	TLBO	HPSO
	ARE	ARE	ARE
Ta001	0.274	0.72	**0**
Ta011	1.707	1.52	**0.335**
Ta021	1.811	2.08	**0.435**
Ta031	0.184	0.20	**0**
Ta041	**2.086**	5.03	2.093
Ta051	**4.604**	6.86	**4.604**
Ta061	0.133	0.12	**0**
Ta071	2.913	2.75	None
Ta081	4.543	5.28	None
Ta091	2.457	1.52	None
Ta101	7.308	5.72	None
Ta111	6.917	4.98	None

5. Conclusions and future extensions

In the present study, we test the performance of recently proposed, simple and efficient meta-heuristic optimization technique named Jaya algorithm for permutation flow-shop scheduling problem for the first time in literature. The uniqueness of the proposed algorithm as against other common meta-heuristics is that it does not require turning of algorithm-specific parameters. The objective is to minimize the makespan (maximum completion time). To obtain a job permutation vector largest order value rule (LOV) is utilized. The results are analyzed on well-known public benchmarks and compared with the efficient heuristics available in the literature. Computational results reveal the effectiveness of Jaya algorithm. An extensive experimental work is carried out in order to explore the potential of Jaya algorithm for solving permutation flow shop scheduling problems. It has been investigated that performance of Jaya on flow shop scheduling problems can give an idea about its possible applications to other discrete combinatorial problems. Further research in the proposed study can be extended in the

following directions. Firstly due to simplicity in its application, Jaya algorithm can be hybridized or modified with other algorithms in order to achieve more efficient results. Secondly, it can also be applied to other complex shop floor problems such as parallel flow-shop, flexible flow shop, open shop, two-stage flow shop etc. considering maintenance and quality effects.

References

Abhishek, K., Kumar, V. R., Datta, S., & Mahapatra, S. S. (2017). Application of JAYA algorithm for the optimization of machining performance characteristics during the turning of CFRP (epoxy) composites: comparison with TLBO, GA, and ICA. *Engineering with Computers*, *33*(3), 457–475.

Baykasoğlu, A., Hamzadayi, A., & Köse, S. Y. (2014). Testing the performance of teaching-learning based optimization (TLBO) algorithm on combinatorial problems: Flow shop and job shop scheduling cases. *Information Sciences*, *276*, 204–218.

Buddala, R., & Mahapatra, S. S. (2017). Improved teaching–learning-based and JAYA optimization algorithms for solving flexible flow shop scheduling problems. *Journal of Industrial Engineering International*, 1–16.

Carlier, J. (1978). Ordonnancements a contraintes disjonctives. *RAIRO-Operations Research*.

Du, D.-C., Vinh, H.-H., Trung, V.-D., Hong Quyen, N.-T., & Trung, N.-T. (2017). Efficiency of Jaya algorithm for solving the optimization-based structural damage identification problem based on a hybrid objective function. *Engineering Optimization*, *273*(September), 1–19.

Engin, O., & Güçlü, A. (2018). A new hybrid ant colony optimization algorithm for solving the no-wait flow shop scheduling problems. *Applied Soft Computing*, *72*, 166–176.

Gao, K., Sadollah, A., Zhang, Y., & Su, R. (2016). Discrete Jaya Algorithm for Flexible Job Shop Scheduling Problem with New Job Insertion. *14th International Conference on Control, Automation, Robotics & Vision*, *2016*(61603169), 13–15.

Gao, K., Zhang, Y., Sadollah, A., Lentzakis, A., & Su, R. (2016). Jaya, harmony search and water cycle algorithms for solving large-scale real-life urban traffic light scheduling problem. *Swarm and Evolutionary Computation*, (May), 1–15.

Guo, J., Gao, K., Wang, C., Sang, H., Li, J., & Duan, P. (2017). Discrete Jaya algorithm for solving flexible job shop rescheduling problem. *2017 29th Chinese Control And Decision Conference (CCDC)*, 6010–6015.

Huang, C., Wang, L., Yeung, R. S. cheung, Zhang, Z., Chung, H. S. H., & Bensoussan, A. (2017). A Prediction Model Guided Jaya Algorithm for the PV System Maximum Power Point Tracking. *IEEE Transactions on Sustainable Energy*, *3029*(c).

Kuo, I.-H., Horng, S.-J., Kao, T.-W., Lin, T.-L., Lee, C.-L., Terano, T., & Pan, Y. (2009). An efficient flow-shop scheduling algorithm based on a hybrid particle swarm optimization model. *Expert Systems with Applications*, *36*(3), 7027–7032.

Lin, Q., Gao, L., Li, X., & Zhang, C. (2015). A hybrid backtracking search algorithm for permutation flow-shop scheduling problem. *Computers & Industrial Engineering*, *85*, 437–446.

Liu, B., Wang, L., & Jin, Y. H. (2008). An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. *Computers and Operations Research*, *35*(9), 2791–2806.

Madhushini, N., & Rajendran, C. (2011). Branch-and-bound algorithms for scheduling in an m-machine permutation flowshop with a single objective and with multiple objectives. *European J. of Industrial Engineering*, *5*(4), 361.

Mishra, A., & Shrivastava, D. (2018). A TLBO and a Jaya heuristics for permutation flow shop scheduling to minimize the sum of inventory holding and batch delay costs. *Computers & Industrial Engineering*, *124*(July), 509–522.

Nawaz, M., Enscore, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega*, *11*(1), 91–95.

Osman, I., & Potts, C. (1989). Simulated annealing for permutation flow-shop scheduling. *Omega*, *17*(6), 551–557.

Pan, Q.-K., Tasgetiren, M. F., Suganthan, P. N., & Chua, T. J. (2011). A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. *Information Sciences*, *181*(12), 2455–2468.

Qian, B., Wang, L., Hu, R., Wang, W. L., Huang, D. X., & Wang, X. (2008). A hybrid differential evolution method for permutation flow-shop scheduling. *International Journal of Advanced Manufacturing Technology*, *38*(7–8), 757–777.

Radhika, S., Ch, S. R., D, N. K., & K, K. P. (2016). *Multi-Objective Optimization of Master Production Scheduling Problems using Jaya Algorithm*. (December), 1729–1732.

Rajendran, C., & Ziegler, H. (2004). Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. *European Journal of Operational Research*, *155*(2), 426–438.

Rao, R. V., & More, K. C. (2017). Design optimization and analysis of selected thermal devices using self-adaptive Jaya algorithm. *Energy Conversion and Management*, *140*, 24–35.

Rao, R. V., More, K. C., Taler, J., & Ocłoń, P. (2016). Dimensional optimization of a micro-channel heat sink using Jaya algorithm. *Applied Thermal Engineering*, *103*, 572–582.

Rao, R. V., & Rai, D. P. (2017). Optimization of submerged arc welding process parameters using quasi-oppositional based Jaya algorithm. *Journal of Mechanical Science and Technology*, *31*(5), 2513–2522.

Rao, R. V., Rai, D. P., & Balic, J. (2016). *Surface Grinding Process Optimization Using Jaya Algorithm*. https://doi.org/10.1007/978-81-322-2731-1_46

Rao, R. V., & Saroj, A. (2017). Economic optimization of shell-and-tube heat exchanger using Jaya algorithm with maintenance consideration. *Applied Thermal Engineering*, *116*, 473–487.

Reeves, C. R., & Yamada, T. (1998). Genetic Algorithms, Path Relinking, and the Flowshop Sequencing Problem. *Evolutionary Computation*, *6*(1), 45–60. https://doi.org/10.1162/evco.1998.6.1.45

Reza Hejazi, S., & Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: a review. *International Journal of Production Research*, *43*(14), 2895–2929.

Rinnooy Kan, A. H. G. (1976). *Machine Scheduling Problems : Classification, complexity and computations*. Springer US.

Ruiz, R., Pan, Q.-K., & Naderi, B. (2019). Iterated Greedy methods for the distributed permutation flowshop scheduling problem. *Omega*, *83*, 213–222.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, *177*(3), 2033–2049.

Singh, S. P., Prakash, T., Singh, V. P., & Babu, M. G. (2017). Analytic hierarchy process based automatic generation control of multi-area interconnected power system using Jaya algorithm. *Engineering Applications of Artificial Intelligence*, *60*(December 2016), 35–44.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, *47*(1), 65–74.

Tseng, F. T., & Stafford, E. F. (2008). New MILP models for the permutation flowshop problem. *Journal of the Operational Research Society*, *59*(10), 1373–1386.

Venkata Rao, R. (2016). Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *International Journal of Industrial Engineering Computations*, *7*, 19–34.

Venkata Rao, R. (2019). Introduction. In *Jaya: An Advanced Optimization Algorithm and its Engineering Applications* (pp. 1–8). https://doi.org/10.1007/978-3-319-78922-4_1

Wang, L., & Zheng, D. Z. (2003). An effective hybrid heuristic for flow shop scheduling. *International Journal of Advanced Manufacturing Technology*, *21*(1), 38–44.

Zhang, W., Wang, Y., Yang, Y., & Gen, M. (2019). Hybrid multiobjective evolutionary algorithm based on differential evolution for flow shop scheduling problems. *Computers & Industrial Engineering*, *130*, 661–670. https://doi.org/10.1016/J.CIE.2019.03.019

Zhang, Y., Yang, X., Cattani, C., Rao, R. V., Wang, S., & Phillips, P. (2016). Tea category identification using a novel fractional fourier entropy and Jaya algorithm. *Entropy*, *18*(3), 1–17. https://doi.org/10.3390/e18030077

Zhao, F., Liu, H., Zhang, Y., Ma, W., & Zhang, C. (2018). A discrete Water Wave Optimization algorithm for no-wait flow shop scheduling problem. *Expert Systems with Applications*, *91*, 347–363.