

Improved teaching–learning-based and JAYA optimization algorithms for solving flexible flow shop scheduling problems

Raviteja Buddala¹  · Siba Sankar Mahapatra¹

Received: 2 May 2017 / Accepted: 21 October 2017 / Published online: 3 November 2017
© The Author(s) 2017. This article is an open access publication

Abstract Flexible flow shop (or a hybrid flow shop) scheduling problem is an extension of classical flow shop scheduling problem. In a simple flow shop configuration, a job having ‘g’ operations is performed on ‘g’ operation centres (stages) with each stage having only one machine. If any stage contains more than one machine for providing alternate processing facility, then the problem becomes a flexible flow shop problem (FFSP). FFSP which contains all the complexities involved in a simple flow shop and parallel machine scheduling problems is a well-known NP-hard (Non-deterministic polynomial time) problem. Owing to high computational complexity involved in solving these problems, it is not always possible to obtain an optimal solution in a reasonable computation time. To obtain near-optimal solutions in a reasonable computation time, a large variety of meta-heuristics have been proposed in the past. However, tuning algorithm-specific parameters for solving FFSP is rather tricky and time consuming. To address this limitation, teaching–learning-based optimization (TLBO) and JAYA algorithm are chosen for the study because these are not only recent meta-heuristics but they do not require tuning of algorithm-specific parameters. Although these algorithms seem to be elegant, they lose solution diversity after few iterations and get trapped at the local optima. To alleviate such drawback, a new local search procedure is proposed in this paper to improve the solution quality. Further, mutation strategy (inspired from genetic

algorithm) is incorporated in the basic algorithm to maintain solution diversity in the population. Computational experiments have been conducted on standard benchmark problems to calculate makespan and computational time. It is found that the rate of convergence of TLBO is superior to JAYA. From the results, it is found that TLBO and JAYA outperform many algorithms reported in the literature and can be treated as efficient methods for solving the FFSP.

Keywords Flexible flow shop · JAYA algorithm · Makespan · Meta-heuristics · Teaching–learning-based optimization

Introduction

Problems of scheduling occur in many economic domains, such as airplane scheduling, train scheduling, time table scheduling and especially in the shop scheduling of manufacturing organizations. A flexible flow shop (FFS), which is also known as hybrid flow shop or flow shop with parallel machines, consists of a set of machine centres with parallel machines. In other words, it is nothing but an extension of the simple flow shop scheduling problem (FSP). In real practice, a shop with a single processor at each stage is rarely encountered. Generally, processors are duplicated in parallel at stages. The purpose of duplication is to balance the capacity of stages, increase the overall shop floor capacity, reduce, if not eliminate, the impact of bottleneck stages and so on. A flexible flow shop scheduling problem (FFSP) has M jobs with ‘g’ number of operations to be carried out in ‘g’ stages. Each job n ($n = 1, 2, \dots, M$) is to be sequentially processed at each stage t ($t = 1, 2, \dots, g$). Each stage t ($t = 1, 2, \dots, g$) has

✉ Raviteja Buddala
raviteja317@gmail.com
Siba Sankar Mahapatra
mahapatrass2003@gmail.com

¹ Department of Mechanical Engineering, National Institute of Technology, Rourkela, Odisha 769008, India

MT_t identical machines. At least one stage will have more than one machine and only one operation is carried out in each stage. The parallel machines in a stage are identical and take same time to process an operation. At any given time, a machine can process only one job. The processing time of each job m at a stage t “ $p(n, t)$ ” is deterministic and it is known in advance. The objective is to find permutation of all jobs at each stage so as to reduce the makespan. For a scheduling problem, the makespan is defined as the completion time of the last job to leave the production system (Pinedo 2008). Among all the shop-scheduling problems, flexible flow shop problem (FFSP) is treated as one of the difficult NP-hard (Non-deterministic polynomial time) problems as pointed out by Gupta (1988) and Hooeveen et al. (1996). It is unlikely that an exact solution to FFSP can be found by a polynomial time algorithm. However, an exact solution can be obtained by various methods of reduced enumeration, typically by a branch and bound (B&B) algorithm. In any case, only small-size instances can be handled by exact methods. To find a good solution for large size problems of practical interest within an acceptable amount of time, two types of algorithms, such as (1) approximation algorithms and (2) heuristic or meta-heuristic algorithms, are used. An algorithm is called an approximation algorithm if it is possible to establish analytically how close the generated solution is to the optimum (either in the worst-case or on average). The performance of a heuristic algorithm is usually analyzed experimentally, through a number of runs using either generated instances or known benchmark instances. Heuristic algorithms can be very simple but still effective. Some of modern heuristics based on various ideas of local search are listed as neighbourhood search, tabu search, simulated annealing, genetic algorithms, etc.

Even though the performance of exact methods such as branch and bound is superior to heuristic and meta-heuristic techniques, exact methods fail to solve large size problems. On the other hand, heuristic techniques are problem-dependent and easily get trapped at the local optimum. To overcome the drawbacks of both exact and heuristic methods, researchers focus on meta-heuristic approaches which happen to be problem-independent and applied to solve problems independent of their size. Recently, many meta-heuristics have been applied in recent past to solve the FFSP to generate near-optimal solutions in a reasonable computation time (Oguz and Ercan 2005; Engin and Doyen 2004; Niu et al. 2009; Cui and Gu 2015). Usually, meta-heuristics are population-based algorithms which can be broadly classified into two groups. They are evolutionary algorithms (EA’s)- and swarm intelligence (SI’s)-based algorithms. Some of the famous EA’s are genetic algorithm (GA), evolution strategy (ES), differential equation (DE), evolutionary programming (EP), etc.

Some of the famous swarm intelligence-based algorithms are particle swarm optimization (PSO), ant colony optimization (ACO), fire fly algorithm (FF), artificial bee colony (ABC) algorithm, etc. Other than these two broad categories, there are other population-based algorithms, such as harmony search (HS), bio-geography-based optimization (BBO), eco-geography-based optimization (EBO) and gravitational search (GS) algorithm, etc. Most of these algorithms have one thing in common, i.e. tuning of the respective algorithm-specific parameters. For example, GA contains parameters, such as mutation and crossover probabilities. PSO contains parameters, such as inertia weight and acceleration constants. Artificial bee colony (ABC) uses the number of scout bees, onlooker bees and employed bees. HS uses pitch adjusting rate and memory consideration rate. Similarly, other algorithms also have their own tuning parameters. To obtain good optimized solutions to specific problems, tuning of the algorithm-specific parameters is essential. Since finding the right tuning parameters is a difficult task, an efficient tuning parameter-less algorithms is required to overcome this drawback. Teaching-learning-based optimization (TLBO) and JAYA are some of the recent optimization techniques that were proposed without any algorithm-specific tuning parameters in comparison with any other meta-heuristics. The present work focuses on the application of TLBO and JAYA algorithms to solve FFSP. TLBO, proposed by Rao et al. (2011), has been applied to different kinds of optimization problems in the past and found to be one of the good algorithms solving NP-hard problems. JAYA algorithm is proposed by Rao (2016) recently. It is observed that JAYA algorithm seems to be an efficient one in solving the constrained and unconstrained optimization benchmark problems. Hence, an attempt is made in this work to apply JAYA to solve FFSP. Further, TLBO is a two-phase algorithm, whereas JAYA solves the optimization problem in a single phase.

The meta-heuristics in the present work also require some improvements in the basic algorithm so that their efficiency may be improved in generating the optimal or near-optimal solutions. Many researchers have embedded local search techniques with meta-heuristics to improve the efficiency of meta-heuristics. For example, Wang et al. (2011) have used a local search technique to simulated annealing (SA) to improve its efficiency while solving FFSP. Choong et al. (2011) have proposed hybrid algorithms combining tabu search (TS) and simulated annealing (SA) to particle swarm optimization (PSO) to improve efficiency of PSO in solving FFSP. Liao et al. (2012) have used a local search technique to PSO. Therefore, a new local search technique is proposed in the present work to improve the solution quality of FFSP generated by present algorithms. Although these algorithms seem to be elegant, they

lose solution diversity after some iteration and may get trapped at the local optima. To overcome this drawback, mutation strategy from genetic algorithm is incorporated to the algorithm to maintain the diversity in the population.

Literature review

A methodology based on branch and bound (B&B), which is considered to be an exact method, has been proposed by Arthanari and Ramamurthy (1971) to solve FFSP. Carlier and Neron (2000) have applied B&B to solve 77 benchmark problems of FFSP. Lower bound for makespan proposed by Santos et al. (1995) can be used as a measure to find the efficiency of the algorithms. To minimize makespan and flow time, Brah and Loo (1999) have proposed several heuristics (previously applied to FSP) to solve FFSP. A regression analysis is proposed to study the effect of problem characteristics on performance of heuristics. Ruiz et al. (2008) have conducted tests on several dispatching rules on realistic FFSP. Results show that modified NEH (Nawaz, Enscore, Ham) heuristic proposed by Nawaz et al. (1983) outperform all other heuristics. Ying and Lin (2009) have proposed an efficient heuristic called the heuristic of multistage hybrid flow shop (HMHF) to solve FFSP with multi-processor tasks. Kahraman et al. (2010) have proposed a parallel greedy algorithm (PGA) to solve FFSP with multi-processor tasks with four constructive heuristic techniques that were used to develop the PGA.

Initially, the mostly used meta-heuristic is GA for solving FFSP with an objective to minimize makespan (Ulusoy 2004; Oguz and Ercan 2005; Ruiz and Maroto 2006; Kahraman et al. 2008; Shiau et al. 2008; Urlings et al. 2010; Engin et al. 2011). Artificial immune system (AIS), which uses the affinity maturation mechanism and clonal selection principle, has been successfully applied by Engin and Doyen (2004) to solve FFSP. With the inspiration from natural phenomenon of ant colony, ant colony optimization (ACO) has been applied by Ying and Lin (2006) to solve FFSP. Alaykyran et al. (2007) have conducted a parametric study on the application of ACO to FFSP and proposed an improved ACO. Niu et al. (2009) have proposed a quantum-inspired immune algorithm (QIA), a hybrid algorithm inspired from quantum algorithm (QA) and immune algorithm (IA), for solving 41 FFSP from Carlier and Neron (2000) instances. Results indicate that QIA is superior to QA and IA. Wang et al. (2011) have proposed SA technique to solve FFSP with multi-processor tasks with an aim to minimize makespan. They have used three different decoding techniques (list scheduling, permutation scheduling, and first-fit method) and a local search technique to SA to obtain makespan. Mousavi et al. (2013) have proposed SA local search technique to minimize combined

total tardiness and makespan for FFSP. Choong et al. (2011) have hybridized PSO with SA and TS techniques and proposed two hybrid PSO algorithms to solve FFSP. Liao et al. (2012) have proposed PSO hybridized with bottleneck heuristic to exploit the bottleneck stage and incorporated SA technique to avoid trapping at the local optima followed by a local search technique to improve the solution quality. Singh and Mahapatra (2012) have proposed PSO to solve FFSP using chaotic numbers generated from logistic mapping function instead of random numbers. Based on the features of integrated greedy algorithms and artificial immune system (AIS), Ying (2012) has proposed a new hybrid immune algorithm (HIA) to solve multistage FFSP with makespan as the objective. Chou (2013) has proposed a cocktail decoding method capable of improving the solution quality of any algorithm incorporated with PSO for solving FFSP with multi-processor tasks. An improved cuckoo search algorithm has been proposed by Marichelvam et al. (2014) to solve multi-stage FFSP. To obtain the near-optimal solutions rapidly using cuckoo search, initial solutions are generated with NEH heuristic. Cui and Gu (2015) have proposed an improved discrete artificial bee colony (IDABC) algorithm to solve FFSP. The IDABC uses a combination of modified variable neighbourhood search (VNS) and differential evolution (DE) techniques to generate new solutions. A modified harmony search approach using smallest position value (SPV) rule is applied to solve multi-stage FFSP with an aim to minimize makespan by Marichelvam and Geetha (2016). For critical analysis of FFSP problems, Ribas et al. (2010) have presented a literature review on exact, heuristic and meta-heuristics methods proposed to solving FFSP. Careful analysis of literature on FFSP reveals that meta-heuristic approaches are widely used for improving solution quality. However, any algorithm or its hybridisation variant needs tuning of algorithm-specific parameters to obtain good solutions. An algorithm that does not require algorithm-specific parameters to be tuned may be another way of generating solution quality because it may reduce the burden of computation of tuning the algorithm for each problem instance. A brief description of the FFSP and its formulation is given in the next section.

Problem description and formulation

FFSP, commonly known as hybrid flow shop problem, has M jobs with ‘ g ’ number of operations to be carried out in ‘ g ’ stages. Each job n ($n = 1, 2, \dots, M$) is to be sequentially processed at each stage t ($t = 1, 2, \dots, g$). Each stage t ($t = 1, 2, \dots, g$) have MT_t identical machines. At least one stage will have more than one machine and only one operation is carried out in each stage. The parallel machines in a stage are identical and take same time to process an

operation. At any given time, a machine can process only one job. The processing time of each job m at a stage t “ $p(n, t)$ ” is deterministic and it is known in advance. The objective is to find permutation of all jobs at each stage to reduce the makespan. The arrows in the Fig. 1 indicate the possible ways that a job can be processed from one stage to another. There are few assumptions (Marichelvam et al. 2014) involved in solving this problem. They are:

- The number of machines at each stage and the number of stages are known in advance.
- The processing times of jobs on these machines are deterministic and are known in advance.
- There is no preemption during the process.
- The transportation times and setup times of jobs are included in the processing times.
- At a time, a machine can process only one job.
- There is no machine breakdown or any other disturbance during the process.

The notations used are as follows:

n	Number of jobs ($n = 1, 2, \dots, M$)
g	Number of stages ($t = 1, 2, \dots, g$)
MTt	Total number of machines at stage $t, (t = 1, 2, \dots, g)$
Jkt	Jobs that are processed on machine k at stage t
$Pntk$	Processing time of job n at stage t on machine k
$Cntk$	Completion time of job n at stage t on machine k
$Sntk$	Starting time of job n at stage t on machine k
$Yntk$	= 1 if job n is processed at stage t on machine k . 0, otherwise
htk	Job h is processed at stage t on machine k . $h = 1, 2, \dots, MTt$

The mathematical model of FFSP is as explained below.

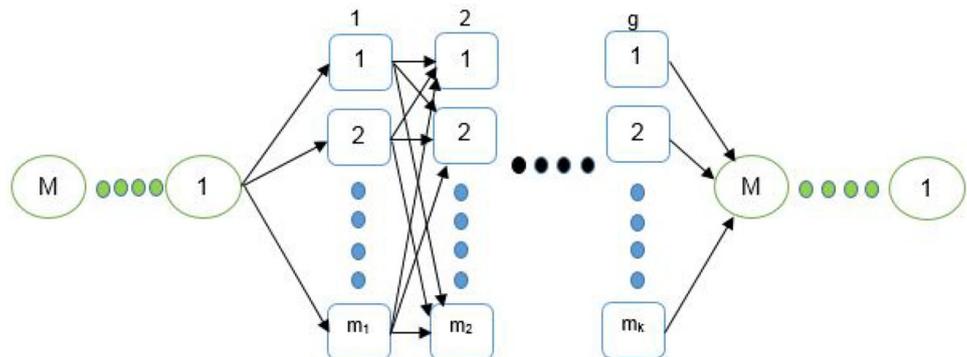
The objective is to minimize makespan (C_{max}):

$$C_{max} \geq Cng \quad \forall n. \tag{1}$$

Subject to constraints

$$\sum_{k=1}^{MTt} Yntk = 1, \quad n = 1, 2, \dots, M; \quad t = 1, 2, \dots, g, \tag{2}$$

Fig. 1 Schematic flow diagram of FFSP



$$\sum_{k=1}^{MTt} Jkt = n, \quad \forall t \in g, \tag{3}$$

$$Cntk \leq Sn(t + 1)r, \quad \forall t \in g, \forall r \in MTt, \tag{4}$$

$$Cntk = Sntk + Pntk \quad \forall n, t, k, \tag{5}$$

$$Chtk \leq S(h + 1)tk \quad \forall t, k, h, \tag{6}$$

$$C_{max} \geq Cng \quad \forall n, \tag{7}$$

Inequality (1) determines the makespan. Constraint (2) governs the assignment of each job at every stage. Constraint (3) assures that the total number of jobs assigned in a stage is equal to n . Constraint (4) ensures that a job cannot be processed in a later stage unless it is completely processed on the current stage. Constraint (5) determines the completion time of a job at a given stage. Constraint (6) determines that a machine cannot process next job unless it processes the current job completely. Constraint (7) determines that the makespan of a schedule should be always greater than or equal to completion time of all jobs at the last stage.

The problem is classified as NP-hard problem (Gupta 1988). The computation time increases exponentially if the size of the problem increases. Therefore, exact solution hardly gives a good solution in reasonable time. The researchers tend to use various meta-heuristic approaches to solve such type of problems. But turning of algorithm-specific parameter is really difficult to attain quality solutions. Hence, TLBO and JAYA algorithms which do not have any algorithm-specific tuning parameters are chosen for the study to evaluate their performance.

Teaching-learning-based optimization

TLBO is proposed by Rao et al. (2011) with an inspiration from the general teaching-learning process how a teacher influence the knowledge of students. Students and teacher are the two main objects of a class and the algorithm explains the two modes of learning, i.e. via teacher (teacher phase) and discussion among the fellow students (student

phase). A group of students of the class constitute the population of the algorithm. In any iteration ‘*i*’, the best student of the class becomes the teacher. Execution of the TLBO is explained in two phases, such as teacher phase and student phase.

Teacher phase

A teacher puts his/her best to increase the knowledge of his/her students to his level. But practically it is not possible as learning by a student depends on his/her potential to learn. In this process, each student’s knowledge increases and in turn the average knowledge of the class increases. At any iteration ‘*i*’, let Z_{mean} denote the mean knowledge of the class and teacher of the class is denoted as Z_{teacher} . Then increased knowledge of a student is given by the expression:

$$Z_{\text{new}i} = Z_{\text{old}i} + r \times (Z_{\text{teacher}} - (T_f \times Z_{\text{mean}})), \tag{8}$$

where r is a random number between zero to one and T_f is called teaching factor whose value is randomly chosen as one or two, and there is no tuning of this teaching factor even though T_f is an algorithm-specific parameter of TLBO. $Z_{\text{new}i}$ is the new knowledge of the student ‘*i*’ after learning from the teacher. $Z_{\text{old}i}$ is the previous knowledge of the student ‘*i*’. Accept $Z_{\text{new}i}$ if it gives a better functional value.

Student phase

After a class is taught, students discuss among themselves. In this process, the knowledge of all students increases. At any iteration ‘*i*’, let Z_a and Z_b be two students who discuss after the class, $a \neq b$. Then increased knowledge of the student is given by the expression:

$$Z_{\text{new}a} = Z_{\text{old}a} + r_i \times (Z_a - Z_b) \text{ if } F(Z_a) < = F(Z_b), \tag{9}$$

$$Z_{\text{new}a} = Z_{\text{old}a} + r_i \times (Z_b - Z_a) \text{ if } F(Z_b) < F(Z_a). \tag{10}$$

$Z_{\text{new}a}$ is the new knowledge of the student ‘*a*’ after learning from the co-student ‘*b*’. $Z_{\text{old}a}$ is the previous knowledge of the student ‘*a*’. Accept $Z_{\text{new}a}$ if it gives a better function value. The pseudo code of proposed TLBO is given in Fig. 3.

JAYA algorithm

JAYA algorithm is proposed by Rao (2016). This algorithm is designed on the simple logic that any solution of the given population should always move towards the best

solution and move away from the worst solution. The beauty of this algorithm is that it contains only one equation and it does not contain any algorithm-specific parameters to be tuned to get optimal solution, which makes it very easy to understand and uniquely different from any other meta-heuristics. As compared to TLBO, JAYA has only one phase.

The mathematical description of JAYA algorithm is as follows. Let $f(x)$ be the objective function to be optimized. At any iteration ‘*i*’, let Z_{best} and Z_{worst} denote the best and worst solutions, respectively, among the population, then a solution of the population is modified as follows:

$$Z_{\text{new}i} = Z_{\text{old}i} + r_1 \times (Z_{\text{best}} - |Z_i|) - r_2 \times (Z_{\text{worst}} - |Z_i|), \tag{11}$$

where r_1 and r_2 are the two random numbers between zero and one. The term $r_1 \times (Z_{\text{best}} - |Z_i|)$ denotes the nature of solution to move towards the best solution and the term $-r_2 \times (Z_{\text{worst}} - |Z_i|)$ denotes the nature of the solution to move away from the worst solution. The new solution is accepted if it gives a better value. The flow chart of JAYA algorithm is given in Fig. 2. The pseudo code of proposed JAYA algorithm is given in Fig. 3.

Although the TLBO and JAYA algorithms seem to be elegant, they lose solution diversity after few iterations and get trapped at the local optima. To alleviate such drawback, a new local search procedure is proposed in this paper to improve the solution quality. A brief description of the proposed local search is given in the next section of the paper as follows.

Local search

Literature suggests that researchers use local search techniques to improve the efficiency of meta-heuristics. For example, Wang et al. (2011) have used a local search technique to SA to improve solution quality. Choong et al. (2011) have embedded TS and SA to PSO separately to improve PSO efficiency. Liao et al. (2012) have used a local search technique to improve the efficiency of PSO. The present tuning parameter-less algorithms also require some improvements in the basic algorithm so that their efficiency may be improved in generating the optimal or near-optimal solutions. Although these algorithms seem to be elegant, they lose solution diversity after few iterations and may get trapped at the local optima. To overcome this drawback, a new local search technique is proposed in the present work to improve the solution quality of FFSP. This method is explained in two steps: (1) Sequence swap (2) machine swap.

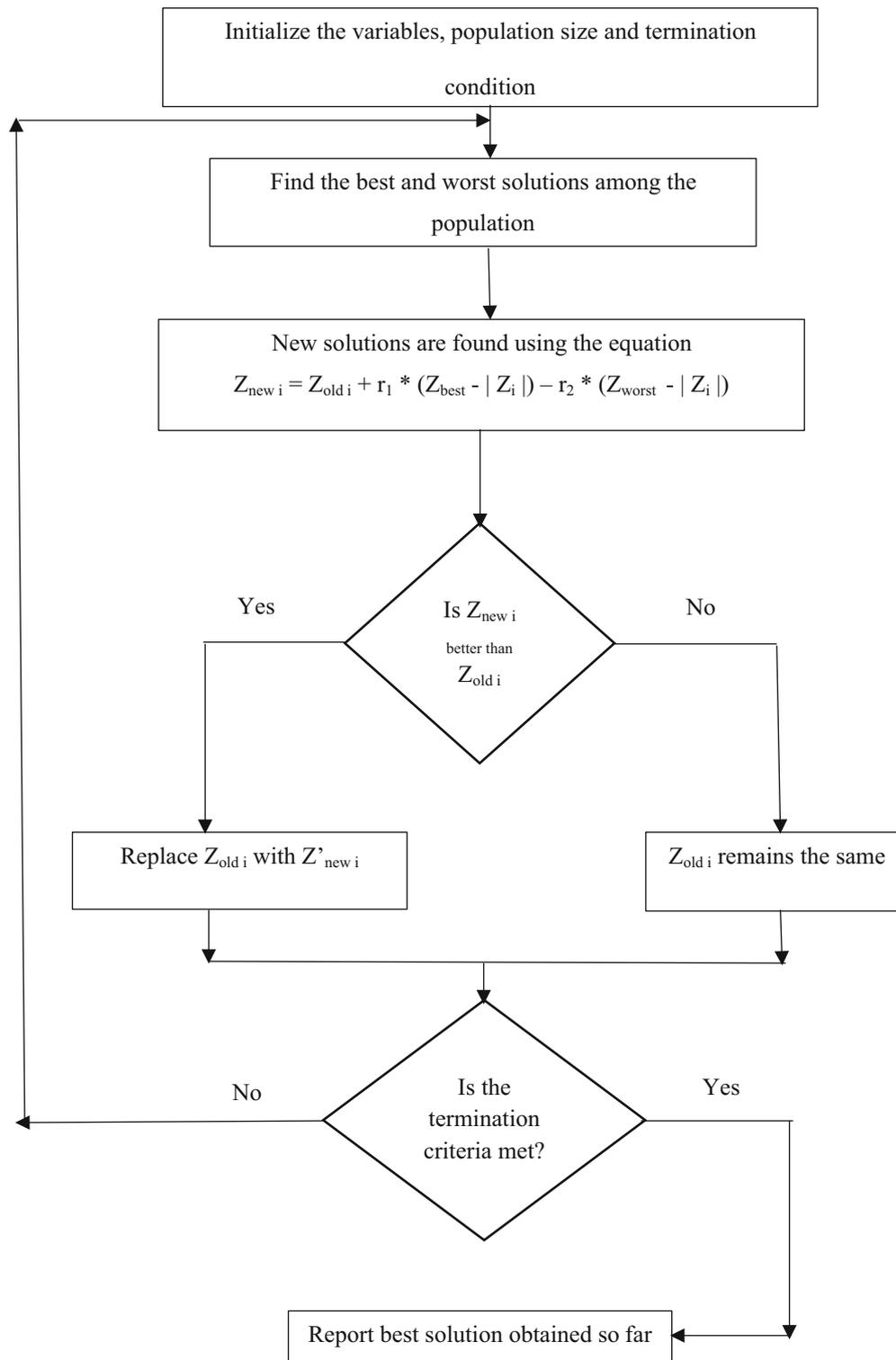


Fig. 2 Flow chart of Jaya algorithm

Sequence swap

Before going to the sequence swap, all the critical operations of the schedule are to be found. Then two or more

jobs that are allotted on the same machine adjacent to each other are picked and adjacent swapping [pair exchange method used by Xia and Wu (2005) and Buddala and Mahapatra (2016)] will be done to check whether there is



Pseudo code for TLBO algorithm	Pseudo code for JAYA algorithm
<pre> Start Initialize the random population, maximum iteration number and iteration count variable iter=0. While (iter < max iter number) Iter =iter +1 Calculate mean of the population Evaluate functional value of each student Start teacher phase Obtain new solutions using equation 8. Accept new solution if it gives a better functional value. End teacher phase Start student phase Obtain new solutions using equation 9&10. Accept new solution if it gives a better functional value. End student phase Conduct the local search. Start Mutation strategy End mutation strategy End while End </pre>	<pre> Start Initialize the random population, maximum iteration number and iteration count variable iter=0. While (iter < max iter number) Iter =iter +1 Evaluate functional value of each one of population Obtain new solutions using equation 11. Accept new solution if it gives a better functional value. Conduct the local search. Start Mutation strategy End mutation strategy End while End </pre>

Fig. 3 Pseudo codes for proposed algorithms

any improvement in the quality of solution. This technique is called sequence swap. If the swap gives a better makespan value, then old solution is replaced by the new solution.

Machine swap

In a FFSP problem, at least one stage will have more than one machine. There is a possibility that critical operations can be allotted to the same machine in a stage with multiple machines. In such condition if one of the jobs is transferred to other parallel machine of the same stage, then there is a

possibility for makespan reduction. This is called machine swap technique. If this swap gives a better makespan value, then old solution is replaced by the new solution.

Even though the TLBO and JAYA algorithms converge quickly, it is observed that the population of these algorithms lose diversity and get trapped at the local optimum. Therefore, to maintain diversity in the population, mutation strategy from GA is incorporated to these algorithms with an inspiration from Singh and Mahapatra (2012). Sing and Mahapatra have incorporated mutation strategy to PSO to maintain the diversity in the population. The working of mutation strategy is as follows.



Mutation strategy

Even though the rate of convergence of both the TLBO and JAYA is good, it is observed that when these algorithms are directly applied to solve the FFSP, they show a good tendency to be struck at the local optimum due to lack of diversity in the population. Therefore, as the iterations proceed, all the populations get converged to a local optimum and there is no improvement in the solution there after. To avoid this drawback, mutation strategy from genetic algorithm is incorporated to each of these basic algorithms after the local search step. Singh and Mahapatra (2012) have applied this mutation strategy to PSO for solving FFSP. Mutation is defined as a sudden change that occurs in the genes of the offspring when compared to that of parent genes during the evolution process. Out of the total population, a fixed percentage of population will undergo mutation. In this case, the percentage is fixed to three percent. It is based on the computational experiments conducted and it is explained in Fig. 5 under results and discussion. This mutation technique is implemented only if there is no improvement in the best solution for every fixed number of iterations. In the present case, mutation is applied if no improvement is observed in five iterations. This technique prevents the population converging to the local optimum and directs the same to the global optimum. After the incorporation of mutation strategy, we observe a good improvement in the quality of results produced by these two algorithms.

Problem mapping mechanism

In this method, a real number encoding system is used to solve the FFSP where the integer part is used to assign the machine at each stage and the fractional part is used to sequence the jobs allotted on each machine. Let us consider an example with jobs ($n = 4$) and operations ($g = 3$), i.e. three stages. Stages one, two and three have three, two and one machines, respectively ($m_1 = 3, m_2 = 2, m_3 = 1$). We generate 12 (4×3) real numbers randomly as shown in second row of Fig. 4 by uniform distribution between $[1, 1 + m(k)]$ for each stage. The processing times $p(n,$

$$t) = \begin{pmatrix} 4 & 6 & 2 \\ 2 & 3 & 5 \\ 7 & 1 & 2 \\ 1 & 2 & 3 \end{pmatrix}. \text{ In stage 1, job 1 is allotted to}$$

machine 1, job 2 and job 4 are allotted to machine 2, and job 3 is allotted to machine 3. The method adopted here is that for stage one the jobs on same machine are sequenced in the increasing order of fractional values. So job 2 is followed by job 4 in stage 1 as fractional value of job 2 is less than that of job 4. For stages greater than one

sequence, priority is given to the completion times of jobs in their previous stage. A job whose completion time is small in the previous stage is sequenced first in the current stage. In stage 2 machine 1, job 3 is followed by job 1 because completion time of job 1 in stage 1 is less than the job 3 in stage 1. If the completion times are same then a job is randomly chosen for sequencing.

Results and discussion

To evaluate the effectiveness of TLBO and JAYA algorithms in solving FFSP, experiments have been conducted on 77 benchmark problems taken from Carlier and Neron (2000). All the problems have been solved and the results are compared with various algorithms, such as BB, PSO, GA, AIS, IDABC, IA, QA, and QIA, from the literature available. Experiments have been conducted using MATLAB software on a 4-GB ram i7 processor running at 3.40 GHz on windows 7 platform. The problem notation, for instance j15c10a1 from Table 1, means a 15-jobs 10-stages problem where j stands for number of jobs, c stands for number of stages, a stands for machine distribution structure and ‘1’ is the index of the problem. In the present work, the problem sizes vary from jobs $10 \times$ stages 5 to jobs $15 \times$ stages 10.

Experiments have been carried out for different percentage of mutation ranging one to five. It is observed that best results of most of the problems are obtained when the mutation probability is fixed to three percent. This is clear from Fig. 5a, b which shows the variation of results obtained to some of the difficult problems solved by varying the mutation percentage for TLBO and JAYA, respectively. In this mutation process, randomly three solutions from the population are picked and they are replaced by new randomly generated solutions. In this way, a diversity in the population is maintained before converging to the optimal solution.

Exact methods such as B&B fail to produce best results for large size FFSP problems. Therefore, there is a need for alternative method to find the best result (least possible makespan) that can be obtained for a FFSP. The least possible makespan values are nothing but the lower bound (LB) values. Therefore, to solve this problem, Santos et al. (1995) derived a formula to find lower bound for FFSP. In the present work, the formula to calculate lower bound values is taken from Santos et al. (1995) and it is given in the Eq. 12. Also, it is well known that heuristic and meta-heuristics does not guarantee a best result all the time. Therefore, lower bounds are essential and are used for comparing the efficiency of different algorithms.



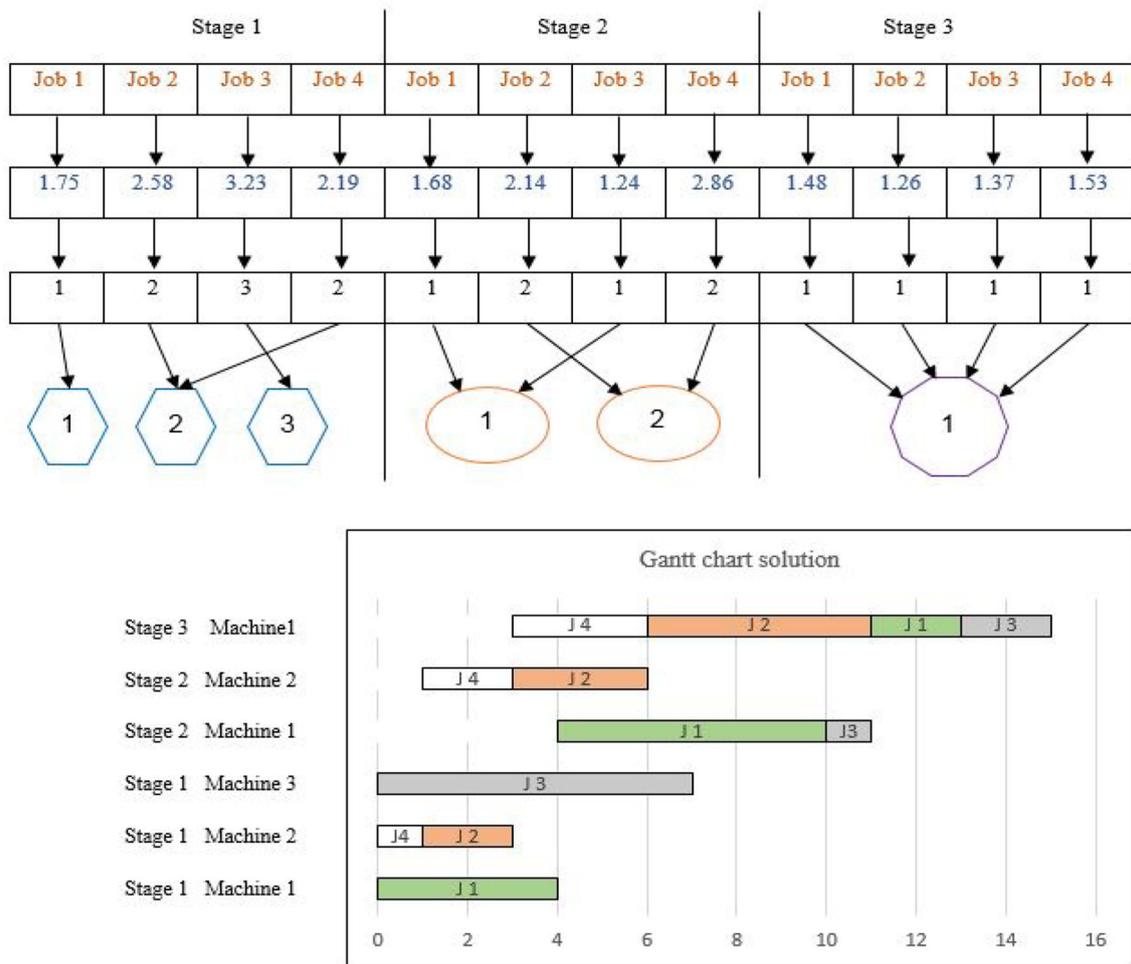


Fig. 4 Problem mapping mechanism and solution representation

$$\begin{aligned}
 LB = \max & \left[\max \left\{ \sum_{t=1}^g p(i,t) \right\}, \max \left\{ \frac{1}{MT(t)} \left(\sum_{y=1}^{MT(t)} LSA(y,t) \right. \right. \right. \\
 & \left. \left. \left. + \sum_{i=1}^n p(i,t) + \sum_{y=1}^{MT(t)} RSA(y,t) \right) \right\} \right]. \tag{12}
 \end{aligned}$$

In Eq. 12, $MT(t)$ is the total number of machines at stage t , $p(i,t)$ is the processing time of job i at stage t , LSA is left side processing times arranged in ascending order for stages in left side and RSA is right side processing times arranged in ascending order for stages in right side. For more information regarding the derivation of the above formula, one may refer to Santos et al. (1995).

Since meta-heuristics are stochastic in nature, the performance of the proposed algorithms is carried out through ten times and best values obtained by these algorithms are tabulated in Table 1. In Table 1, C_{max} stands for makespan and PD is the percentage deviation of the solution from the lower bound (LB) value. The second column in Table 1 gives the LB values calculated using Eq. 12. At the end of

the Table 1, average percentage deviation (APD) of each algorithm to the problems is calculated. The formula to find PD and APD are given in Eqs. 13 and 14, respectively.

$$PD = \frac{C_{max} - LB}{LB} \times 100, \tag{13}$$

$$APD = \frac{\sum_{X=1}^N PD(X)}{N}. \tag{14}$$

In Eq. 14, N is the total number of problems available with results to an algorithm and X is the index of the problem. A comparison of computational times taken by different algorithms to reach its optimal solution is shown in Table 2. In Table 2, C_{max} stands for makespan, $T(s)$ denotes time in seconds, and a denotes that the algorithm could not reach its optimum in 1600 s. A comparison of the best and average makespan values obtained by TLBO and JAYA are shown in Table 3. In Table 3, C_{max} stands for makespan.

A comparison of convergence rates for TLBO and JAYA are given in the following Figs. 6, 7, 8 and 9 (for

Table 1 Results of benchmark problems

Problem	BB (Carlier and Neron 2000)		PSO (Liao et al. 2012)		GA (Kahraman et al. 2008)		AIS (Engin and Doyen 2004)		IDABC (Cui and Gu 2015)		IA (Niu et al. 2009)		QA (Niu et al. 2009)		QIA (Niu et al. 2009)		Proposed TLBO		Proposed JAYA	
	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD
j10c5a2	88	0	88	0	88	0	88	0	88	0	88	0	88	0	88	0	88	0	88	0
j10c5a3	117	0.86	117	0.86	117	0.86	117	0.86	117	0.86	117	0.86	117	0.86	117	0.86	117	0.86	117	0.86
j10c5a4	121	0	121	0	121	0	121	0	121	0	121	0	121	0	121	0	121	0	121	0
j10c5a5	122	0	122	0	122	0	122	0	122	0	122	0	122	0	122	0	122	0	122	0
j10c5a6	110	0	110	0	110	0	110	0	110	0	110	0	110	0	110	0	110	0	110	0
j10c5b1	130	0	130	0	130	0	130	0	130	0	130	0	130	0	130	0	130	0	130	0
j10c5b2	107	0	107	0	107	0	107	0	107	0	107	0	107	0	107	0	107	0	107	0
j10c5b3	109	0	109	0	109	0	109	0	109	0	110	0.92	110	0.92	109	0	109	0	109	0
j10c5b4	122	0	122	0	122	0	122	0	122	0	122	0	124	1.64	122	0	122	0	122	0
j10c5b5	153	0	153	0	153	0	153	0	153	0	153	0	153	0	153	0	153	0	153	0
j10c5b6	115	0	115	0	115	0	115	0	115	0	115	0	115	0	115	0	115	0	115	0
j10c5c1	67	1.49	68	1.49	68	1.49	68	1.49	68	1.49	76	13.4	79	17.9	69	2.99	68	1.49	69	2.99
j10c5c2	74	0	74	0	74	0	74	0	74	0	80	8.11	82	10.8	76	2.7	74	0	75	1.35
j10c5c3	64	10.9	71	10.9	71	10.9	72	12.5	72	12.5	77	20.3	81	26.6	74	15.6	71	10.9	74	15.6
j10c5c4	61	8.2	66	8.2	66	8.2	66	8.2	66	8.2	81	32.8	81	32.8	75	23	67	9.84	69	13.1
j10c5c5	76	2.63	78	2.63	78	2.63	78	2.63	78	2.63	83	9.21	87	14.5	79	3.95	78	2.63	78	2.63
j10c5c6	64	7.81	69	7.81	69	7.81	69	7.81	69	7.81	77	20.3	80	25	72	12.5	69	7.81	72	12.5
j10c5d1	62	6.45	66	6.45	66	6.45	66	6.45	66	6.45	73	17.7	75	21	69	11.3	66	6.45	68	9.68
j10c5d2	63	15.9	73	15.9	73	15.9	73	15.9	73	15.9	81	28.6	85	34.9	76	20.6	74	17.5	76	20.6
j10c5d3	56	14.3	64	14.3	64	14.3	64	14.3	64	14.3	71	26.8	75	33.9	68	21.4	65	16.1	67	19.6
j10c5d4	66	6.06	70	6.06	70	6.06	70	6.06	70	6.06	79	19.7	83	25.8	75	13.6	70	6.06	73	10.6
j10c5d5	63	4.76	66	4.76	66	4.76	66	4.76	66	4.76	75	19	78	23.8	71	12.7	66	4.76	68	7.94
j10c5d6	56	10.7	62	10.7	62	10.7	62	10.7	62	10.7	68	21.4	74	32.1	64	14.3	63	12.5	64	14.3
j10c10a1	139	0	139	0	139	0	139	0	139	0	141	1.44	142	2.16	139	0	139	0	139	0
j10c10a2	158	0	158	0	158	0	158	0	158	0	160	1.27	163	3.16	158	0	158	0	158	0
j10c10a3	148	0	148	0	148	0	148	0	148	0	150	1.35	151	2.03	148	0	148	0	148	0
j10c10a4	149	0	149	0	149	0	149	0	149	0	151	1.34	154	3.36	149	0	149	0	149	0
j10c10a5	148	0	148	0	148	0	148	0	148	0	149	0.68	150	1.35	148	0	148	0	148	0
j10c10a6	146	0	146	0	146	0	146	0	146	0	147	0.68	147	0.68	146	0	146	0	146	0
j10c10b1	163	0	163	0	163	0	163	0	163	0	NA	NA	NA	NA	NA	NA	163	0	163	0
j10c10b2	157	0	157	0	157	0	157	0	157	0	NA	NA	NA	NA	NA	NA	157	0	157	0
j10c10b3	169	0	169	0	169	0	169	0	169	0	NA	NA	NA	NA	NA	NA	169	0	169	0
j10c10b4	159	0	159	0	159	0	159	0	159	0	NA	NA	NA	NA	NA	NA	159	0	159	0

Table 1 continued

Problem	LB using Eq. 12	BB (Carlier and Neron 2000)	PSO (Liao et al. 2012)		GA (Kahraman et al. 2008)	AIS (Engin and Doyen 2004)		IDABC (Cui and Gu 2015)		IA (Niu et al. 2009)	QA (Niu et al. 2009)		QIA (Niu et al. 2009)	Proposed TLBO		Proposed JAYA	
			C_{max}	PD		C_{max}	PD	C_{max}	PD		C_{max}	PD		C_{max}	PD	C_{max}	PD
j10c10b5	162	165	1.85	0	165	1.85	165	1.85	165	1.85	NA	NA	NA	165	1.85	165	1.85
j10c10b6	165	165	0	165	0	165	0	165	0	165	0	165	0	165	0	165	0
j10c10c1	110	127	15.5	115	4.55	4.55	115	4.55	115	4.55	NA	NA	NA	116	5.45	121	10
j10c10c2	116	116	0	117	0.86	0.86	119	2.59	119	2.59	NA	NA	NA	117	0.86	118	1.72
j10c10c3	113	133	17.7	113	0	116	2.65	116	2.65	116	2.65	NA	NA	116	2.65	119	5.31
j10c10c4	112	135	20.5	112	0	120	7.14	120	7.14	120	7.14	NA	NA	115	2.68	118	5.36
j10c10c5	121	145	19.8	125	3.31	3.31	126	4.13	125	3.31	NA	NA	NA	125	3.31	129	6.61
j10c10c6	100	112	12	103	3	106	6	106	6	106	6	NA	NA	105	5	108	8
j15c5a1	178	178	0	178	0	178	0	178	0	178	0	NA	NA	178	0	178	0
j15c5a2	165	165	0	165	0	165	0	165	0	165	0	NA	NA	165	0	165	0
j15c5a3	130	130	0	130	0	130	0	130	0	130	0	NA	NA	130	0	130	0
j15c5a4	156	156	0	156	0	156	0	156	0	156	0	NA	NA	156	0	156	0
j15c5a5	162	164	1.23	164	1.23	164	1.23	164	1.23	164	1.23	NA	NA	164	1.23	164	1.23
j15c5a6	178	178	0	178	0	178	0	178	0	178	0	NA	NA	178	0	178	0
j15c5b1	170	170	0	170	0	170	0	170	0	170	0	NA	NA	170	0	170	0
j15c5b2	152	152	0	152	0	152	0	152	0	152	0	NA	NA	152	0	152	0
j15c5b3	157	157	0	157	0	157	0	157	0	157	0	NA	NA	157	0	157	0
j15c5b4	145	147	1.38	147	1.38	147	1.38	147	1.38	147	1.38	NA	NA	147	1.38	147	1.38
j15c5b5	166	166	0	166	0	166	0	166	0	166	0	NA	NA	166	0	166	0
j15c5b6	175	175	0	175	0	175	0	175	0	175	0	NA	NA	175	0	175	0
j15c5c1	85	85	0	85	0	85	0	85	0	85	0	NA	NA	86	1.18	89	4.71
j15c5c2	88	90	2.27	90	2.27	91	3.41	91	3.41	90	2.27	NA	NA	91	3.41	94	6.82
j15c5c3	87	87	0	87	0	87	0	87	0	87	0	NA	NA	88	1.15	91	4.6
j15c5c4	87	90	3.45	89	2.3	89	2.3	89	2.3	89	2.3	NA	NA	89	2.3	93	6.9
j15c5c5	73	84	15.1	74	1.37	75	2.74	74	1.37	74	1.37	NA	NA	75	2.74	78	6.85
j15c5c6	91	91	0	91	0	91	0	91	0	91	0	NA	NA	91	0	91	0
j15c5d1	167	167	0	167	0	167	0	167	0	167	0	NA	NA	167	0	167	0
j15c5d2	80	85	6.25	84	5	84	5	84	5	84	5	NA	NA	84	5	87	8.75
j15c5d3	75	96	28	83	10.7	83	10.7	83	10.7	82	9.33	NA	NA	83	10.7	87	16
j15c5d4	76	101	32.9	81	6.58	84	10.5	84	10.5	84	10.5	NA	NA	84	10.5	88	15.8
j15c5d5	72	97	34.7	80	11.1	80	11.1	80	11.1	79	9.72	NA	NA	80	11.1	83	15.3
j15c5d6	78	87	11.5	82	5.13	82	5.13	82	5.13	81	3.85	NA	NA	82	5.13	85	8.97
j15c10a1	236	236	0	236	0	236	0	236	0	236	0	236	0	236	0	236	0

Table 1 continued

Problem	LB using Eq. 12	BB (Carlier and Neron 2000)		PSO (Liao et al. 2012)		GA (Kahraman et al. 2008)		AIS (Engin and Doyen 2004)		IDABC (Cui and Gu 2015)		IA (Niu et al. 2009)		QA (Niu et al. 2009)		QIA (Niu et al. 2009)		Proposed TLBO		Proposed JAYA		
		C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	C_{max}	PD	
j15c10a2	200	0	200	0	200	0	200	0	200	0	200	0	205	2.5	210	5	200	0	200	0	207	3.5
j15c10a3	198	0	198	0	198	0	198	0	198	0	198	0	200	1.01	202	2.02	198	0	198	0	200	1.01
j15c10a4	224	0.45	225	0.45	225	0.45	225	0.45	225	0.45	225	0.45	228	1.79	229	2.23	225	0.45	225	0.45	229	2.23
j15c10a5	183	0.55	182	0	182	0	182	0	182	0	182	0	186	2.2	189	3.85	182	0	182	0	183	0.55
j15c10a6	200	0	200	0	200	0	200	0	200	0	200	0	200	0	205	2.5	200	0	200	0	201	0.5
j15c10b1	215	3.26	222	3.26	222	3.26	222	3.26	222	3.26	222	3.26	223	3.72	223	3.72	222	3.26	222	3.26	222	3.26
j15c10b2	187	0	190	1.6	187	0	187	0	187	0	187	0	187	0	187	0	187	0	187	0	187	0
j15c10b3	222	0	222	0	222	0	222	0	222	0	222	0	222	0	222	0	222	0	222	0	222	0
j15c10b4	221	0	221	0	221	0	221	0	221	0	221	0	221	0	221	0	221	0	221	0	221	0
j15c10b5	199	0.5	200	0.5	200	0.5	200	0.5	200	0.5	200	0.5	202	1.51	203	2.01	200	0.5	200	0.5	200	0.5
j15c10b6	219	0	219	0	219	0	219	0	219	0	219	0	219	0	221	0.91	219	0	219	0	219	0
APD			4.14	2.03	2.28	2.3	2.22	6.31	8.23	3.9	2.32	3.63										

Bold values indicate APD values

few problems, first problem from each set), where blue line indicates JAYA and black line indicates TLBO algorithms, respectively. Results show that rate of convergence of TLBO to reach the optimal result is superior to JAYA.

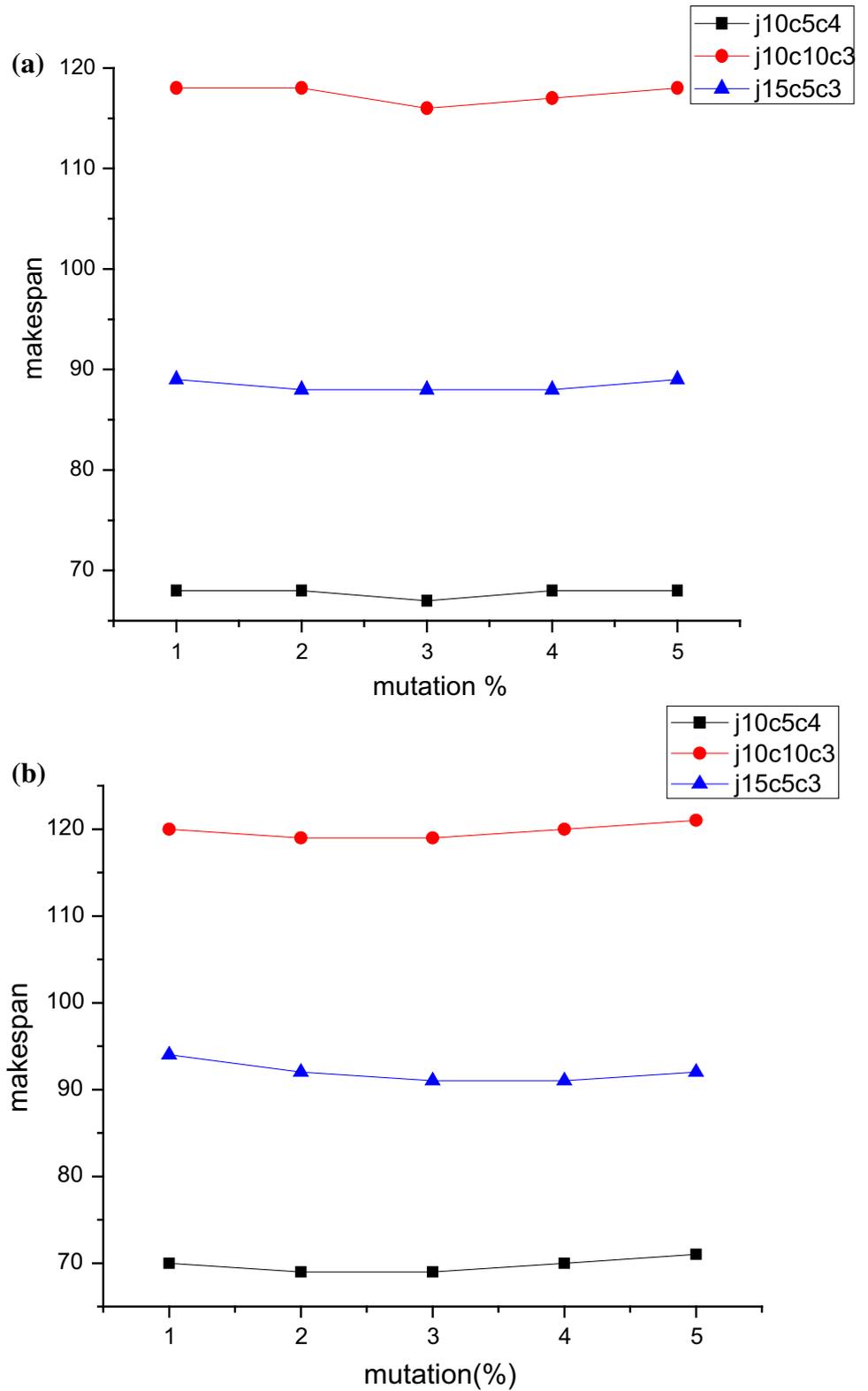
Out of these 77 problems, only forty-one problems have been solved by IA, QA and QIA whose APD values for these 41 problems are 6.31, 8.23 and 3.9, respectively. The APD value of TLBO for these 41 problems is 2.47 whose value is superior to QA, IA and QIA, and hence outperformed these three algorithms. Also, TLBO (APD = 2.32) outperformed BB (APD = 4.14). The total APD (for 77) value of TLBO is 2.32 which is very much closer to APD values of remaining algorithms, such as PSO (APD = 2.03), AIS (APD = 2.3), GA (APD = 2.28) and IDABC (APD = 2.22). Thus, we can conclude that TLBO is also one of the best meta-heuristic techniques that can be applied to solve FFSP.

The APD value of JAYA for the forty-one problems solved by IA, QA and QIA is 3.5 which outperformed all these three QA (APD = 6.31), IA (APD = 8.23) and QIA (APD = 3.9) algorithms. JAYA’s total APD (for 77 problems) value is 3.63 which also outperformed BB (APD = 4.14). The remaining algorithms PSO (APD = 2.03), AIS (APD = 2.3), GA (APD = 2.28) and IDABC (APD = 2.22) are superior in APD to JAYA (APD = 3.63). Thus, we can conclude that JAYA is one of the competitive meta-heuristic techniques that can be applied to solve FFSP.

Conclusions

In case of application of meta-heuristics of solving NP-hard problems, tuning of algorithm-specific parameters is essential to obtain near-optimal solutions. The present research work focuses on the strength of “algorithm-specific tuning parameter-less algorithms” as tuning of the “algorithm-specific tuning parameters” is a difficult task. A problem needs to be solved again and again until the right tuning parameters are found for that problem. This further increases the computational burden. This paper discusses the problem of flexible flow shop scheduling problems with an objective to minimize makespan. To the best of our knowledge, all the meta-heuristic algorithms that were used to solve FFSP are either very complex or have algorithm-specific parameters to be tuned or a combination of both. Therefore, the present paper focuses on simple meta-heuristic techniques which do not have any algorithm-specific tuning parameters that can be used to solve NP-hard FFSP problem. TLBO and JAYA are selected for the study as they proved to be simple and efficient algorithms in the recent past. A new local search technique followed by mutation strategy borrowed from

Fig. 5 a Result variation with mutation percentage for TLBO. **b** Result variation with mutation percentage for JAYA



GA has been incorporated in these basic algorithms to improve the efficiency. The proposed TLBO and JAYA have been applied to conduct computational experiments

on 77 benchmark problems of FFSP and the makespan results obtained by these algorithms are compared with all the other eight algorithms available in the literature.

Table 2 Computational times

Problem	B&B		PSO		GA		AIS		IDABC		IA		QA		QIA		Proposed TLBO		Proposed JAYA	
	C_{max}	T (s)	C_{max}	T (s)	C_{max}	T (s)														
j10c5a3	117	7	117	0.002	117	0.000	117	1	117	0.003	117	0.003	117	0.002	117	0.002	117	0.002	117	0.003
j10c5b3	109	9	109	0.012	109	0.000	109	1	109	0.003	110	0.003	109	0.003	109	0.003	109	0.003	109	0.004
j10c5c3	71	240	71	36.997	71	0.016	72	a	72	a	77	71	74	a	74	a	71	a	74	a
j10c5d3	64	481	64	0.098	64	0.015	64	15	64	0.007	71	75	68	65	65	0.008	65	0.008	67	0.022
j10c10a3	148	58	148	0.017	148	0.047	148	1	148	0.008	150	151	148	148	148	0.006	148	0.006	148	0.015
j10c10b3	169	19	169	0.014	169	0.000	169	1	169	0.007	-	-	-	169	0.015	169	0.015	169	0.058	
j10c10c3	133	a	116	a	116	a	116	a	116	a	-	-	-	116	a	119	a	119	a	
j15c5a3	130	34	130	0.006	130	0.015	130	1	130	0.017	-	-	-	130	0.005	130	0.005	130	0.021	
j15c5b3	157	15	157	0.03	157	0.015	157	1	157	0.014	-	-	-	157	0.023	157	0.023	157	0.042	
j15c5c3	87	202	87	2.398	87	0.109	87	16	87	0.048	-	-	-	88	0.057	88	0.057	91	0.183	
j15c5d3	96	a	82	a	83	a	83	a	82	a	-	-	-	83	a	87	a	87	a	
j15c10a3	198	45	198	0.171	198	0.063	198	4	198	0.032	200	202	198	198	0.030	198	0.030	200	0.060	
j15c10b3	222	80	222	0.007	222	0.015	222	1	222	0.039	222	222	222	222	0.045	222	0.045	222	0.093	

Bold values indicate the best values

Table 3 Comparison of best and average makespan values

Problem	Proposed TLBO		Proposed JAYA	
	Best C_{max}	Average C_{max}	Best C_{max}	Average C_{max}
j10c5a3	117	117	117	117
j10c5b3	109	109	109	109
j10c5c3	71	71.4	74	74.8
j10c5d3	65	65.6	67	68.4
j10c10a3	148	148	148	148
j10c10b3	169	169	169	169.2
j10c10c3	116	116.8	119	122
j15c5a3	130	130	130	130
j15c5b3	157	157	157	157.4
j15c5c3	88	89.4	91	92.2
j15c5d3	83	84.4	87	88.6
j15c10a3	198	198	200	200
j15c10b3	222	222	222	222.5

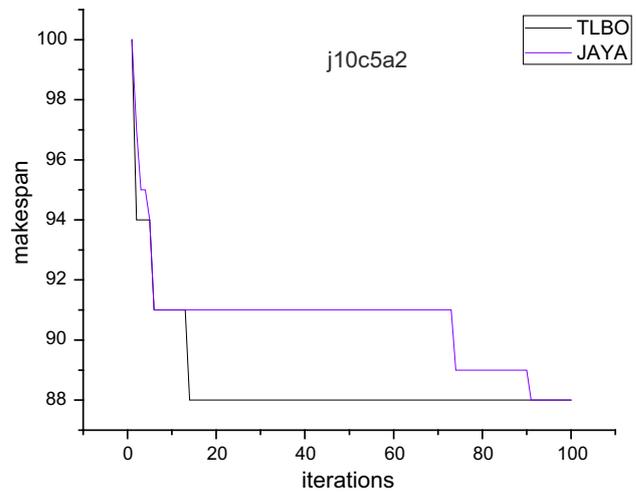


Fig. 6 Problem instance j10c5a2

Makespan and APD results from the Table 1 show that TLBO is one of the efficient algorithms that can be applied to solve FFSP since it produces results comparable to best results suggested by other algorithms. It is also observed that TLBO outperforms many algorithms, such as QA, IA, QIA and BB, in solving FFSP. On the other hand, JAYA which is an easy and simple to understand having only one algorithm-specific equation has also outperformed some algorithms, such as QA, IA, QIA and BB. It also gives near-optimal makespan results to best algorithms available in the literature. Thus, it can be concluded that JAYA is one of the best competitive meta-heuristic technique that can be applied to solve FFSP.

This paper encourages the future researchers to focus on “algorithm-specific tuning parameter-less algorithms” that can be applied to solve FFSP. As finding the right tuning

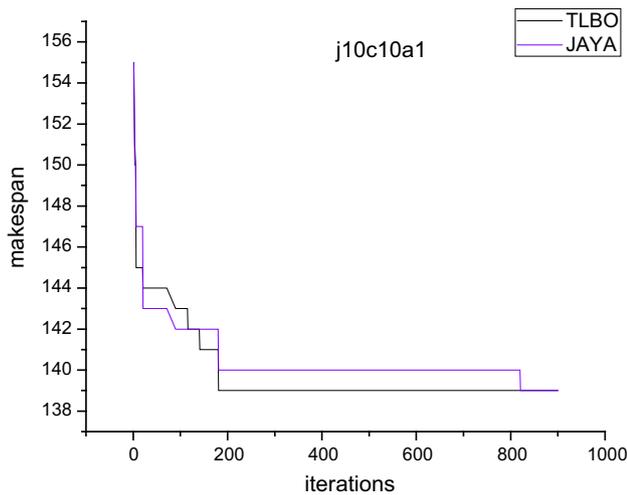


Fig. 7 Problem instance j10c10a1

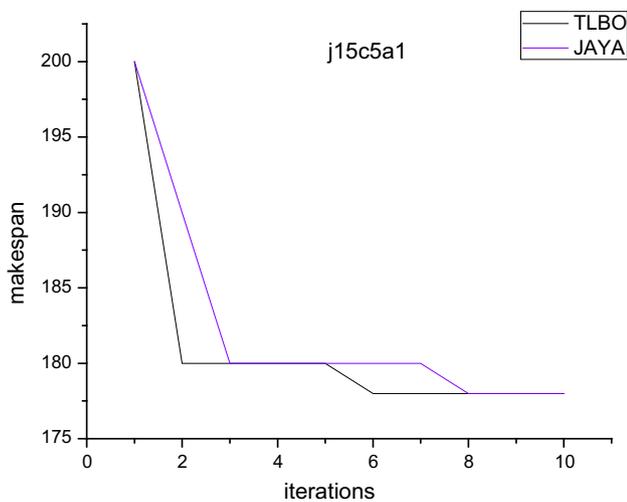


Fig. 8 Problem instance j15c5a1

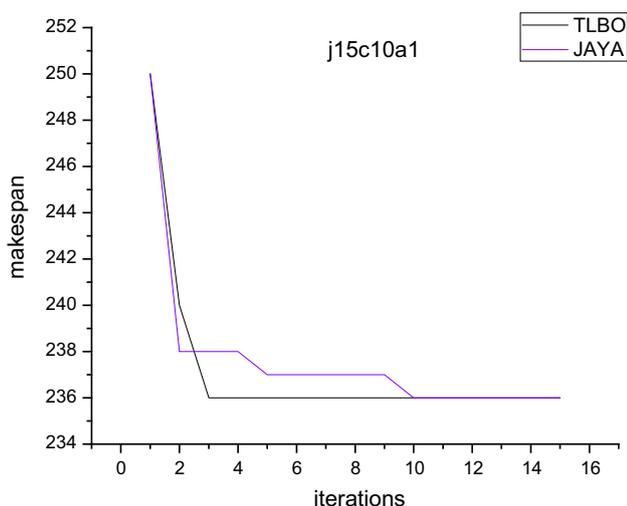


Fig. 9 Problem instance j15c10a1

parameters is a difficult task, this drawback can be addressed using the algorithm-specific tuning parameter-less algorithms, such as TLBO and JAYA. The present study can be extended to hybridize TLBO and JAYA with other algorithms in the future. The study can also be extended considering different uncertainties, such as machine breakdown and variable processing times in future.

Acknowledgements The authors express hearty thanks to the Editor-in-chief of JIEI and anonymous reviewers for their careful reading and suggestions in improving the quality of the paper.

Open Access This article is distributed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits unrestricted use, distribution, and reproduction in any medium, provided you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license, and indicate if changes were made.

References

Alaykyran K, Engin O, Doyen A (2007) Using ant colony optimization to solve hybrid flow shop scheduling problems. *Int J Adv Manuf Technol* 35(5):541–550

Arthanari TS, Ramamurthy KG (1971) An extension of two machines sequencing problem. *Opsearch* 8(1):10–22

Brah SA, Loo LL (1999) Heuristics for scheduling in a flow shop with multiple processors. *Eur J Oper Res* 113(1):113–122

Buddala R, Mahapatra SS (2016) An effective teaching learning based optimization for flexible job shop scheduling. *Int Conf Electr Electron Optim Tech IEEE* 3087–3092

Carlier J, Neron E (2000) An exact method for solving the multiprocessor flow-shop. *RAIRO Oper Res* 34(1):1–25

Choong F, Phon-Amnuaisuk S, Alias MY (2011) Metaheuristic methods in hybrid flow shop scheduling problem. *Expert Syst Appl* 38(9):10787–10793

Chou FD (2013) Particle swarm optimization with cocktail decoding method for hybrid flow shop scheduling problems with multiprocessor tasks. *Int J Prod Econ* 141(1):137–145

Cui Z, Gu X (2015) An improved discrete artificial bee colony algorithm to minimize the makespan on hybrid flow shop problems. *Neurocomputing* 148:248–259

Engin O, Doyen A (2004) A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Gener Comput Syst* 20(6):1083–1095

Engin O, Ceran G, Yilmaz MK (2011) An efficient genetic algorithm for hybrid flow shop scheduling with multiprocessor task problems. *Appl Soft Comput* 11(3):3056–3065

Gupta JN (1988) Two-stage, hybrid flowshop scheduling problem. *J Oper Res Soc* 1:359–364

Hoogeveen JA, Lenstra JK, Veltman B (1996) Minimizing the makespan in a multiprocessor flow shop is strongly NP-hard. *Eur J Oper Res* 89(1):172–175

Kahraman C, Engin O, Kaya I, Kerim Yilmaz M (2008) An application of effective genetic algorithms for solving hybrid flow shop scheduling problems. *Int J Comput Intell Syst* 1(2):134–147

Kahraman C, Engin O, Kaya I, Ozturk RE (2010) Multiprocessor task scheduling in multistage hybrid flow-shops: a parallel greedy algorithm approach. *Appl Soft Comput* 10(4):1293–1300

- Liao CJ, Tjandradjaja E, Chung TP (2012) An approach using particle swarm optimization and bottleneck heuristic to solve hybrid flow shop scheduling problem. *Appl Soft Comput* 12(6):1755–1764
- Marichelvam MK, Geetha M (2016) Application of novel harmony search algorithm for solving hybrid flow shop scheduling problems to minimise makespan. *Int J Ind Syst Eng* 23(4):467–481
- Marichelvam MK, Prabaharan T, Yang XS (2014) Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan. *Appl Soft Comput* 19:93–101
- Mousavi SM, Zandieh M, Yazdani M (2013) A simulated annealing/local search to minimize the makespan and total tardiness on a hybrid flowshop. *Int J Adv Manuf Technol* 64(1–4):369–388
- Nawaz M, Enscore EE, Ham I (1983) A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. *Omega* 11(1):91–95
- Niu Q, Zhou T, Ma S (2009) A quantum-inspired immune algorithm for hybrid flow shop with makespan criterion. *J UCS* 15(4):765–785
- Oğuz C, Ercan MF (2005) A genetic algorithm for hybrid flow-shop scheduling with multiprocessor tasks. *J Scheduling* 8(4):323–351
- Pinedo M (2008) *Scheduling: theory, algorithms, and systems*, 3rd edn. Springer, Germany, Berlin
- Rao R (2016) Jaya: A simple and new optimization algorithm for solving constrained and unconstrained optimization problems. *Int J Ind Eng Comput* 7(1):19–34
- Rao RV, Savsani VJ, Vakharia DP (2011) Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. *Comput Aided Des.* 43(3):303–315
- Ribas I, Leisten R, Framiñan JM (2010) Review and classification of hybrid flow shop scheduling problems from a production system and a solutions procedure perspective. *Comput Oper Res* 37(8):1439–1454
- Ruiz R, Maroto C (2006) A genetic algorithm for hybrid flowshops with sequence dependent setup times and machine eligibility. *Eur J Oper Res* 169(3):781–800
- Ruiz R, Şerifoglu FS, Urlings T (2008) Modeling realistic hybrid flexible flowshop scheduling problems. *Comput Oper Res* 35(4):1151–1175
- Santos DL, Hunsucker JL, Deal DE (1995) Global lower bounds for flow shops with multiple processors. *Eur J Oper Res* 80(1):112–120
- Shiau DF, Cheng SC, Huang YM (2008) Proportionate flexible flow shop scheduling via a hybrid constructive genetic algorithm. *Expert Syst Appl* 34(2):1133–1143
- Singh MR, Mahapatra SS (2012) A swarm optimization approach for flexible flow shop scheduling with multiprocessor tasks. *Int J Adv Manuf. Technol* 62(1–4):267–277
- Ulusoy G (2004) Multiprocessor task scheduling in multistage hybrid flow-shops: a genetic algorithm approach. *J Oper Res Soc* 55(5):504–512
- Urlings T, Ruiz R, Serifoglu FS (2010) Genetic algorithms with different representation schemes for complex hybrid flexible flow line problems. *Int J Metaheuristics* 1(1):30–54
- Wang HM, Chou FD, Wu FC (2011) A simulated annealing for hybrid flow shop scheduling with multiprocessor tasks to minimize makespan. *Int J Adv Manuf Technol* 53(5):761–776
- Xia W, Wu Z (2005) An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. *Comput Ind Eng* 48(2):409–425
- Ying KC (2012) Minimising makespan for multistage hybrid flowshop scheduling problems with multiprocessor tasks by a hybrid immune algorithm. *Eur J Ind Eng* 6(2):199–215
- Ying KC, Lin SW (2006) Multiprocessor task scheduling in multistage hybrid flow-shops: an ant colony system approach. *Int J Prod Res* 44(16):3161–3177
- Ying KC, Lin SW (2009) Scheduling multistage hybrid flowshops with multiprocessor tasks by an effective heuristic. *Int J Prod Res* 47(13):3525–3538

