Int J Adv Manuf Technol (2013) 68:1583–1599
DOI 10.1007/s00170-013-4956-3

ORIGINAL ARTICLE

Hybrid flexible flowshop scheduling problem with unrelated parallel machines and limited waiting times

S. F. Attar · M. Mohammadi · R. Tavakkoli-Moghaddam

Received: 26 October 2011 / Accepted: 22 March 2013 / Published online: 25 April 2013

Abstract This paper deals with the problem that reflects real-world situations adequately. Several constraints including unrelated machines, limited waiting times between every two successive processing operations and ready time of jobs are studied. These constraints and characteristics affect on some operations in a large number of companies. In recent researches, they have been tackled many times while so far have not been considered simultaneously. The aim of this paper is to model and solve the addressed problem by applying an efficient metaheuristic algorithm, entitled biogeography-based optimization (BBO). To assess the proposed BBO, two experiments are conducted and their results in terms of solutions quality, as well as computation efficiency compared against two popular algorithms, namely imperialist competitive algorithm and population-based simulated annealing. Due to the sensitivity of the values of parameters in the metaheuristic algorithms, a response surface methodology as a strength statistical tool is used to tune the parameters. The computational results show that the proposed BBO algorithm significantly outperforms the other foregoing algorithms.

Keywords Hybrid flexible flowshop · Limited waiting times · Unrelated parallel machines · Ready times · Biogeography-based optimization · Response surface methodology

S. F. Attar · M. Mohammadi (✉)
Department of Industrial Engineering, Faculty of Engineering, Kharazmi University, Karaj, Iran
e-mail: mohammadi@tmu.ac.ir

R. Tavakkoli-Moghaddam
Department of Industrial Engineering and Center of Excellence for Intelligence Based Experimental Mechanics, College of Engineering, University of Tehran, Tehran, Iran

1 Introduction

Sequencing and scheduling is a form of decision making that plays a crucial role in manufacturing and service industries. In the current competitive environment, an effective sequencing and scheduling has become a necessity for survival in the market place. In fact, scheduling problems are the allocation of limited resources to perform a set of activities in a period of time [1].

A hybrid flexible flowshop scheduling (HFFS) problem gives two concepts: (1) a flexible flowshop scheduling (FFS) problem consists of several stages so that at least one stage contains more than one machine and all jobs visit a chain of operations in a unique order. (2) A HFFS problem includes the same concept as the FFS problem except that jobs can skip from stages. Skipping is a feature that makes the problem more adaptable. Additionally, we consider some important constraints, including:

- Release date of jobs means jobs might not be available immediately to be processed.
- Unrelated machines express that at least one stage contains parallel machines and machine i can process job j at speed v_{ij} [1]. In other words, the speed of each machine is dependent on the job which is processed on it.
- Limited waiting times constraint means the waiting times between any two consecutive processing operations cannot be greater than a given upper bound.

This kind of HFFS problems with limited waiting times constraint are applied in many industrial environments (e.g., airplane engine production, electronics manufacturing, semi-conductors, and petrochemical production [2]). It is used in steel-making processes so that the melted steel must be kept liquid hot enough in the buffer for next processing. Also, the waiting time after the operations in furnace tubes is

limited in order to prevent the absorption of particles suspended in the air [3]. One of the most important applications of our problem is when the capacity of intermediate buffers is bounded and products must leave buffers in a short time. When the values of upper bounds for the waiting times are equal to zero or in other words, there are no buffers between the stages, it is a sample of no-wait problems. However, the introduced problem is an important study; the literature review reveals that there is no research with these assumptions.

The rest of this paper is organized as follows: In Section 2, literature reviews is presented. Section 3 describes the characteristics and assumptions of our problem in details. In Section 4, the structures of the applied algorithms are explained. Section 5 displays an illustrative example that is solved by biogeography-based optimization (BBO) algorithm. Then, the numerical tests established to solve the problems are illustrated in Section 6. Finally, Section 7 presents the summary of this research with the recommendation for further studies.

2 Literature review

In this section, we briefly explain some previous studies in the field of hybrid flowshop scheduling (HFS) problems under the various constraints and objective function.

Lin and Liao [4] investigated a two-stage hybrid flowshop scheduling (HFS) with one machine at the first stage and two groups of machines at the second stage, in which each group has two identical machines. Low et al. [5] focused on the system with unrelated machines at the first stage, in which these machines belong to a certain type and each type of machines can process a subset of job types. Bertel and Billaut [6] developed a mixed-integer programming model and heuristic approaches to solve the three-stage HFS problem. Garey and Johnson [7] demonstrated that the HFS problem in order to minimize the makespan is strongly NP hard. Thus, they proposed the several heuristics and approximation algorithms for various HFS problems. Kahraman et al. [8] showed that a parallel greedy algorithm (PGA) is effective to solve a HFS problem with multiprocessor task for minimizing the maximum of completion time. They proved PGA by comparing the result of this metaheuristic algorithm with the earlier studies of Oğuz et al. [9, 10] and Oğuz [11]. A general class of heuristics for the multistage FFS problem with uniform parallel machines and the objective of minimizing of makespan are developed by Kyparisis and Koulamas [12].

Gholami et al. [13] investigated the HFS problem with sequence-dependent setup time and stochastic breakdown of machines. Allahverdi and Al-Anzi [14] studied a simulated annealing (SA) method for an *m*-stage HFS that models client–server requests and then presented several heuristics. Ying and Lin [15] developed ant colony optimization for the multiprocessor task problem with precedence relationships and indicated their results were superior to the results reported by Oğuz et al. [9]. The performance of dispatching rules, some appropriate heuristics for an *m*-stage problem with uniform parallel machines and identical jobs is investigated by Verma and Dessouky [16]. Tang and Zhang [17] combined a neural network approach with local search to improve the quality of the obtained solutions for the HFS problem in order to minimize the sum of setup times. Wang and Tang [18] developed tabu search (TS) combined with a scatter search (SS) method to minimize the weighted sum of completion time of all jobs for the HFS problem with finite intermediate buffers.

Kurz and Askin [19] examined four heuristics to find schedules to minimize the makespan in flexible flow lines with sequence-dependent setups. These methods included a simplistic greedy method, approaches based on the natures of TSP and flow line problems with an application of the random keys genetic algorithm. Naderi et al. [20] studied a flexible flowshop with anticipatory sequence-dependent setup times and job-independent transportation times in a multitransporter system to minimize the total weighted tardiness. They formulated the problem as a mixed-integer linear programming model and proposed an electromagnetism algorithm. Low [21] proposed a SA method for the problem minimizing the total flow time and considered unrelated parallel machines at each stage with sequence-dependent removal times and sequence-independent setup times. Ruiz and Maroto [22] focused on an HFS problem with unrelated parallel machines, sequence-dependent setup times and machine eligibility and proposed some genetic algorithms. Jungwattanakit et al. [23] applied both constructive approach (e.g., insertion-based approach) and iterative methods (i.e., SA, TS, and genetic algorithms) to minimize a convex combination of the makespan and the number of tardy jobs for the FFS problem with unrelated parallel machines, sequence and machine-dependent setup times. Low [21] offered a SA-based heuristic method to minimize the total flow time for a multistage flowshop scheduling problem with unrelated parallel machines considering independent setup times and dependent removal times constraints.

A mixed-integer programming model for the flexible flow line problem without intermediate buffers by assuming in-process buffers and sequence-dependent setup time is presented by Tavakkoli-Moghaddam and Safaei [24]. Ruiz et al. [25] provided a mixed-integer programming model and some heuristics for a flowshop problem from the ceramic tile sector with machines release dates, unrelated parallel machines, machine eligibility, skip possibility, sequence-dependent setup times, anticipatory and non-anticipatory possibility for setup times, positive and negative time lags, and precedence relationships between jobs.

Chang et al. [26] presented a heuristic method to solve the no-wait HFS problem. Su [3] proposed a mixed-integer programming model and a heuristic algorithm to minimize the makespan for a hybrid two-stage flowshop problem with a batch processor in stage 1 that can process a batch (limited number) of jobs simultaneously and a single processor in stage 2 considered the waiting time for the second stage that cannot be greater than a given upper bound. Yang and Chern [27] indicated the problem with limited waiting time constraints and one single processor at each stage is NP hard and represented a branch-and-bound algorithm to solve the problem. A constructive backtracking heuristic approach to minimize the makespan for the HFS problem with limited waiting time constraint is presented by Li and Li [28]; however, they supposed the fixed upper bound of waiting times for all jobs and stages. Chen [29] introduced eight mixed-binary integer programming models for open shop, job shop, flowshop, and permutation flowshop environments with limited waiting time constraints to minimize the makespan. Liu et al. [30] studied the HFS problem with limited waiting time constrain to minimize the makespan. They proposed a constructive backtracking heuristic method that was composed of a recursive backtracking algorithm and TS. A discrete colonial competitive algorithm (CCA) to determine a schedule that minimizes the sum of the linear earliness and quadratic tardiness in the HFS problem with sequence-dependent setup times and limited waiting time is developed by Behnamian and Zandieh [2]. To justify this method, they compared the CCA with the hybrid TS-SS proposed by Wang and Tang [18] and recursive backtracking combined with TS proposed by Liu et al. [30].

Although many realistic considerations and constraints have been addressed in several papers in literature, a few papers have considered such realistic constraints jointly. Also, there is no attempt to join the set of realistic constraints along with the limited waiting times constraint.

According to our studies, researchers have not utilized our proposed BBO. Simon [31] stated that this method is a novel method for solving the NP hard problems. In this paper, we prove that this method is more efficient than imperialist competitive algorithm (ICA) and population-based simulated annealing (PBSA) methods.

3 Problem formulation

The HFFS problem contains a set of n jobs and s stages, where $N=\{1, 2, \ldots, n\}$ and $S=(1, 2, \ldots, s)$. Jobs are available at the different times indicated by $R=\{r_1, r_2, \ldots, r_n\}$, such a way that each job may skip some stages. Thus, they must be processed on their required stages. In this problem, we assume all jobs visit entire stages and the processing times at the skipped stages are equal to 0. Accordingly, the sequence of operations for all jobs is on the unique order of s stages. Job $k \in N$ must be processed by only one machine in each stage. In ith stage, there are m_i-unrelated machines in parallel that means the speed of machine j is dependent on the job processed on it. The waiting times between any two consecutive operations in the buffer cannot be greater than its given upper bound. The limited waiting time of job k between two operations i and $i+1$ is illustrated with $lwt_{i,\ i+1,\ k}$. Job k needs the processing time p_{ijk} to be processed at stage i on machine j.

3.1 Assumptions

Modification, removal or addition of assumptions and/or constraints to the standard problem which described in Ruiz et al. [32] leads to different HFS variants. The characteristics and assumptions that present our problem are as follows:

1. All data used in all test problems are known deterministically.
2. In HFS, each stage has at least one machine, and at least one stage has more than one machine.
3. There are n independent jobs which are processed without defect. Additionally, all jobs have neither preemption nor priority values for processing.
4. The processing operations for all jobs cannot be interrupted. Processors are available at all times with no breakdowns and no scheduled or unscheduled maintenance.
5. Each operation must be processed by only one machine at each stage. Furthermore, all machines in a stage are capable to process all jobs.
6. Transportation and set up times between the stages are assumed negligible and ignored.
7. A part which its processing has been completed at a stage is transferred either directly to an available machine in the next stage or to a buffer ahead of that stage.
8. The intermediate storages are unlimited.
9. The waiting time between any two consecutive processing operations is limited.
10. A machine can process only one job at a time.
11. Jobs might not be available for processing immediately.
12. The processing time of every job depends on the machine which processes it. In other words, there are machines that their speeds are different and dependent on the jobs processed on them.

A relatively simple HFS (e.g., a two-stage HFS with limited waiting time) is NP hard in the strong sense [3]. Also, Yang and Chern [27] indicated the problem with limited waiting time constraint and one single processor at each stage is NP hard. Subsequently, our problem has at least the same difficulty and three efficient metaheuristic algorithms are suggested to solve the test problems.

3.2 Notations

The following notations are applied to develop a mathematical model for the addressed problem.

3.2.1 Input parameters

S	Number of processing stages.
N	Number of jobs.
m_i	Number of unrelated machines at stage i.
R_k	Ready time of job k.
p_{ijk}	Processing time of job k on processor j at stage i.
u_{ik}	Waiting time's upper bound of job k between two consecutive stages i and $i+1$.
V	A very big number.

3.2.2 Indices

i	Processing stage, where $i=1, \ldots, S$.
j	Processor in stage i, where $j=1, \ldots, m_i$.
k, l	Job, where $k, l=1, \ldots, N$.

3.2.3 Decision variables

c_{max}	Makespan.
c_{ijk}	Completion time of job k at stage i on processor j.
c_{ik}	Completion time of job k at stage i.
B_{ijk}	Beginning time of job k at stage i on processor j.
x_{ijlk}	Binary variable that take value 1 if job k is assigned to processor j at stage i and here job l is its predecessor job; otherwise, $x_{ijlk}=0$.
g_{ijk}	The idle time of processor j at stage i that waits to process job k after the complete time of processing the predecessor, job l.

We define a dummy job 0 with zero process time that must be placed before the first jobs on each machine [25]. Also the jobs which skip some stages are assumed to have zero processing time to perform on those stages.

3.3 Mixed-integer nonlinear programming

The decision variables are assigning the machines to operations and determining the sequence of jobs at each stage so that the makespan is minimized. The mathematical model of the described problem is as follows:

Min $c_{\max}$

s.t.

$$\sum_{j=1}^{m_i} \sum_{\substack{l \in \{0,\ N\},\\ l \neq k}} x_{ijlk} = 1 \quad \forall i,k \tag{1}$$

$$\sum_{j=1}^{m_i} \sum_{\substack{l \in N,\\ l \neq k}} x_{ijkl} \leq 1 \quad \forall i,k \tag{2}$$

$$\sum_{j=1}^{m_i} \left(x_{ijlk} + x_{ijkl}\right) \leq 1 \quad k \in N,\ l = k+1, \ldots,\ N \quad ,\ l \neq k \quad i \in S \tag{3}$$

$$\sum_{k \in N} x_{ij0k} \leq 1 \quad i \in S, \quad j \in m_i \tag{4}$$

$$C_{i0} = 0 \quad i \in S \tag{5}$$

$$\begin{cases} B_{ijk} + V\left(1 - x_{ijlk}\right) \geq c_{ijl} + g_{ijk} \\ B_{ijk} - V\left(1 - x_{ijlk}\right) \leq c_{ijl} + g_{ijk} \end{cases} i \in S;\ j \in m_i;\ k \in N;\ l \in \{0, N\}, l \neq k \tag{6}$$

$$\begin{cases} c_{ijk} + V\left(1 - x_{ijlk}\right) \geq \mathrm{B}_{ijk} + p_{ijk} \\ c_{ijk} - V\left(1 - x_{ijlk}\right) \leq \mathrm{B}_{ijk} + p_{ijk} \end{cases} i \in S;\ j \in m_i;\ k \in N;\ l \in \{0, N\}, l \neq k \tag{7}$$

$$\sum_{j=1}^{m_i} \sum_{\substack{l \in \{0,\ N\},\\ l \neq k}} \left(x_{ijlk} \times c_{ijk}\right) \leq \sum_{j=1}^{m_{(i+1)}} \sum_{\substack{l \in \{0,\ N\},\\ l \neq k}} \left(x_{(i+1)jlk} \times B_{(i+1)jk}\right) \quad i \in S; \quad k \in N \tag{8}$$

$$\sum_{j=1}^{m_{(i+1)}} \sum_{\substack{l \in \{0,\ N\},\\ l \neq k}} \left(x_{(i+1)jlk} \times B_{(i+1)jk}\right) - \sum_{j=1}^{mi} \sum_{\substack{l \in \{0,\ N\},\\ l \neq k}} \left(x_{ijlk} \times c_{ijk}\right) \leq u_{ik} \quad i \in S; \quad k \in N \tag{9}$$

$$\begin{cases} B_{1jk} + V\left(1 - x_{1jlk}\right) \geq R_k \\ B_{1jk} - V\left(1 - x_{1jlk}\right) \leq R_k \end{cases} \quad j \in m_1; \quad k \in N; \quad l \in \{0, N\}, l \neq k \tag{10}$$

$$\begin{aligned} & c_{ik} = \sum_{j=1}^{m_i} \sum_{\substack{l \in \{0,\ N\}, \\ l \neq k}} \left(x_{ijlk} \times c_{ijk}\right) \quad i \in S; \quad k \in N \\ & c_{max} \geq c_{ik} \quad i \in S; \quad k \in N \end{aligned} \tag{11}$$

$$x_{ijlk} \in \{0, 1\} \quad i \in S; \quad j \in m_i; \quad l \in \{0,\ N\},\ l \neq k; \quad k \in N; \quad c_{ijk}, g_{ijk} \geq 0 \quad \forall i, k, m_i \tag{12}$$

The objective function is to minimize the maximum completion time. Constraint (1) ensures job k is assigned to only one machine for processing at each stage in a manner that job l is processed before it on the same machine. Constraint (2) is similar in the way that every job should have at most one successor at each stage. Constraint (3) avoids the occurrence of cross-precedence. Constraint (4) enforces that nominal job 0 is the predecessor of the first job placed on each processor. Constraint (5) expresses the dummy job 0 is completed at time 0 in all stages. Constraints set (6) calculate the beginning time of processing job k on processor j at stage i. This value is the sum of finishing time of the operation which is performed before job k on processor j and the idle time duration of that processor after completing job l. In fact, V as a large number converts an equal constraint into two unequal constraints in order to be imposed when x_{ijlk} is one and otherwise would be deceived. Constraints set (7) calculate the complete time of every job at each stage. Constraint (8) ensures that the processing of each job is begun after completing its operation at previous stage. Constraint (9) controls the waiting times at the intermediate storages. Actually, the waiting time of job k after its complete time at stage i until the beginning time of processing at stage i+1 cannot exceed its upper bound (u_{ik}). Constraints set (10) stipulate that the beginning time of each job in stage 1 should be after its ready time. Finally, constraint (11) defines the maximum completion time.

4 Proposed algorithms

This paper seems to be the first study in this field and justifies the BBO algorithm to solve the given problem.

In order to validate the performance of our proposed algorithm, we apply two other metaheuristics, namely PBSA and ICA. At first, these metaheuristics (i.e., BBO, PBSA, and ICA) are coded by using MATLAB 2009b. Then after solving several test problems, the results of their performances are compared together. In this section, the structure of BBO algorithm is elaborated. In addition, the executed procedures of ICA and BPSA are briefly explained.

4.1 Biogeography-based optimization

BBO proposed by Simon [31] is a new evolutionary optimization algorithm based on geographic distribution of biological organisms (i.e., biogeography theory). The BBO approach is nearly similar to genetic algorithm (GA) and particle swarm optimization (PSO). However, BBO has some features that has been unique among the biology-based algorithms. For example, the information is sharing between the suitable solutions and other solutions from one iteration to the next one.

The main core of BBO is based on migration and mutation. With probabilistic migration, BBO is able to share more information from good solutions to poor ones. In other words, this algorithm prevents the good solutions be destroyed during the evolution. Thus, it can efficiently utilize the characteristics and information of population in per iteration. This feature leads to find the better solutions in a short time rather than other metaheuristics. Additionally, the mutation operator increases the diversity among the population. Without this modification, the relatively good solutions will tend to be more dominant in population. The mutation approach makes both solutions with low or high habitat suitability index (HSI) likely to mutate and gives a chance of improving to both types of solutions in comparison to their earlier values [33].

Geographic distribution of biological organisms or biogeography theory describes the islands, in which their species migrate between them via flotsam, wind, flying, swimming, and the like, as shown in Fig. 1. Migration is related to the features of islands (e.g., rainfall, area, topography, diversity of

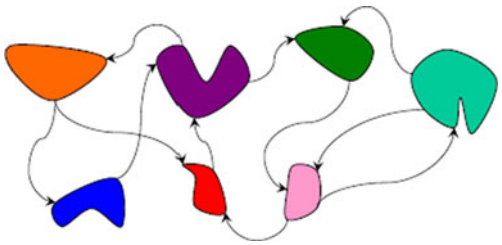

Fig. 1 Migration of spices [31]

vegetation, and temperature), which are called suitability index variables (SIVs) and affect the worth of HSI. Some islands are more suitable for habitation than others. Islands that are well suited as habitats for biological species are said to have a high HSI. In addition, islands with the high HSI possess a large number of species that emigrate to nearby islands and those with a low HSI have a smaller number of species.

The approach of the BBO algorithm is explained as follows:

Step 1: Initializing the BBO's parameters
Step 2: Generating the initial population in a number of initial population (n_{Pop})
Step 3: Calculating the HIS of each individual in the population
Step 4: Sorting the population from the best cost to the worst
Step 5: Selecting the elite solutions to keep in a number of n_{Keep}. Elite habitats are identified based on the value of HSI
Step 6: Calculating the immigration and emigration rates (λ_{i} and μ_i) for every individual X_i
Step 7: Modifying the population by the migration operator (all habitats are probabilistically modified by performing the migration process)
Step 8: Mutating the population by a mutation operator
Step 9: Evaluating the population and select the best of them
Step 10: Transferring the new population to next iteration

4.1.1 Initializing the BBO's parameters

Since the parameters of many metaheuristics play important roles in the quality of solutions. Without tuning parameters, algorithms may not work properly. So determining the appropriate values for parameters is needed. The procedure of tuning parameters is explained in Section 6.2. The BBO parameters include the maximum number of iteration ($\mathrm{Max_{It}}$), n_{Pop}, probability of modification ($p_{\mathrm{Modification}}$), n_{Keep}, and mutation probability (p_{Mutation}).

4.1.2 Generating the initial population

In this step, n_{Pop} solutions, each of which is called an "island" or "habitat" in BBO algorithm must be generated. Furthermore, each solution is shown by an array X_i (i=1, … , n_{Pop}) composed of SIVs. These variables accept the values which must be optimized. x_{ij} is the jth SIV in solution X_i.

In the GA terminology, this array is called "chromosome" and in ICA is expressed as "country"; while in BBO, "island or habitat" plays the same role. In an N-dimensional optimization problem, a habitat is a $1\times N$ array. This array is defined by:

$$\mathrm{Habitat} = (\mathrm{SIV}_1,\ \mathrm{SIV}_2,\ \mathrm{SIV}_3, \ldots,\ \mathrm{SIV}_{\mathrm{N}}) \tag{13}$$

These values of SIVs are defined by characteristics of each specific problem as well as should be optimized and are similar to gene in GA. Each variable in a habitat denotes the climatic conditions in that island such as rainfall, area, topography, diversity of vegetation, temperature, etc.

In coded BBO, each solution represents a sequence of jobs, which should be assigned to available machines. In order to reach this sequence, at first, the amounts of variables are generated randomly by uniform distribution function in a range between zero and one. Then, these values are interpreted by sorting of them. For a problem with five jobs, Fig. 2a indicates the initial structure and decoded structure is shown in Fig. 2b.

4.1.3 Evaluating the HSI for each individual

Each individual is considered as an "island" with a HSI to measure its own cost function. The variables of an island which are called SIVs characterize the habitability or evaluate the quality of each solution. In optimization problems, solutions with the high HSI have further quality rather than solutions with the low HSI.

The fitness function of each habitat is calculated as a function of variables (SIV_1, SIV_2, … , SIV_N) as follows:

$$c_i = f(\mathrm{Habitat}_i) = f(\mathrm{SIV}_{i1},\ \mathrm{SIV}_{i2}, \ldots,\ \mathrm{SIV}_{iN}) \tag{14}$$

4.1.4 Sorting the members of population and elitism of them

All solutions must be sorted based on their obtained fitness function values from the best cost to the worst ones. Then, n_{Keep} individuals are chosen among the elite habitats with the lesser costs.

4.1.5 Calculating the immigration and emigration rates for each individual

The emigration and immigration rates are used to share information between the habitats probabilistically [33]. In BBO, an island has its own immigration and emigration rates, μ and λ. These rates are the functions of number of species on an island. These functions are as follows:

$$\lambda_k = I\left(1 - \frac{k}{n}\right) \tag{15}$$

$$\mu_k = E\left(\frac{k}{n}\right) \tag{16}$$

(a) Initial structure

0.87	0.45	0.23	0.47	0.64

(b) decoded structure

5	2	1	3	4

Fig. 2 The structure of a solution with five jobs in BBO

where I and E are respectively the maximum possible immigration and emigration rates, k is the number of species in kth island and $n=S_{max}$ is the total species as shown in Fig. 3.

This method is one of the most common ways for computing λ and μ presented by Simon [31]. In our problem, the amounts of these rates for each member depend on its fitness function. In fact, a solution with a better objective function acquires a larger μ and smaller λ rather than a solution with a worse cost. In this procedure at first, all members of population are sorted from the best fitness function to the worst. Then according to the ranks which members acquire, μ and λ are calculated under Eqs. (17) and (18).

$$w_{\mu_i} = \frac{(r_i - 1)}{(1 - n_{pop})} + 1, \quad w_{\lambda_i} = 1 - w_{\mu_i}, \quad i = 1, \ldots, n_{pop} \tag{17}$$

$$\mu_i = \frac{w_{\mu_i}}{\sum_{i=1}^{n_{pop}} w_{\mu_i}}, \quad \lambda_i = \frac{w_{\lambda_i}}{\sum_{i=1}^{n_{pop}} w_{\lambda_i}} \tag{18}$$

where $w_{\mu i}$ and $w_{\lambda i}$ denote respectively the weight of emigration and immigration rates in solution i. Also r_i is the rank of solution i with regard to its acquired situation after sorting the objective functions. As an example in accordance with the illustrated procedure, the amounts of μ and λ for a population consists of six solutions are calculating and has been shown in Fig. 4.

4.1.6 Modifying the population by the migration operator

Based on the information and experiences of all solutions, each of them with a habitat $p_{Modification}$ is modified. For implementing this process, at first for each solution, a random number which is varying between zero and one is generated. If this number is lower than $p_{Modification}$, the procedure of migration for all of variables in that solution is performed. Then for every one of the variables in solution i such as jth variable (SIV_{ij}), the procedure of migration which is composed of the following steps should be done.

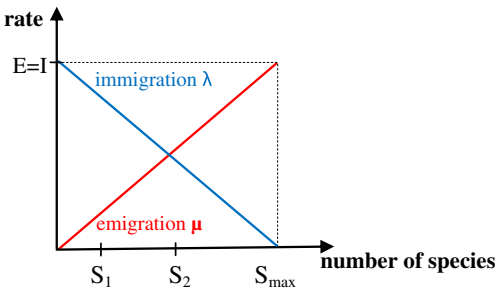

Fig. 3 The relationship between the number of species and migration rates [31]

individuals	rank	μ	λ
Solution 1	3	0.2	0.13
Solution 2	5	0.07	0.27
Solution 3	1	0.33	0
Solution 4	2	0.27	0.07
Solution 5	6	0	0.33
Solution 6	4	0.13	0.2

Fig. 4 The values of μ and λ for solutions in a population with six members

- A random number varying between zero and one is compared with the immigration rate of solution i (λ_i). If this amount is lower than λ_i, immigration process will be accepted. In reality, a weak solution has the further immigration rate (λ) and is more probable to receive the characteristics of the powerful members in population.
- It needs to select the jth variable from another solution such as jth variable of solution k (SIV_{kj}) that transfers the own characteristics to SIV_{ij}. In order to an appropriate selection, the roulette wheel method is applied on emigration rate (μ). It is obvious through the roulette wheel method, a powerful solution is more likely to select for emigrating its own information to other weak solutions. This procedure is shown in Fig. 5.

4.1.7 Mutating the population with the mutation operator

The migration procedure makes the high HSI solutions dominant in population and the low HSI solutions are eliminated after several iterations. Therefore, the mutation operator is applied to avoid the convergence of a local optimum and increases diversity among the members of a population [34]. In fact, the mutation approach probabilistically mutates both low and high HSI solutions and gives a chance to improve both kinds of the solutions in comparison to their earlier values. To this end, for all variables in solution i three following phases must be implemented.

- A random value between zero and one is compared with $p_{Mutation}$ to determine which SIVs are selected to mutate. As an example, Fig. 6b indicates variables in the 3rd and 4th positions that are selected for mutation.
- After the SIV_{ij} is chosen, a random number (dx) is generated uniformly from the interval ($-Max_{Variation}$

Fig. 5 The pseudo-code of migration procedure

```
"Create a new population"
for i = 1: npop do
    If random (0,1) < p_Modification then do
        for V_ij = 1: SIVs do
            If random (0,1) < λ_i then do
            Roulette Wheel method on μ is performed to select the another pop such as pop(k);
            Position var(j) of newpop(i) = Position var(j) of pop(k);
            end if
        end for
    end if
end for
```

and +$\text{Max}_{\text{Variation}}$). In this algorithm, $\text{Max}_{\text{Variation}}$ is considered 0.1 as shown in Fig. 6c.

- Then, the initial value of SIV_{ij} is replaced with the sum of SIV_{ij} and dx as shown in Fig. 6d.

Finally, the values of muted structure in Fig. 6d are interpreted by sorting of them according to Fig. 6e, which indicates a sequence of five jobs.

4.1.8 Evaluating the population

After the migration and mutation processes on the initial population, although some solutions might have not been changed, all of them have been evaluated and sorted from the best cost to the worst. The new population that transferring to next iteration consists of new and old habitats. New habitats are selected in this phase after sorting in the number of ($n_{\text{Pop}}-n_{\text{Keep}}$) and old habitats are the elites chosen from the initial population in a number of n_{Keep}. The final output in this iteration is the best solution which has the least cost. In addition, the next iterations pass this procedure.

4.1.9 Stopping criteria

Expiry criterion in our proposed algorithm is to get the Max_{It}, which is adjusted by tuning the BBO parameters.

4.2 Imperialist competitive algorithm

The ICA originates the socio-political evolution and has been modelled mathematically by Atashpaz-Gargary and Lucas [35]. The steps of this algorithm according to ICA presented by Attar et al. [36] are as follows:

1. *Generating the initial empires*—n_{Pop} solutions, which are named the initial countries, are randomly generated in a form of array $1\times N$. Then, these countries are divided into two groups. The first group is the most powerful members that are belonged to imperialist countries in a number of N_{imp} and the second group contains the remaining members in the name of colony countries. Based on the power of imperialists that are calculated using Eqs. (19) and (20), the colonies are randomly distributed between the empires. Therefore, the initial number of colonies at an empire is determined according to Eq. (21):

$$C_n = \max c_i - c_n, \quad i = 1, 2, \ldots, N_{imp} \tag{19}$$

$$p_n = \left| \frac{C_n}{\sum_{i=1}^{N_{\text{imp}}} C_i} \right|, \quad \sum_{i=1}^{N_{\text{imp}}} p_i = 1 \tag{20}$$

$$\text{NC}_\text{n} = \text{round}\{\text{p}_\text{n} \times \text{N}_\text{col}\} \tag{21}$$

Fig. 6 The mutation procedure

(a) Initial structure	**0.87**	**0.45**	**0.95**	**0.47**	**0.64**
(b) random values to define the mutation applying p_{Mutation}=0.3	0.54	0.86	0.13	0.25	0.38
(c) random number (dx) between (-0.1,+0.1)	-	-	0.09	-0.07	-
(d) mutated structure	**0.87**	**0.45**	**1.04**	**0.40**	**0.64**
(e) decoded structure	**4**	**2**	**5**	**1**	**3**

where c_n is the cost of nth imperialist and C_n is its normalized cost. p_n. and NC_n denote respectively power and the initial number of colonies for nth imperialist. Also, N_{col} is the number of all colonies.

2. *Assimilation*—imperialists try to improve all of their colonies. The aim of assimilation procedure is to assimilate the colony's characteristic such as culture, social structure, and language toward its imperialist. As shown in Fig. 7, each colony moves toward the imperialist by x units. x is a random number with uniform distribution (U (0, $\beta \times d$)), where β is a number greater than 1 and d is distance among the colony and imperialist which is the vector of movement for colony toward imperialist. Parameter β causes the colony to get closer to imperialist from both sides. To intensify the property of this method and search a wider area around the current solution, we add a random amount of deviation θ to the direction of movement. θ is a number with uniform distribution $(\theta \in (-\gamma, \gamma)$, where γ is a parameter that adjusts the deviation from the original path.
3. *Revolution*—this mechanism is similar to mutation process in genetic algorithm to create diversification among the solutions. In each iteration, for every colony a random number varying between zero and one is generated and compared with probability of revolution (P_R). If the random number is lower than P_R, the procedure of revolution is performed. We determine the number of variables which should be changed, through multiplying P_R in a number of jobs. These numbers of the elements are selected at random in a colony and their values are randomly changed. Finally, the new colony will be replaced with the previous colony while its cost is improved.
4. *Exchanging positions of the imperialist and the best colony*—after assimilation for all colonies and revolution for a percentage of them, the best colony in every empire is compared with its imperialist. If the best colony is better than its imperialist, the positions of them are exchanged.

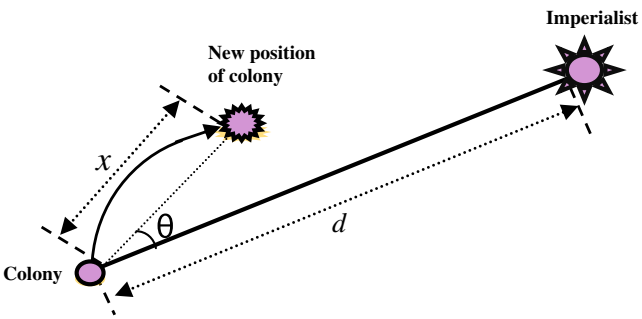

Fig. 7 Moving colonies direction

5. *Computing the total cost of an empire*—it is clear that the power of an empire is including the imperialist power and their colonies. Moreover, the power of imperialist has a main effect on total power of an empire while colonies have the lower impact. Hence, the equation of the total cost is defined as follows:

$$\begin{aligned} TC_n &= \text{cost}(\text{imperialist}_n) + \\ &\xi_{\text{mean}}\{\text{cost}(\text{colonies of empire}_n)\} \end{aligned} \tag{22}$$

where TC_n is the total cost of nth empire and zeta (ξ) is a positive number considered to be less than one. The total power of an empire is determined by just the imperialist when the value of ξ is small. The role of the colonies becomes more important since the value of ξ increases.

6. *Imperialistic competition*—imperialists try to increase their power by more possessing and controlling the colonies of other empires. For this purpose, the weakest colony in the weakest empire is determined to assign to other empires. At first, the possession probability of each empire is compute, the normalized total cost must be calculated by:

$$NTC_n = \max\{TC_i\} - TC_n \quad i = 1, 2, \ldots, N_{imp} \tag{23}$$

where NTC_n and TC_n are respectively the normalized total cost and total cost of nth empire. At last, the possession probability of each empire is calculated according to Eq. (24). We use the roulette wheel method for assigning the mentioned colony to an empire.

$$p_{\text{emp}_n} = \left| \frac{NTC_n}{\sum_{i=1}^{N_{imp}} NTC_i} \right| \tag{24}$$

7. *Eliminating the powerless empires*—when each empire loses all of colonies, will collapse and its imperialist as a colony is assigned to other empires. This approach should be continued until one empire is remained.
8. *Stopping criteria*—the stopping criterion for each test problem is to get the CPU time which is recorded after solving by proposed BBO algorithm.

4.3 Population-based simulated annealing

SA is one of well-known metaheuristics to solve many nonpolynomial hard optimization and operations research problems. It was introduced by Metropolis et

Table 1 The processing stages of jobs

Job	1	2	3	4	5	6
Stage	[1, 2, 3]	[1, 2, 3]	[1, 2, 3]	[1, 2, 3]	[1, 2]	[1]

al. [37] and presented by Kirkpatrick et al. [38]. We apply PBSA which has a similar structure to SA; however, it proceeds with several initial solutions. In fact, the further members in the initial population make to increase the diversification and overcome the local optimal points. Consequently, the more accurate solutions and the global optimum point will be acquired. This algorithm is composed of the following phases:

1. The algorithm starts with the initial solutions, which are randomly selected from search area in a number of the primary population. The fitness functions of population are calculated and the temperature of system is set. The starting temperature (T_0) must be adjusted by attention to the dimension and nature of the investigated problem. Because if T_0 is very hot, the algorithm will proceed slowly and extends the search area. Besides if T_0 is very low, the algorithm may stay in local-optimal search and will lead to untimely convergence. By using an annealing schedule, the process of searching will be continued until temperature decreases to final temperature (T_f). Annealing schedule defines how the temperature is changed during the annealing process. It should be noted that the decrease of T_f until zero is usual.
2. Each of the solutions finds a neighborhood for itself to create a new population. There are three approaches to generate a neighborhood that described below.

   - Swap—the positions of two selected jobs in an array are exchanged.
   - Reversion—in this policy besides conducting swap, the jobs located in between the swapped jobs are reversed as well.
   - Insertion—in this case, the job in the second position is located immediately after the job in the first location and the other jobs are shifted right hand accordingly.

3. Algorithm replaces the new neighborhood of solution q with solution q if there is an improvement in the fitness. But if there is no improvement in the cost function, algorithm accepts the new neighborhood with a probability in order to escape the local-optimal solutions, this probability is obtained from Boltzman distribution computed by:

$$P(\Delta E) = e^{\frac{-\Delta E}{KT_i}} \tag{25}$$

   where ΔE is the difference between cost of the old and new states, T_i denotes the temperature of the system at ith iteration and k is a constant parameter which in this paper is equal to one.
4. Finally, all members of the new population are sorted based on their fitness values and the best of them is the output of this iteration. At the end of each iteration, the temperature of system must be updated. There are many cooling approaches, such as geometric or linear decreasing, hyperbolic and exponential functions. We consider

Table 2 The processing times of operations

Job	$\frac{S}{m_i}$				
	1		2	3	
	1	2	1	1	2
1	13	3	15	13	14
2	10	8	5	14	9
3	4	5	10	7	4
4	16	16	11	10	5
5	16	16	14	–	–
6	12	15	–	–	–

Table 3 The waiting time's upper bound between two consecutive stages

$S(i-i+1)$	Job					
	1	2	3	4	5	6
(0–1)	0	0	0	0	0	0
(1–2)	4	2	3	1	1	–
(2–3)	2	0	0	3	–	–

Table 4 The ready times of jobs

Job	1	2	3	4	5	6
Ready time	6	8	5	7	13	9

Fig. 8 The pseudo-code of fitness evaluation

Create a random sequence of jobs ;
Set t_{ijk} = The earliest beginning time of job kth on machine j at stage i;
for i=1 to S **do**
for k=1 to N **do**
for j=1 to m_i **do**
Set $t_{ij} = x_{ijlk} \times c_{ijl}$; % The availability time of machine jth at stage i;
if i=1 **do**
$t_{1jk} = max\ \{R_k, t_{1j}\}$;
else
$t_{ijk} = max\ \{C_{(i-1)k}, t_{ij}\}$;
end if
$j := min\ (t_{ijk} + p_{ijk})$ % Assign the best machine to job kth at stage i.
end for

"using linprog function of MATLAB"

$Min\ c_{ik}$ %Calculate the best start time of job kth on the determined machine at stage i.
s.t.
$t_{ijk} + g_{ijk} = b_{ijk}$;
$b_{(i+1)jk} - C_{ik} <= u_{ik}$;
$b_{(i+1)jk} \geq C_{ik}$;
end for
end for

the linear function $T_i = \alpha T_{i-1}$ to update the temperature at each time. α represents a positive constant number which is lesser than one and named cooling factor. Also, T_i is the temperature of the system at ith iteration. In order to find a better solution, the described phases should be executed Max_{ipt} (i.e., maximum iteration per temperature) times at each temperature.

5. The next iterations are implemented according to the above phases. Each iteration starts with the population acquired at the third phase of its previous iteration. SA usually terminates when T_0 decreases to T_f after several iterations. The termination criterion in this algorithm, similar to ICA, for every kind of test problems is to meet the CPU time that is acquired through running on the BBO algorithm.

5 Numerical example

In this section, an example is illustrated in order to make the described problem easier to comprehend. This example is considered with six jobs, three stages so that there are two machines in either first or third stages, while only one

Table 5 The beginning times (b_{ijk}) of operations

Stage	Job					
	1	2	3	4	5	6
1	6	12	32	12	34	22
2	9	24	40	29	50	–
3	24	29	50	40	–	–

Table 6 The number of machine which assigned to each operation

Stage	Job					
	1	2	3	4	5	6
1	2	1	2	2	1	1
2	1	1	1	1	1	–
3	1	2	1	2	–	–

Table 7 The waiting time in the intermediate buffers

Stage	Job					
	1	2	3	4	5	6
1	0	0	0	0	0	0
2	0	2	3	1	0	–
3	0	0	0	0	–	–

machine exists in the second stage (i.e., n=6, S=3, m_1=2, m_2=1, and m_3=2).

Table 1 indicates jobs 1, 2, 3, and 4 must pass through all three stages whereas job 5 skips stage 3 and job 6 skips stages 2 and 3. Table 2 shows the processing times of operations. There are the unrelated parallel machines in each stage so that they can process one type of operation at the different duration of times. Table 3 displays the upper bounds of waiting times between every two consecutive operations with this assumption that stage 0 is considered.

Table 4 shows the ready times of jobs, "times at which the jobs are available for processing." The decision variables dealt in this problem are the sequence of jobs on each machine, machines assignment to operations and beginning times of operations in a manner that $C_{\max}$ is minimized.

This example is solved by the proposed BBO algorithm coded using MATLAB 2009b software. Figure 8 depicts the used pseudo-code of fitness evaluation which is developed in two phases. As explained below, the first phase is trying to obtain the best response through searching the solution area. Also, second phase by applying the response of first phase, acquires the best result under solving the linear programming.

1. In first phase, by searching the solution area and applying a policy that provides the minimum completion time for an operation, the best sequence and machines allocation are gotten. Initially, a random sequence of jobs is created. According to sequencing of jobs and from the first stage to the last, the availability times of all machines (t_{ij}) are calculated, which are equal to completion time of the previous work on them. Also, the ready time of processing the intended job (t_{ijk}) is calculated by comparing its completion time in the previous stage or its release data at the workshop. By assuming the considered operation will be processed on all machines, its completion time is estimated. At last, the minimum of these estimates determines the best machine to process that job.
2. After determining the sequence of jobs and assignment of machines, second phase through solving the linprog function on MATLAB, calculates the best start times for all operations in all stages so that the limited waiting times constraint is met.

Table 8 The factors and their levels

Factors	Levels
Number of jobs	20, 50, 100
Number of stages	2, 4, 8
number of machines at per stage	Constant (4), variable (U (2, 8))
Processing time	U (25, 95)
P_{skip}	0.1, 0.4
LWT	U (0, 20)
Ready time	U (0, 50)

Consequently, the best solution is obtained in a sequence of q={1, 2, 4, 6, 3, and 5} with the cost function $C_{\max}$=64. All outputs are shown in Tables 5, 6, and 7. Table 5 exhibits the beginning times of operations. Table 6 denotes the machine assigned to each

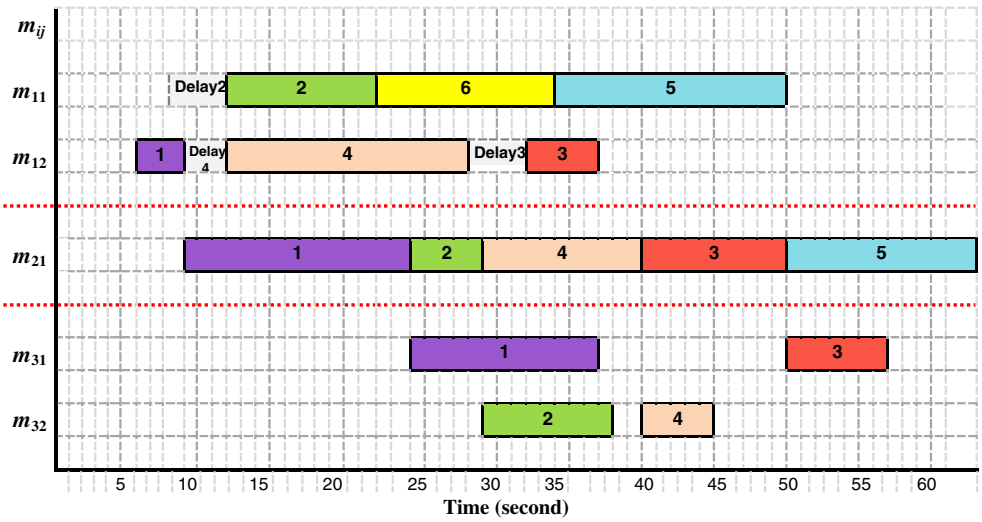

Fig. 9 Gantt chart for an example problem, $C_{\max}$=64

Table 9 The levels of parameters of BBO algorithm

Parameters	Levels	
	Lower	Upper
Max_{It}	100	150
n_{Pop}	40	60
n_{Keep}	$0.1 \times n_{Pop}$	$0.15 \times n_{Pop}$
$p_{Mutation}$	0.05	0.1

operations. Table 7 presents the duration of times that jobs must be waited at the intermediate buffers. Additionally, the values of this matrix must be lower than the matrix displayed in Table 3 in order to satisfy the limited waiting times constraint.

Table 10 The results of experiments

No.	Max_{It}	n_{Pop}	n_{Keep}	$p_{Mutation}$	Response
1	125	50	0.13	0.10	1,339
2	100	40	0.10	0.10	1,341
3	125	50	0.13	0.05	1,319
4	150	40	0.15	0.05	1,328
5	150	40	0.10	0.05	1,352
6	125	50	0.13	0.08	1,309
7	150	40	0.10	0.10	1,305
8	150	60	0.10	0.10	1,310
9	100	50	0.13	0.08	1,331
10	100	60	0.15	0.10	1,319
11	100	40	0.10	0.05	1,390
12	150	60	0.10	0.05	1,325
13	125	50	0.13	0.08	1,367
14	125	50	0.15	0.08	1,331
15	125	40	0.13	0.08	1,274
16	125	50	0.13	0.08	1,352
17	150	60	0.15	0.10	1,348
18	125	50	0.13	0.08	1,325
19	100	60	0.15	0.05	1,317
20	150	60	0.15	0.05	1,321
21	100	40	0.15	0.10	1,323
22	125	50	0.13	0.08	1,317
23	125	60	0.13	0.08	1,328
24	100	60	0.10	0.10	1,322
25	100	40	0.15	0.05	1,390
26	125	50	0.13	0.08	1,362
27	150	50	0.13	0.08	1,326
28	100	60	0.10	0.05	1,312
29	125	50	0.10	0.08	1,347
30	150	40	0.15	0.10	1,309

Table 11 The optimum values of BBO parameters

Max_{It}	n_{Pop}	n_{Keep}	$p_{Mutation}$	Predicted response
150	40	0.15	0.1	1,300.772

Figure 9 illustrates the acquired results as a Gantt chart. It can be observed that there is at stage one the available machines earlier than the beginning times of jobs 2, 3, and 4. In reality, the processing of these jobs is postponed in order to satisfy the limited waiting times constraint. Therefore, after several iterations, the proposed BBO algorithm gives a near-optimal solution so that the maximum completion time is minimized.

6 Evaluation method

The proposed metaheuristics for our problem are coded by using MATLAB 2009b on a personal computer with a Pentium (R) Dual-core, CPU E5700, 3.00 GHz and 3.25 GB of RAM memory under the Microsoft Windows XP. The ICA and SA are among the most popular algorithms in all of optimization and combinatorial optimization problems. Some researches in many kinds of scheduling problems have applied ICA [2, 39–41] and SA (PBSA) or modified versions of SA [20, 42–47] for solving their problems.

Hence, for assessing the performance of all three proposed algorithms, 36 various test problems are generated. These problems are in different dimensions, small, medium and large sizes that include 20, 50, and 100 jobs, respectively.

Table 12 The obtained $\overline{RPD}$ of results by algorithms

Instance	Algorithms		
$n \times m$	PBSA	ICA	BBO
20 jobs	5.179874	8.173175	0.393349
20×2	6.641099	11.37652	0.090909
20×4	6.457202	7.876704	0.853514
20×8	2.441323	5.266298	0.235624
50 jobs	5.196064	6.395222	0.012681
50×2	4.928611	4.994669	0
50×4	4.008929	5.817911	0.038043
50×8	6.650652	8.373086	0
100 jobs	3.518237	3.7845	0.015247
100×2	2.865093	3.548803	0
100×4	2.539178	2.434507	0.045741
100×8	5.150441	5.37019	0
Average	3.796355	3.995606	0.015812

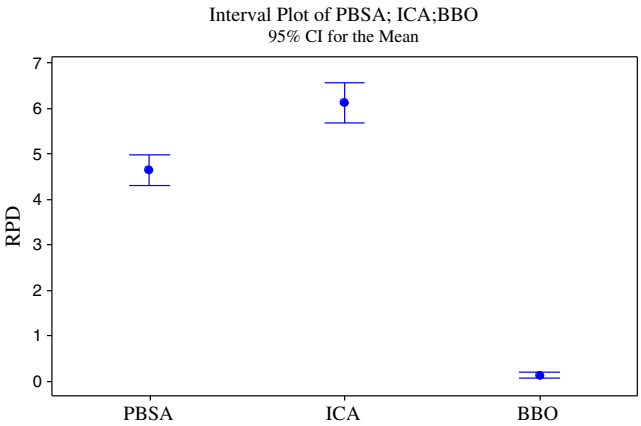

Fig. 10 Means plot and Tukey's intervals (at 95 % confidence level) for the type of algorithms

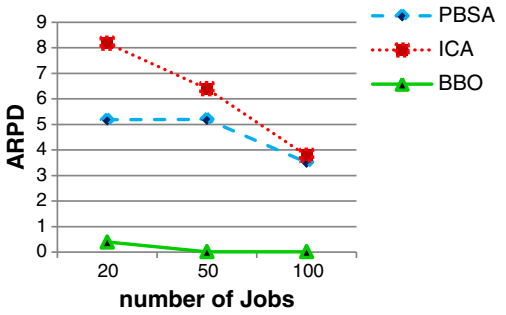

Fig. 11 Plots of $\overline{\text{RPD}}$ for the interaction among the type of algorithm and n

Each of the test problems is run ten times separately on all three metaheuristics. Then for every problem and algorithm, the average relative percentage deviation $\left(\overline{\text{RPD}}\right)$ is obtained from the results of ten runs. At last, the obtained $\overline{\text{RPD}}$ on all three algorithms are compared with together.

6.1 Data generation

The required data to randomly generate the test problems consists of the number of jobs, stages and machines in each stage, also, processing times, LWT and job's ready time distribution functions and probabilities of skip. These factors and their levels are demonstrated in Table 8. All combinations of these levels lead to make the various test problems.

6.2 Parameter tuning

The quality of the metaheuristics is significantly affected by the values of parameters. Therefore, we need to find a best combination of parameters in the BBO, ICA, and PBSA algorithms in order to achieve the better results. So, for each of the algorithms, a regression analysis between the different values of parameters and their responses is implemented on Design-Expert software.

The response surface methodology (RSM) proposed by Myers [48] is a comprehensive experimental design and mathematical method. RSM fits the relationships of the results into a regression equation and is effective to determine the best condition.

In comparison with other parameters calibration methods (e.g., Taguchi method [49]), one of the advantages of RSM is considering all main and interaction effects between the parameters. Hence, this method obtains better values for parameters.

In order to find the best possible combination of parameters, the estimated regression function along with the determined range of parameters is minimized by Lingo software. As an instance, the procedure of parameter tuning for proposed BBO algorithm is described in detail. The BBO parameters and their levels are shown in Table 9.

It is noticeable that $p_{\text{Modification}}$ is considered equal to one in order to modify all members of population. A medium sized problem is investigated to tune the parameters. This problem consists of 50 jobs, 4 stages so that the number of machines at each stage is generated uniformly from the interval [2, 8] and probability of skip is supposed to 0.4. The RSM designs 30 experiments for these four parameters that contain 6 central and 24 axial points. In each experiment, the BBO parameters are changing in their corresponding ranges to run the addressed problem. Table 10 indicates these 30 experiments and their obtained responses.

The final estimated regression equation based on the results has been obtained as Eq. (26). After solving the following model by Lingo, the optimum values are shown in Table 11.

$$\begin{aligned} f\left(\text{Max}_{\text{It}},\ n_{\text{Pop}},\ n_{\text{Keep}},\ p_{\text{Mutation}}\right) = {} & 2,154.71667 - 3.31889 \times \text{Max}_{\text{It}} - 12.53611 \times \text{n}_{\text{Pop}} - 1,670\times \\ & n_{\text{Keep}} - 4,056.66667 \times p_{\text{Mutation}} + 0.046 \times \text{Max}_{\text{It}} \times \text{n}_{\text{Pop}} + \\ & 3 \times \text{Max}_{\text{It}} \times \text{n}_{\text{Keep}} + 5 \times \text{Max}_{\text{It}} \times \text{p}_{\text{Mutation}} + 18.5\times \\ & n_{\text{Pop}} \times n_{\text{Keep}} + 51.5 \times n_{\text{Pop}} \times p_{\text{Mutation}} + 4,400 \times n_{\text{Keep}} \times p_{\text{Mutation}} \end{aligned} \tag{26}$$

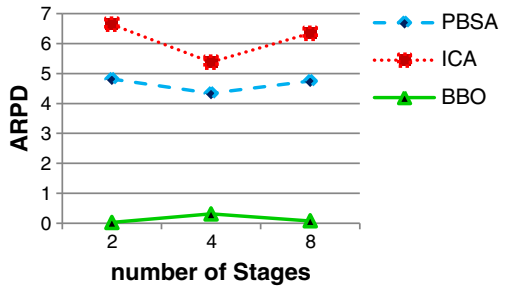

Fig. 12 Plots of $\overline{\text{RPD}}$ for the interaction among the type of algorithm and S

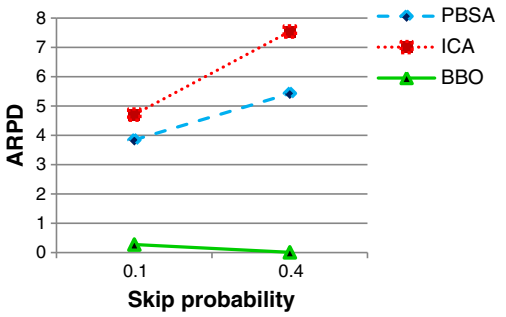

Fig. 14 Plots of $\overline{\text{RPD}}$ for the interaction among the type of algorithm and P_{skip}

Subject to:

$$100 \le \text{Max}_{\text{It}} \le 150$$
$$40 \le n_{\text{Pop}} \le 60$$
$$0.1 \times n_{\text{Pop}} \le n_{\text{Keep}} \le 0.15 \times n_{\text{Pop}}$$
$$0.05 \le p_{\text{Mutation}} \le 0.1$$
$$\text{Max}_{\text{It}},\ n_{\text{Pop}}, \text{ and } n_{\text{Keep}}/\text{integer}$$

Based on the results of parameter tuning by executing the above approach, the values of parameters in ICA including n_{Pop}, N_{imp}, ξ, and P_R are set respectively to 40, 4, 0.15, and 0.3. Also, the parameters of PBSA algorithm (e.g., n_{Pop}, T_0, T_f, α, and Max_{ipt}) are adjusted to 3, 10,000, 0, 0.995, and 2, respectively. Indeed, the Max_{It} of ICA and PBSA are when the algorithm is reached the set time. The set time is the same termination criterion of ICA and PBSA as described in Section 4.

6.3 Evaluating results

After tuning the parameters, at first, each random test problem is solved on BBO and its CPU time is recorded. So, the other algorithms are tuned based on the acquired CPU time to run the same problem. Each kind of the problems is run at a specific time by all three algorithms.

The benchmarks applied to evaluate the performance of these metaheuristics are the relative percentage deviation (RPD) and the $\overline{\text{RPD}}$. Thus, the results obtained by the algorithms are converted to RPD and $\overline{\text{RPD}}$ according to Eqs. (27) and (28) defined by:

$$\text{RPD} = \frac{|\text{Algorithm}_{\text{sol}} - \text{Best}_{\text{sol}}|}{\text{Best}_{\text{sol}}} \times 100 \tag{27}$$

$$\text{ARPD} = \overline{\text{RPD}} = \frac{\sum_{i=1}^{\text{number of run}} \text{RPD}_i}{\text{number of run}} \tag{28}$$

where $\text{Algorithm}_{\text{sol}}$ is the makespan obtained by an algorithm for a given problem and Best_{sol} is a minimum makespan among all results of ten runs for that problem.

The comparison of the performances of applied metaheuristics is shown in Table 12. Furthermore, we implement the sensitive analysis on the number of jobs, stages and number of machines per stage, as well as skip probability to achieve a more efficient algorithm from the various points of views.

To assess the algorithms, Tukey's 95 % confidence intervals of algorithms are conducted on the RPD of the results. This analysis is illustrated in Fig. 10 that indicates the BBO algorithm is significantly superior to ICA and PBSA. Additionally, BBO has the lesser deviation range.

The sensitive analysis is utilized on the four controlled factors to prove the performance of BBO algorithm from the various points of views. Figure 11 shows the interaction between type of algorithm and the number of jobs using $\overline{\text{RPD}}$. Figure 12 displays the effect of number of stages on the quality of algorithms. Both Figs. 11 and 12 indicate that for every number of jobs and stages, the proposed BBO algorithm acts better than ICA and PBSA. Moreover, PBSA generally works better than ICA especially in a lesser number of jobs.

Figures 13 and 14 respectively exhibit the effect of the distribution of machine number and skipping probability on the quality of algorithms. As is obvious, in all cases, the BBO algorithm is more advanced to ICA and PBSA.

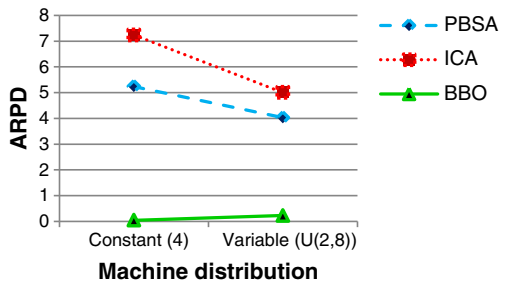

Fig. 13 Plots of $\overline{\text{RPD}}$ for the interaction among the type of algorithm and O_i

7 Conclusion and future study

In this paper, a HFFS problem with some constraints including the existence of unrelated parallel machines, limited waiting times between every two successive operations and the job release date was investigated. We first formulated this problem by a mathematical model in the form of a mixed-integer nonlinear program. Furthermore, a novel BBO method, ICA and PBSA were employed to minimize the makespan. To evaluate the performance of the presented techniques, we conducted the experiments and statistical analyses on them. The evaluating results illustrated that the proposed BBO algorithm had the ability to find the better-quality solution as well as convergence characteristics rather than PBSA and ICA. Since the efficiency of BBO was shown in our study, it is recommended to take the advantages of this algorithm for other combinatorial optimization problems or develop the new metaheuristics for this problem. A possible and natural extension for this paper is the considering more realistic assumptions, such as machine availability constraint, preventive maintenance, and due dates. A limited buffer capacity between the machines or blocking environment may be explored as well. In addition, multiobjective cases and other criteria, for example, the total weighted earliness and tardiness, flow time, and the maximum lateness of the problem can be developed.

References

1. Pinedo M (1995) Scheduling: theory, algorithms, and systems. Prentice-Hall, New Jersey
2. Behnamian J, Zandieh M (2011) A discrete colonial competitive algorithm for hybrid flowshop scheduling to minimize earliness and quadratic tardiness penalties. Expert Syst Appl 38(12):14490–14498
3. Su LH (2003) A hybrid two-stage flow-shop with limited waiting time constraints. Comput Ind Eng 44:409–424
4. Lin HT, Liao CJ (2003) A case study in a two-stage hybrid flow shop with setup time and dedicated machines. Int J Prod Econ 86(2):133–143
5. Low CY, Hsu CJ, Su CT (2008) A two-stage hybrid flowshop scheduling problem with a function constraint and unrelated alternative machines. Comput Oper Res 35(3):845–853
6. Bertel S, Billaut JC (2004) Agenetic algorithm for an industrial multiprocessor flow shop scheduling problem with recirculation. Eur J Oper Res 159(3):651–662
7. Garey MR, Johnson DS (1979) A guide to the theory of NP-completeness. Computers and intractability. Freeman, San Francisco
8. 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 10:1293–1300
9. Oğuz C, Zinder Y, Do VH, Janiak A, Lichtenstein M (2004) Hybrid flow shop scheduling problems with multiprocessor task systems. Eur J Oper Res 152:115–131
10. Oğuz C, Ercan MF (2005) A genetic algorithm for hybrid flow shop scheduling with multiprocessor tasks. J Sched 8:323–351
11. Oğuz C (2006) Data for hybrid flow-shop scheduling with multiprocessor tasks. Available from http://home.ku.edu.tr/~coguz/
12. Kyparisis GJ, Koulamas C (2006) Flexible flow shop scheduling with uniform parallel machines. Eur J Oper Res 168:985–997
13. Gholami M, Zandieh M, Alem-Tabriz A (2009) Scheduling hybrid flow shop with sequence-dependent setup times and machines with random breakdowns. Int J Adv Manuf Technol 42(1):189–201
14. Allahverdi A, Al-Anzi FS (2006) Scheduling multi-stage parallel-processor services to minimize average response time. J Oper Res Soc 57(1):101–110
15. Ying KC, Lin SW (2006) Multiprocessor task scheduling in multistage hybrid flowshops: an ant colony system approach. Int J Prod Res 44(16):3161–3317
16. Verma SK, Dessouky MI (1999) Multistage hybrid flowshop scheduling with identical jobs and uniform parallel machines. J Sched 2:135–150
17. Tang L, Zhang Y (2005) Heuristic combined artificial neural networks to schedule hybrid flow shop with sequence dependent setup times. Advances in Neural Networks-ISNN 2005, pp. 315–364
18. Wang X, Tang L (2009) A tabu search heuristic for the hybrid flowshop scheduling with finite intermediate buffers. Comput Oper Res 36:907–918
19. Kurz ME, Askin RG (2004) Scheduling flexible flow lines with sequence-dependent setup times. Eur J Oper Res 159:66–82
20. Naderi B, Zandieh M, Shirazi M (2009) Modeling and scheduling a case of flexible flowshops: total weighted tardiness minimization. Comput Oper Res 57:1258–1267
21. Low CY (2005) Simulated annealing heuristic for flow shop scheduling problems with unrelated parallel machines. Comput Oper Res 32(8):2013–2025
22. 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
23. Jungwattanakit J, Reodecha M, Chaovalitwongse P, Werner F (2009) A comparison of scheduling algorithms for flexible flowshop problems with unrelated parallel machines, setup times and dual criteria. Comput Oper Res 36:358–378
24. Tavakkoli-Moghaddam R, Safaei N (2006) Modeling flexible flow lines with blocking and sequence dependent setup time. In: Proceeding of the 5th International Symposium on Intelligent Manufacturing Systems (IMS2006), Sakarya, Turkey, May 29–31, pp. 149–158
25. Ruiz R, Şerifoğlu FS, Urlings T (2008) Modeling realistic hybrid flexible flowshop scheduling problems. Comput Oper Res 35:1151–1175
26. Chang J, Ma G, Ma X (2006) A new heuristic for minimal makespan in no-wait hybrid flowshops. Control Conference, 2006. CCC 2006. Chinese, Harbin, China, pp. 1352–1356
27. Yang DL, Chern MS (1995) A two-machine flowshop scheduling problem with limited waiting time constraints. Comput Ind Eng 28(1):63–70
28. Yan Li, Tieke Li (2007) Constructive backtracking heuristic for hybrid flowshop scheduling with limited waiting times. In: International Conference on Wireless Communications, Networking and Mobile Computing, 2007. WiCom 2007, Shanghai, China, pp. 6671–6674
29. Chen JS (2006) Model formulations for the machine scheduling problem with limited waiting time constraints. J Inf Optim Sci 27(1):225–240
30. Liu S, Cui J, Li Y (2008) Heuristic-tabu algorithm for hybrid flowshop scheduling with limited waiting time. In: International Symposium on Computational Intelligence and Design, ISCID '08, Wuhan, China, pp. 233–237
31. Simon D (2008) Biogeography-based optimization. IEEE Trans Evol Comput 12:702–713
32. Ruiz R, Vázquez-Rodríguez JA (2010) The hybrid flow shop scheduling problem. Eur J Oper Res 205:1–18

33. Roy PK, Ghoshal SP, Thakur SS (2010) Biogeography-based optimization technique for multi-constrained economic load dispatch problems. Int J Recent Trends Eng Technol 3(3):77–81
34. Behnamian J, FatemiGhomi SMT (2011) Hybrid flowshop scheduling with machine and resource- dependent processing times. Appl Math Model 35(3):1107–1123
35. Atashpaz-Gargari E, Lucas C (2007). Imperialist competitive algorithm: an algorithm for optimization inspired by imperialist competitive. In: IEEE Congress on Evolutionary Computation, Singapore, pp. 4661–4667
36. Attar SF, Mohammadi M, Tavakkoli-Moghaddam R (2011) A novel imperialist competitive algorithm to solve flexible flow shop scheduling problem in order to minimize maximum completion time. Int J Comput Appl 28(10):27–32
37. Metropolis N, Rosenbluth AW, Rosenbluth MN, Teller AH, Teller E (1953) Equation of state calculations by fast computing machines. J Chem Phys 21(6):1087–1092
38. Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680
39. Bagher M, Zandieh M, Farsijani H (2011) Balancing of stochastic U-type assembly lines: an imperialist competitive algorithm. Int J Adv Manuf Technol 54(1):271–285
40. Forouharfard S, Zandieh M (2010) An imperialist competitive algorithm to schedule of receiving and shipping trucks in cross-docking systems. Int J Adv Manuf Technol 51:1179–1193
41. Jolai F, Rabiee M, Asefi M (2012) A novel hybrid meta-heuristic algorithm for a no-wait flexible flow shop scheduling problem with sequence dependent setup times. Int J Prod Res 50(24):7447–7466
42. Elmi A, Solimanpur M, Topaloglu S, Elmi A (2011) A simulated annealing algorithm for the job shop cell scheduling problem with intercellular moves and reentrant parts. Comput Ind Eng 61:171–178
43. Gaafar LK, Masoud SA (2005) Genetic algorithms and simulated annealing for scheduling in agile manufacturing. Int J Prod Res 43(14):3069–3085
44. Lin SW, Gupta JND, Ying KC, Lee ZJ (2009) Using simulated annealing to schedule a flowshop manufacturing cell with sequence-dependent family setup times. Int J Prod Res 47(12):3205–3217
45. Mcmullen PR, Gregory FV (2000) A simulated annealing approach to mixed model sequencing with multiple objectives on a just-in-time line. IIE Trans 32(8):679–686
46. Vahdani B, Zandieh M (2010) Scheduling trucks in cross-docking systems: robust meta-heuristics. Comput Ind Eng 58:12–24
47. Zhou E, Chen X (2010) A new population-based simulated annealing algorithm. Winter Simulation Conference (WSC). In: Proceedings of the 2010, Baltimore, USA, pp. 1211–1222
48. Myers WR (2003) Encyclopedia of biopharmaceutical statistics. Marcel Dekker, New York
49. Ross RJ (1989) Taguchi techniques for quality engineering. McGraw-Hill, New York