Neural Computing and Applications (2021) 33:1559–1573
https://doi.org/10.1007/s00521-020-05055-7

ORIGINAL ARTICLE

An improved bat optimization algorithm to solve the tasks scheduling problem in open shop

Morteza Babazadeh Shareh[1] · Shirin Hatami Bargh[2] · Ali Asghar Rahmani Hosseinabadi[3] · Adam Slowik[4]

Received: 14 January 2020 / Accepted: 3 June 2020 / Published online: 16 June 2020

Abstract

The open shop scheduling problem involves a set of activities that should be run on a limited set of machines. The purpose of scheduling open shops problem is to provide a timetable for implementation of the entire operation so that the total execution time is reduced. The tasks scheduling problem in open shops is important in many applications due to the arbitrariness of the processing sequence of each job and lack of a prioritization of its operations. This is an NP-hard problem and obtaining an optimal solution to this problem requires a high time complexity. Therefore, heuristic techniques are used to solve these problems. In this paper, we investigate the tasks scheduling problem in open shops using the Bat Algorithm (BA) based on ColReuse and substitution meta-heuristic functions. The heuristic functions are designed to increase the rate of convergence to the optimal solution. To evaluate the performance of the proposed algorithm, standard open shop benchmarks were used. The results obtained in each benchmark are compared with those of the previous methods. Finally, after analyzing the results, it was found that the proposed BA had a better performance and was able to generate the best solution in all cases.

Keywords Open shop · Scheduling · Makespan · Bat algorithm · Heuristic functions

1 Introduction

The scheduling and sequencing of operations is, in fact, the optimal allocation of limited resources to activities over time [1]. Due to its importance and practical application, extensive research has been carried out in this area since the early 1950s. Suppose machine ($i = 1, 2, \ldots, m$) must process n work ($j = 1, 2, \ldots, n$). A schedule is the allocation of time periods for processing these jobs on machines. Any schedule can be displayed on a Gantt chart [2]. An efficient and suitable schedule will lead to increased profitability, reduced costs, reduced time required to accomplish activities and winning the customers' confidence. In most manufacturing factories and service companies, timely delivery of customers' orders or timely delivery of services is important. Costs of delay in these problems not only lead to customer loss, but it also reduces the credit of service companies or manufacturing factories. Therefore, attention to scheduling problems is important in many management issues and planning principles. Workshop environments including workshop jobs and workshop flows are used in many industrial and service processes [3–5]. An open shop scheduling problem (OSSP) environment is a workspace environment in which there is no operation-dependent sequence. Therefore, it has a wider solution space than other workspace environments. As a result, less attention

✉ Adam Slowik
aslowik@ie.tu.koszalin.pl

Morteza Babazadeh Shareh
babazadeh@baboliau.ac.ir

Shirin Hatami Bargh
sh.hatami@mubabol.ac.ir

Ali Asghar Rahmani Hosseinabadi
a.r.hosseinabadi1987@gmail.com

1 Department of Computer Engineering, Islamic Azad University, Babol Branch, Babol, Iran

2 Babol University of Medical Sciences, Babol, Iran

3 Young Researchers and Elite Club, Ayatollah Amoli Branch, Islamic Azad University, Amol, Iran

4 Department of Electronics and Computer Science, Koszalin University of Technology, Koszalin, Poland

has been paid to the open shop problem (OSP) compared to other workspace environments [6]. The OSSP was first introduced in major car repair garages [7]. It has other applications, most notably quality control centers, semiconductor manufacturing centers, teacher classroom assignments, inspection schedules and satellite communications [8–10]. Scheduling problems are one of the most important issues of production planning. Scheduling is a resource allocation process limited to activities over time [11]. Scheduling problems are formulated in the form of machines and jobs (activities), where machines represent resources and jobs represent activities that need to be done using these resources [12]. The OSP is one such problem. In fact, it is such that there are a number of machines and a number of jobs in the workshop. Each job includes a number of operations and each operation must be processed on a predetermined machine. However, the important point that distinguishes this from the rest of the problems is that the ordering of operations of a job is not predetermined. Provided that the order of operations had been important, we had a job shop problem (JSP) [13]. Therefore, any arbitrary order can be considered for operations of a job. For this reason, the formulation of the OSSP is more complicated than the job shop scheduling problem (JSSP) and the flexible job shop scheduling problem (FJSSP) problems. In the OSP, each machine can process at most one job at a time, and each job can also be processed at most on a single machine at a time [14–16].

In recent years, the application of mathematical models to finding the optimal solution (OS) of the scheduling problems has attracted the attention of many researchers. In this regard, many studies have been done on modeling the workshop jobs and workshop flows. In addition, formulation of the OSSP is less attractive to researchers. Since an increase in the number of machines ($m \geqslant 3$) turns the problem into an NP-hard one [7, 17, 18], heuristic and meta-heuristic algorithms have been used in most studies on the OSSP. Tautenhahn and Woeginger [19] studied the OSSP with regard to unit-time operations so that the availability of resources varies over time in their problem. They developed polynomial algorithms for multiple target functions such that the number of machines, the number of resources and the demand for each resource were limited. They were able to define a boundary between easy and hard versions of the problem by proving that they are NP-hard problems. The most important studies were carried out on heuristic algorithms by Brasel et al. [20] who presented a comparative study of heuristic algorithms for OSP aimed at minimizing the average flow time. They made comparisons for problems with up to 50 jobs and 50 machines. They showed that if the number of jobs is higher than machines, the algorithm solving time is far less. Among the meta-heuristic algorithms, Alcaide et al. [21] developed a tabu search (TS) algorithm for solving the OSP with the goal of minimizing the makespan. They tested their proposed algorithm on random data of varying dimensions. Liaw [22] proposed a hybrid genetic algorithm (HGA) to solve the OSSP aimed at minimizing the makespan. His proposed algorithm has been developed by adding a local optimization function to genetic algorithm (GA). The optimization function is based on TS algorithm. He used his proposed algorithm to solve random problems and the benchmark problems present in the literature up to 20 jobs and 20 machines, and in some cases, he was able to deliver results close to optimal values. Blum [23] presented an ant colony algorithm (ACO) using the beam search process and was able to deliver outstanding results for the problem. Huang and Lin [24] proposed a new honeybee optimization algorithm for the OSP considering the idle time. This method reduced time and cost based on the fact that the less the idle time, the less the makespan. Hosseinabadi and Ahmadizar [25] proposed a HGA for the OSSP with the goal of minimizing the maximum completion time of the jobs. In this algorithm, a special intersection operator is used to maintain the order of the machines and also a strategy is used to avoid the search for unnecessary responses in the jump operator. Chen et al. [26] presented the parallel open workshop scheduling by providing an approximate algorithm. Each job consists of two independent operations each of which is processed by one of the m pairs of parallel machines. The goal was to minimize the makespan of the last job. Naderi and Zandieh [27] studied the problem of open workshop planning without intermediate buffering. The paper presents an easy-to-use procedure of encryption and decryption frameworks for nonstop open shop problems (NS-OSP). The proposed meta-heuristic operators are designed to create a problem encryption framework. Bai et al. [28] explored the static and dynamic versions of the flexible open shop problem (FOSP) to minimize the makespan of the last job. They used the general dense scheduling algorithm (GDSA) for solving problems in a larger scale aimed at accelerating convergence. Tanimizu et al. [29] presented a scheduling method for solving the open workshop problems including disassembly and post-processing operations. A co-evolutionary algorithm was designed to improve the sequence of operations and product loading. Evolutionary algorithms have been widely used not only in the OSP, but also in all issues related to scheduling. For example, Gao et al. [30] proposed a new method based on the bee colony algorithm (BCA) that solves the FJSSP. In this method, the uncertainty of the processing time is modeled with a parameter called fuzzy processing time. Defined goals include the minimization of two metrics, fuzzy completion time and fuzzy machine workload. The proposed algorithm, IABC, can be a part of a decision maker's expert system in management and scheduling. In another study, Zhao et al. [31] proposed a new method based on the water wave optimization algorithm (WWOA) to solve the no-wait flow shop scheduling problem (NW-FSSP). The researchers devised a discrete version of the WWOA that has the ability to solve the flow shop scheduling problem (FSSP). In this method, a crossover operator is also defined besides the main operators of the WWOA. The crossover operator is used to prevent getting caught in local optima. Marichelvam et al. [32] solved the multistage hybrid FSSP using the bat algorithm (BA). Hybrid flow shop (HFS) is a generalization of the product deployment problem including several machines. Oulamara et al. [33] solved the two-machine OSSP. They considered the processing time of an operation as a combination of two times, preparation time and runtime. The preparation time period comes before the runtime period on the machine and requires renewable resources. The authors were able to prove that various versions fall into the category of NP-hard problems. Zaher et al. [34] introduced a meta-heuristic method based on BA in order to optimize the workshop deployment problem. This algorithm works by reducing the start delay instead of reducing the makespan. In [26], parallel OSSP with the aim of reducing makespan is addressed. Due to the NP-hard nature of the problem, using approximate algorithms is suggested to solve the problem. Goldansaz et al. [35] addressed the multi-processor OSSP with some constraints such as independent setup time, process time and sequence-dependent removal time. They used a combination of imperialist competitive algorithm (ICA) and GA to solve the problem. It has been proven in [36] that, if a constraint is considered for machines' load, three-machine proportionate OSP can be solved in $O(n \log n)$ time. If no constraint is taken into account, approximate methods can be used to solve the problem. Low and Yeh [37] defined the OSSP as an integer 0-1 programming model and then proposed a HGA with the aim of reducing the total delay of jobs and considered some constraints such as independent setup and dependent removal time. In [38], Lagrange expansion has been used to reduce the total quadratic completion time of small OSSP. In [21], an effective HGA named hybrid NSGA-II is proposed to solve the OSSP. In the proposed method, local search algorithm (LSA) is used to generate initial population. This method is used to solve medium and large problems. Results of the proposed method are compared with SPEA-II in terms of quality and diversity. Results indicated the superiority of the proposed method. An HGA with special operators is presented in [25] to solve the OSSP. Proposed crossover operator is able to eliminate the order in which jobs are performed on machines, and mutation operator avoids searching for additional solutions. The proposed method is a combination of iterative randomized active scheduling, dispatching index and lower bound.

There are many different versions of this problem which have their adjusted solution. One of the issues which can be added to this problem is priority. Some jobs have higher level of priority in comparison with others. This version of the problem has been addressed by an approximation algorithm. This method handles the process through a directed acyclic priority graph [39, 40]. Open shop scheduling problem or OSSP can be solved by other constraints. For example, predefined travel time between machines and setup times without any specific order. In order to tackle this problem, an innovative permutation model has been proposed to be used in variable neighborhood search. Evaluation of the algorithm on both random problems and standard benchmarks suggests that the innovative permutation method has the most positive impact on produced outputs [41, 42].

Some papers have focused on a solution for OSSP which can be run on a multiprocessor. Parallel executing of an algorithm could increase its productivity significantly. For example, in a paper a new method has been proposed. In this method, an innovative SS/PR meta-heuristic has been designed for the multiprocessor OSSP. Moreover, a distance function has been developed to show the difference between different solutions. Based on a standard benchmark, evaluating the algorithm shows gaining optimal solution in 7 cases and new upper bound in 18 other special cases [43, 44].

In this paper, we will solve the OSP for the first time using the bat meta-heuristic algorithm. Previous methods have some shortcomings in exploring problem space. Hence, we need a better algorithm in terms of exploration in order to tackle this problem. Since BA is a strong algorithm in both aspects of exploration and exploitation, it has been used in this article. The structure of the paper is organized as follows: in Sect. 2, the problem statement is described. Section 3 describes the bat algorithm. Section 4 presents the proposed algorithm. Simulation results and conclusions are presented in Sects. 5 and 6, respectively.

2 Problem statement

The OSSP is a matter of general task scheduling. The open shop is an NP-hard problem and solving it in polynomial time is not possible. In this case, there are n jobs and m machines. Each job contains m operations each of which must be run by a machine. The machine that must execute an operation is already specified. The order of the operations of a job is not important, but more than a single operation cannot be executed at a moment. The problem solution is generated as a matrix. Each row of this matrix shows the order of the jobs a machine should run. The following parameters are used to define the OSP:

n – number of jobs in the OSP,
m – number of machines in the OSP,
i – machine index in relations,
j – job index in relations,
$op_{j,i}$ – Job_j operation that must be run on $Machine_i$,
$d_{j,i}$ – $op_{j,i}$ runtime; this parameter is an input for the scheduling problem,
$t_{j,i}$ – start time of running $op_{j,i}$ on $Machine_i$,
f_i – $Machine_i$ idle time; the default value of this parameter is 0 when starting the scheduling for all machines,
$assigned_{j,i}$ – equal 1 if $op_{j,i}$ is selected for execution and equals 0 if $op_{j,i}$ is not selected for execution. The default value of this parameter is 0 when starting the scheduling for all $op_{j,i}$.

By reference to the above parameters, the runtime period will be equal to $[t_{j,i}, t_{j,i} + d_{j,i}]$. To calculate this time period, we must first calculate $t_{j,i}$ as follows:

$$t_{j,i} = \max\{f_i, (t_{j,k} + d_{j,k})\} \tag{1}$$

for all k where $assigned_{j,k} = 0$ and $k \neq i$. In Eq. 1, variable i represents the machine chosen for calculation the scheduling. Machines are selected in a special order for scheduling. The choice of $Machine_i$ for calculating in the current stage is done using Eq. 2. After selecting $Machine_i$ and assigning $op_{j,i}$ to it, $assigned_{j,i} = 1$ and $f_i = t_{j,i} + d_{j,i}$.

$$i = k \quad \text{where} \quad f_k = \min_{l=1}^{m}(f_l) \tag{2}$$

If Eq. 3 is applied, the open shop scheduling (OSS) is a valid and acceptable scheduling. Equation 3 shows that more than one operation will not be executed at a single moment for any job.

$$i = k \quad \text{where} \quad f_k = \min_{l=1}^{m}(f_l) \tag{3}$$

The main purpose of solving the OSP is to reduce the amount of makespan. After calculating $t_{j,i}$, Eq. 4 is used to obtain the makespan. In solving the OSP, we try to reduce the value of Eq. 4 function. In other words, this function is used as the target function in the problem.

$$\mathrm{I}_{i=1}^{m}[t_{j,i}, d_{j,i}] = \phi(\text{for} \quad j = 1 \quad to \quad n) \tag{4}$$

3 The bat algorithm

Meta-heuristics usually inspired by nature and physical processes are now used as one of the most powerful methods to solve many complex optimization problems. BA is one of the nature-inspired meta-heuristic algorithms introduced by Yang [45]. This algorithm is based on the principles of bat life. Bats are the only mammals with wings that use echolocation for hunting prey. So far, it has been used to solve binary problems [46] and multi-objective optimization problems [47]. However, there are many discrete optimization problems that can be solved with existing meta-heuristics. So the goal of this paper is to provide a suitable form of bat algorithm for this type of problems. The BA is inspired by the traceability of small bats searching for hunt. So that small bats can hunt their prey and receive it in absolute darkness by emitting sound [48]. Three rules are used to develop this algorithm:

- all bats use echolocation to detect distances and know the difference between food and progressive barriers,
- bats fly randomly at velocity v_i, at position x_i, with fixed frequency of frq_i and different wavelengths λ and loudness of A_0 for hunting prey. Also, they can automatically set emitted waves and sent pulse rates ($r \in [0, 1]$) according to proximity to their hunts,
- given that loudness may vary in many different ways, we assume that loudness varies from R_0 (maximum value) to R_{min} (minimum value).

According to the rules, the position x_i^t with velocity v_i^t for each virtual bat i in iteration t and frequency frq_i is calculated as follows:

$$frq_i = frq_{min} + (frq_{max} - frq_{min}) \cdot \beta \tag{5}$$

$$v_i^t = v_i^{t-1} + (x_i^{t-1} - X^*) \cdot frq_i \tag{6}$$

$$x_i^t = x_i^{t-1} + v_i^t \tag{7}$$

where $\beta \in [0, 1]$ is a random vector with uniform distribution, X^* is the best current position that is selected in each iteration and after comparison with the position of the virtual bats. Usually, frequency frq_i is selected between $frq_{\min} = 0$ and $frq_{\max} = 100$. In each iteration of local search, one solution is selected as the Best Solution (BS), and the new position of each bat is updated with a random step as follows:

$$x_{\text{new}} = x_{\text{old}} + \epsilon \cdot A^t \tag{8}$$

where $\epsilon \in [-1, 1]$ is a random number and $A^t = <A_i^t>$ is the average loudness of bats in iteration t. loudness A_i and pulse rate r are updated as follows:

$$A_i^{t+1} = \alpha \cdot A_i^t \quad r_i^{t+1} = r_i^0 \cdot [1 - exp(-\gamma \cdot t)] \tag{9}$$

where α and γ are constants and for each $0 < \alpha < 1$ and $r > 0$ when $t \to \infty$, we have

$$A_i^t \to 0 \quad r_i^t \to r_i^0 \quad t \to \infty \tag{10}$$

According to the discussion above, BA is summarized in Algorithm 1.

Algorithm 1: Steps of bat algorithm

```
Initialization. Set the generation counter t = 1; Initialize randomly the
  population P consisted of NP bats and each bat corresponding to a potential
  solution to the given problem; Define loudness A_i, pulse frequency frq_i and the
  initial velocities v_i (i = 1, 2, ..., NP); Set pulse rate r_i.
while the termination criterion is not satisfied or t < MaxGeneration do
  | Generate new solutions by adjusting frequency, and updating velocities and
  |  location solutions (Equations 5 and 6) if rand > r_i then
  |   | Select a solution among the best solutions
  |   | Generate a location solution around the selected best solution (Equation 7)
  | end
  | Generate a new solution by flying randomly (Equation 8)
  | if rand < A_i and f(x_i) < f(x_*) then
  |   | Accept the new solution
  |   | Increase r_i and reduce A_i (Equation 9)
  | end
  | Rank the bats and find the current best X*
  | t = t + 1
end
Return X*
```

The BA has the ability to achieve OS. The convergence analysis of the BA with the aid of the Markovian framework and dynamical system theory shows that this method will approach to the OS. The convergence of the BA is proved by both the Markovian framework and dynamical system theory. The empirical results of implementing this algorithm on well-known problem confirm its theoretical analysis [49].

4 The proposed algorithm

There are many applied solution methods developed for the optimizations problem in the literature such as exact methods, heuristics, meta-heuristics and so forth. Evolutionary algorithms are widely used to solve task scheduling problems. Given that the task scheduling problems are of an NP-complete type, the use of evolutionary methods can lead to a reasonable solution to this problem in a logical time. In this paper, a particle-based algorithm called bat is used to solve the problem. The BA is inspired by bats group movement to find food. In the following, how to formulate the task scheduling problem for the BA and the steps of the proposed algorithm are described.

4.1 Steps of the proposed algorithm

The proposed algorithm begins by creating some bats. Each bat has a random solution to the task scheduling problem. The best bat will be the base of the bats' movement in the next round. The fitness of a bat shows its excellence. The bat fitness function represents the delivery time of the entire jobs in the scheduling problem. The discussion on how to calculate fitness is given in the following sections. After creating the bat and initializing the parameters of the algorithm, the main steps of the algorithm which simulates the movement of the bats begin. Algorithm 2 shows the general steps of the proposed BA for the task scheduling problem. In Algorithm 2, the fitness function is used to calculate the fitness of a bat which works on the basis of delivery time or makespan. Another function called *ColReuse* was used to calculate velocity of the bats. The concept of "velocity" has been redefined for this problem. The new definition is based on a heuristic similarity between bats. Also, the bats movement was done using the *Substitution*, *Fold*, *FullReverse*, *Join*, *ShiftUp*, and *ShiftDown* functions. These functions randomly change part of a solution. The *InactionDel* and *SmallWalk* functions are used to generate optimal solutions around the BS. The *InactionDel* function detects and eliminates the largest idle time between processors. The *SmallWalk* function creates a small movement in the bat by shifting two random jobs. The operation and pseudocode of these functions are described in the following sections.

Algorithm 2: OSSP based on BA

```
Initialize loudness (A), pulse rate (r) and create NP bats
Select the best bat (X*) based on fitness function
for t=1 to number of walks (MaxGeneration) do
  | for each bat (x_i) do
  |   | Set frequency (frq_i) of bat randomly
  |   | Set velocity based on Colreuse function and frequency
  |   | Temporarily move bat based on velocity and one of the following functions:
  |   |   Fold, Reverse, FullReverse, Join, ShiftUp, ShiftDown
  |   | By probability of 1 − r temporarily move bats around the best based on
  |   |   SmallWalk and InactionDel functions, and ignore previous position
  |   | By probability of A if new position of bat is better than previous, accept
  |   |   new position
  |   | if new position is better than best bat (X*) then
  |   |   | Set new position as the best bat (X*)
  |   | end
  | end
  | Update pulse rate (r)
end
Return best bat (X*) as solution
```

4.1.1 Creating a bat

Given that the number of machines and jobs is equal in one of the benchmarks of this problem, all pseudocodes and examples are written for an equal number of machines and jobs without losing the whole problem. This is done only to simplify the description of the functions. Therefore, a bat is a random solution to a problem of a $k \times k$ size. To generate random solutions, we first convert the solution space to a permutation of integers from one to $k \times k$. Table 1 shows how this conversion works for $k = 3$. If we list all possible combinations of machines and jobs, we can create a sequence of numbers from one to $k \times k$ by taking a

Table 1 Numbering the job and machine states

Sequence number creation									
Machine	1	1	1	2	2	2	3	3	3
Job	1	2	3	1	2	3	1	2	3
Numbers	1	2	3	4	5	6	7	8	9

sequence number. Any permutation of these numbers is a solution to the problem.

Equations 11 and 12 are used to calculate machine number and job number.

$$Machine = \lfloor \frac{\text{number}}{k} \rfloor + 1 \tag{11}$$

$$Job = (\text{number} \mod k) + 1 \tag{12}$$

In these equations, k is the dimension of the problem (the number of machines and jobs) and “number” is a number in $[1, k \times k]$. Equations 11 and 12 are calculated for all the numbers in the initial solution, and finally, a $k \times k$ matrix is returned as the final solution. In Table 2, there is an example of a randomized solution to the problem for $k = 4$.

4.1.2 Fitness function

The fitness value of a bat is equal to the amount of the makespan or the time it takes to complete all the jobs. To calculate this value, you must first schedule all the tasks. At the end of the scheduling, the exact time of the start and end of each task is determined. Therefore, the makespan value is simply quantifiable. The method of computing makespan is described in Eq. 4. In OSSP, each task is a set of subtasks that should not be run simultaneously. Therefore, the scheduling function postpones the start time of each subtask until the machine is empty and the associated subtasks are completed. Algorithm 3 shows the pseudocode of the scheduling function.

Algorithm 3: Scheduling OSSP based on solution S

```
for j=1 to k do
    for each machine do
        job = s(machine, j)
        duration = time of this job's subjob that assigned to machine
        start = max(MET(machine), PET(job))
        finish = start + duration
        begin(machine, j) = start
        end(machine, j) = finish
        MET(machine) = PET(job) = finish
    end
end
Return begin and finish
```

In Algorithm 3, $MET_{1\times k}$ is the time each machine takes to be empty with an initial value of zero, $PET_{1\times k}$ is the completion time of processing a job with an initial value of zero, *duration* is the time length of performing a subtask, *start* is the start time of a subtask, *finish* is the end time of a subtask, $begin_{k\times k}$ is start time of subtasks of each machine, and $end_{k\times k}$ is the end time of subtasks of each machine.

4.1.3 ColReuse function

To calculate the difference between a bat (x_i) and the best bat (X^*), a heuristic function called *ColReuse* was used. This function shows the maximum repetition of a number in a scheduling column. For a better understanding of the role of *ColReuse* in scheduling quality, look at the standard 5×5 example in Table 3. In this example, there are three different schedules, *ColReuse* 5, 4 and 3, respectively. By paying attention to the fitness of these schedules, we find that the higher the number of duplicate numbers in a column, the lower the scheduling quality. Using this function, we take the distance between the two bats as the *ColReuse* difference between them. The velocity of a bat in the proposed algorithm is equal to the difference between the bat (x_i) and the best bat (X^*). Therefore, the more the distance between the bat and the BS, the faster it moves to reach it.

4.1.4 Substitution function

Substitution function creates a new permutation in a number of matrix rows. Rows are selected if one of their elements plays a role in increasing the *ColReuse* value. By creating a new permutation, *ColReuse* will drop and *Fitness* value will probably decrease. Table 4 shows a solution for a problem with 5 jobs and 5 machines, before and after substitution. The modified rows is marked in bold.

4.1.5 Fold, FullReverse and join functions

In this section, three new functions used to move a bat are examined. Each of these functions in a particular way converts a solution to a new one. Therefore, the search in the solution space becomes more complete and the likelihood of finding a BS becomes greater. The *Fold* function

Table 2 A sample solution for $k = 4$

	Job number			
Machine 1	1	3	4	2
Machine 2	3	2	4	1
Machine 3	1	2	3	4
	4	1	2	3

Table 3 The effect of the *ColReuse* function

Solution	1	2	3	4	5	2	3	4	5	1	2	3	4	5	1
	1	2	3	4	5	1	2	3	4	5	5	1	2	3	4
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
ColReuse	5					4					3				
Fitness	1220					948					713				

Table 4 Substitution performance

	Before substitution					After substitution				
Solution	3	2	1	5	4	3	2	1	5	4
	1	3	4	5	2	**4**	**1**	**5**	**2**	**3**
	1	2	5	4	3	**2**	**4**	**3**	**1**	**5**
	1	3	5	4	2	**3**	**5**	**1**	**2**	**4**
	5	4	2	3	1	5	4	2	3	1
ColReuse	3					2				

Table 5 Performances of *Fold*, *FullReverse* and *Join* functions

Solution				After Fold				After FullReverse				After Join			
4	2	1	3	4	2	1	3	**3**	**1**	**2**	**4**	4	2	1	3
3	2	1	4	3	2	1	4	**4**	**1**	**2**	**3**	3	2	1	4
3	4	2	1	**1**	**2**	**4**	**3**	**1**	**2**	**4**	**3**	**2**	**1**	**4**	**3**
1	3	2	4	**4**	**2**	**3**	**1**	**4**	**2**	**3**	**1**	**3**	**4**	**1**	**2**

chooses one of the rows randomly and reverses the top or bottom of that row completely. The *FullReverse* function reverses all the solution rows. The *Join* function selects a few rows in a solution and replaces them with the rows of a random bat. Table 5 shows an example of the performance of these functions. Modified parts are shown in bold.

4.1.6 ShiftUp and ShiftDown functions

The *ShiftUp* and *ShiftDown* functions choose one of the response columns randomly and then shift it up or down (depending on the function). After the shift operation, a duplicate number is created in some rows. To correct this state, the deleted number is replaced the number that has just entered. Therefore, a new and correct solution is obtained. Algorithm 4 shows the pseudocode of these functions.

Table 6 *ShiftUp/Down* performance

Solution					ShiftUp					ShiftDown			
2	1	3	5	4	**4**	1	3	5	**2**	2	**3**	**1**	5
4	3	1	5	2	**5**	3	1	**4**	2	4	**1**	**3**	5
5	4	1	3	2	**3**	4	1	**5**	2	5	4	**1**	3
3	5	4	2	1	**2**	5	4	**3**	1	3	5	**1**	2
2	4	1	3	5	**2**	4	1	3	5	2	**1**	**4**	3

Algorithm 4: ShiftUp/Down function

```
Randomly select column j
Rotate Up/Down column j
for each row in S do
    if x loaded on y in this row then
        Replace another x in this row by y
    end
end
Return S
```

The performances of *ShiftUp* and *ShiftDown* functions are illustrated in Table 6 (the columns are shifted and the modified elements are shown in bold).

4.1.7 SmallWalk and InactionDel functions

All the functions described in the previous sections are meant for moving ordinary bats. However, the two functions, namely *SmallWalk* and *InactiveDel*, were intended for creating a place next to the best bat (X^*). Based on a $1 - r$ probability, a bat (x_i) is moved to that place. Pulse rate (r in Eqs. 9 and 10) has an initial value of 1 and decreases according to Eq. 13.

$$r = 1 - \left(\frac{1}{MaxGeneration + 1 - t}\right) \tag{13}$$

In Eq. 13, *MaxGeneration* is the maximum number of bats movements and t is the number of moves up to this moment. The diagram of this function, as depicted in Fig. 1, shows that at the beginning of the algorithm, the bats move to the area around the best bat with a little probability and they try to find the solution themselves. However, at the end of running the algorithm, most bats move to the area around the best bat and try to improve the BS.

The *InactionDel* function was created with the aim of eliminating the greatest idle time between machines. This

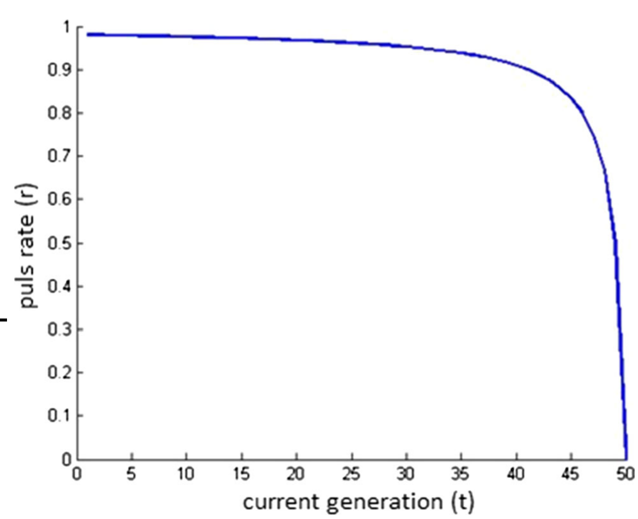

Fig. 1 Pulse rate (r) modification for 50 generations

function finds the biggest gap between machines and shifts the machine tasks to the right. Therefore, the task that was delayed and created a gap moves from its place and another task takes this place and probably uses this idle time. The pseudocode of the *InactionDel* function is written in Algorithm 5.

Algorithm 5: InactionDel function

```
Scheduling S
for each machine i do
    Find max inaction time for machine
end
row = i (where max inaction machine_i is greatest inaction time)
Rotate row to right
Return S
```

The last function examined is the *SmallWalk* function. This function randomly selects two points of the matrix and moves its values. After moving, duplicate values may be created in two rows of the matrix. Duplicate values are deleted in the same way as performed for the *Shift* function. In the next section, the proposed method is implemented and the results are compared with those of the previous methods. In this section, functions designed to solve the OSP using the BA are described. After full description of these functions, it is necessary to determine how they will be used during the execution of the BA. Figure 2 shows the flowchart of the proposed algorithm.

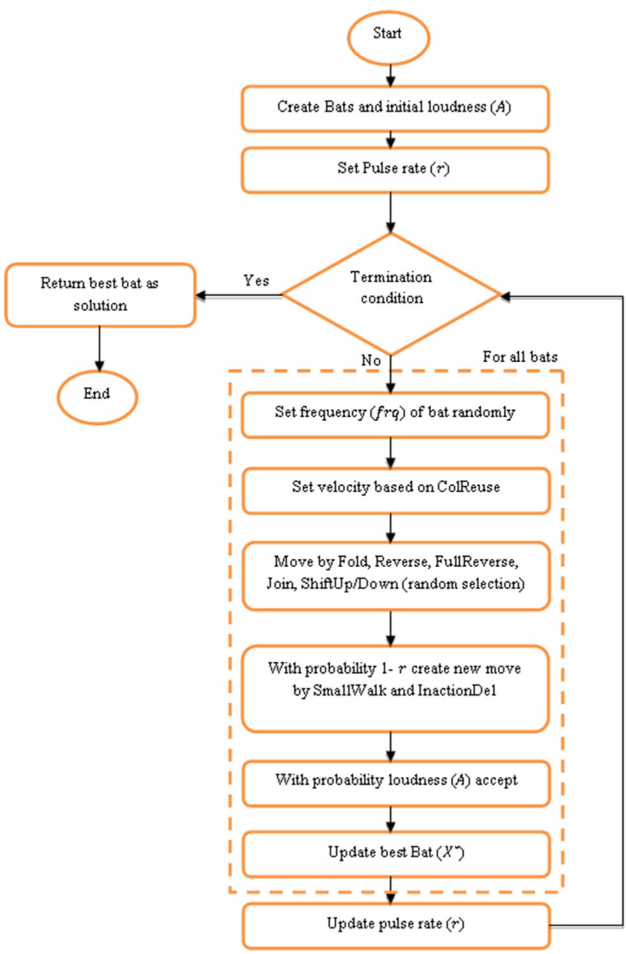

Fig. 2 Flowchart of proposed algorithm

5 Simulation results

In this section, simulation results of the proposed algorithm (BA_OS) are presented. MATLAB programming language on a Laptop of Intel Core i7-4720HQ CPU 2.60GHz and 12.0 GB RAM is used to implement the proposed algorithm. In order to investigate the efficiency of the proposed algorithm for solving OSSP, this section presents scenarios including various jobs and machines for solving the problem using BA_OS algorithm; then, results are compared. Generated scenarios are divided into three groups according to the number of jobs and the complexity of the scenarios. Table 7 shows these scenarios with the number of jobs and machines. The parameters of the proposed algorithm are defined as follows: the number of bats that is set by the user for each problem. The number of generations or a number of bat movements are set by the user for each problem. Loudness (A) is set to 0.95. Pulse rate (r) emission starts from 1 and gradually (based on Eq. 13) reaches to 0 (Table 8).

The BA_OS has been tested on Taillard criteria [50]. Tables 9 and 10 show the results obtained after applying any Taillard criteria [50] using the determined parameters. As seen, BA_OS achieves optimal solutions in most samples and achieved a relative advantage over compared algorithms.

Comparison of the proposed algorithm (BA_OS) with other algorithms is presented in Tables 9 and 10. The proposed algorithm is compared with simulated algorithm (SA) [51], GA [52, 53] HGA [52, 54], neural networks [55], ant colony system (ACS) [56], cuckoo search (CS) [56] and cat swarm optimization algorithm (CSO) [57]. As seen, proposed algorithm could solve the problem well and compete with compared algorithms.

Table 7 Generated scenarios

Data	Number of jobs	Number of machines	Display
Small-scale	4	4	4×4
data	5	5	5×5
Medium-scale	7	7	7×7
data	10	10	10×10
Large-scale	15	15	15×15
data	20	20	20×20

Table 8 Parameter setting

Pulse rate (r)	Loudness (A)	Number of generations	Number of bats
Start from 1 and gradually reaches to 0	0.95	2000–3000	40–200

Table 9 Obtained results for benchmarks of size 4×4, 5×5, 7×7 and 10×10

Benchmarks	Optimal solution	SA [51]	GA [52]	HGA [52]	HGA [54]	GA [53]	EGA_OS [58]	ACS [56]	CS [56]	CSO [57]	BA_OS
$4 \times 4 - 1$	193	193	193	213	193	193	193	193	193	193	193
$4 \times 4 - 2$	236	236	236	240	236	239	239	236	236	236	236
$4 \times 4 - 3$	271	271	271	293	271	271	271	271	271	271	271
$4 \times 4 - 4$	250	250	250	253	250	250	250	250	252	250	250
$4 \times 4 - 5$	295	295	295	303	295	295	295	295	295	295	295
$4 \times 4 - 6$	189	189	189	209	189	189	189	189	189	189	189
$4 \times 4 - 7$	201	201	201	203	201	201	201	201	201	201	201
$4 \times 4 - 8$	217	217	217	224	217	217	217	217	217	217	217
$4 \times 4 - 9$	261	261	261	281	261	261	261	261	261	261	261
$4 \times 4 - 10$	217	217	217	230	217	217	217	217	217	217	217
$5 \times 5 - 1$	300	300	301	323	300	301	300	301	301	300	300
$5 \times 5 - 2$	262	262	262	269	262	263	262	262	262	262	262
$5 \times 5 - 3$	323	323	331	353	323	335	323	331	335	323	323
$5 \times 5 - 4$	310	310	N/A	N/A	310	316	310	315	314	310	310
$5 \times 5 - 5$	326	326	N/A	N/A	326	330	326	331	329	326	326
$5 \times 5 - 6$	312	312	312	327	312	312	312	317	318	312	312
$5 \times 5 - 7$	303	303	N/A	N/A	303	308	303	308	305	303	303
$5 \times 5 - 8$	300	300	N/A	N/A	300	304	300	304	303	300	300
$5 \times 5 - 9$	353	353	353	373	353	358	353	358	358	353	353
$5 \times 5 - 10$	326	326	326	341	326	328	326	329	329	326	326
$7 \times 7 - 1$	435	435	438	447	435	436	435	435	436	435	435
$7 \times 7 - 2$	443	443	455	454	443	447	443	445	447	443	443
$7 \times 7 - 3$	468	468	N/A	N/A	468	472	468	479	472	468	468
$7 \times 7 - 4$	463	463	N/A	N/A	463	463	463	467	466	463	463
$7 \times 7 - 5$	416	416	N/A	N/A	416	417	416	419	416	416	416
$7 \times 7 - 6$	451	451	N/A	N/A	451	455	451	460	454	452	451
$7 \times 7 - 7$	422	422	443	450	422	426	422	435	425	422	422
$7 \times 7 - 8$	424	424	N/A	N/A	424	424	424	424	424	426	424
$7 \times 7 - 9$	458	458	465	467	458	458	458	458	458	458	458
$7 \times 7 - 10$	398	398	405	406	398	398	398	398	399	398	398
$10 \times 10 - 1$	637	637	667	655	637	637	637	638	639	645	637
$10 \times 10 - 2$	588	588	N/A	N/A	588	588	588	588	688	588	588
$10 \times 10 - 3$	598	598	N/A	N/A	598	598	598	599	600	599	598
$10 \times 10 - 4$	577	577	586	581	577	577	577	577	577	577	577
$10 \times 10 - 5$	640	640	N/A	N/A	640	640	640	640	640	640	640
$10 \times 10 - 6$	538	538	555	541	538	538	538	538	538	538	538
$10 \times 10 - 7$	616	616	N/A	N/A	616	616	616	616	616	616	616
$10 \times 10 - 8$	595	595	N/A	N/A	595	595	595	595	595	595	595
$10 \times 10 - 9$	595	595	627	598	595	595	595	595	595	595	595
$10 \times 10 - 10$	596	596	623	605	596	596	596	596	596	596	596

"N/A" is common in the table for the phrase not applicable

Table 10 Obtained results for benchmarks of size 15×15, and 20×20

Benchmarks	Optimal solution	SA [51]	GA [52]	HGA [52]	HGA [54]	GA [53]	Neural [55]	EGA_OS [58]	ACS [56]	CS [56]	CSO [57]	BA_OS
$15 \times 15 - 1$	937	937	967	937	937	937	937	937	937	937	937	937
$15 \times 15 - 2$	918	918	N/A	N/A	918	918	918	918	918	918	920	918
$15 \times 15 - 3$	871	871	904	871	871	871	871	871	871	871	871	871
$15 \times 15 - 4$	934	934	969	934	934	934	934	934	934	934	934	934
$15 \times 15 - 5$	946	946	N/A	N/A	946	946	946	946	946	946	952	946
$15 \times 15 - 6$	933	933	N/A	N/A	933	933	933	933	933	933	933	933
$15 \times 15 - 7$	891	891	N/A	N/A	891	891	891	891	891	891	891	891
$15 \times 15 - 8$	893	893	928	893	893	893	893	893	893	893	893	893
$15 \times 15 - 9$	899	899	N/A	N/A	899	899	899	899	902	902	913	899
$15 \times 15 - 10$	902	902	N/A	N/A	902	902	902	902	902	902	902	902
$20 \times 20 - 1$	1155	1155	1230	1165	1155	1155	1155	1155	1155	1155	1166	1155
$20 \times 20 - 2$	1241	1241	N/A	N/A	1241	1241	1242	1241	1242	1243	1260	1241
$20 \times 20 - 3$	1257	1282	1292	1257	1257	1257	1257	1257	1257	1257	1257	1257
$20 \times 20 - 4$	1248	1274	N/A	N/A	1248	1248	1248	1248	1248	1248	1253	1248
$20 \times 20 - 5$	1256	1289	1315	1256	1256	1256	1256	1256	1256	1256	1256	1256
$20 \times 20 - 6$	1204	1204	1266	1207	1204	1204	1204	1204	1204	1204	1204	1204
$20 \times 20 - 7$	1294	1294	N/A	N/A	1294	1294	1294	1294	1295	1294	1310	1294
$20 \times 20 - 8$	1169	1169	N/A	N/A	1173	1171	1173	1170	1176	1175	1210	1170
$20 \times 20 - 9$	1289	1307	1339	1289	1289	1289	1289	1289	1289	1289	1289	1289
$20 \times 20 - 10$	1241	N/A	1307	1241	1241	1241	1241	N/A	1241	1241	1241	1241

"N/A" is common in the table for the phrase not applicable

Table 9 shows the results for small size problems. In these problem sets, the optimal solutions are clear. Results of the proposed method show that all of the optimal solutions have been obtained by BA_OS. Table 10 lists the results for medium size problems. Solutions of proposed method indicate that in all problems (except $20 \times 20 - 8$) the outputs are exactly equal to optimal solution. In comparison with other mentioned algorithms, this shows a reasonable enhancement. Large size problems have been tested in further tables.

The Gantt chart of a sample schedule for benchmark $10 \times 10 - 1$ is depicted in Fig. 3. The white pars of the chart show the gap times in each machine. Each color shows the operations of a particular job.

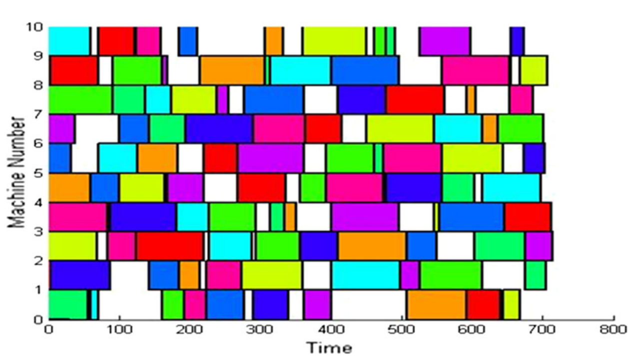

Fig. 3 Gantt chart of a sample schedule for benchmark $10 \times 10 - 1$

5.1 Test problems

In this section, for further verification, the performance of the proposed BA_OS is evaluated using the benchmarks presented recently by Shamshirband et al. [6]. According to these benchmarks, some GA-based algorithms including DGA [37], SAGA [37], TSGA [37] and PGA [37] have been proposed to solve the problem which are compared to the proposed BA_OS. The main body of these methods is GA. The difference between these methods is in their local optimizations which are implemented by GA, SA (simulated annealing), and TS. The structure of TS and SA with a short memory is the same as classical forms (SA [59], TS [60]).The obtained results are presented in Tables 11 and 12 for small-sized (job size $n = 3$ and 4; machine size $m = 2$ and 3) and large-sized problems (job size $n = 10$, 30 and 50; machine size $m = 5$, 10 and 15), respectively. Since the results in Tables 11 and 12 include the BS and mean ones, the stability of the proposed method should be analyzed before investigating these tables. A multiple run of a plot and drawing of the plot box from the values obtained shows how far the outputs of that method are

Table 11 Performance comparisons of the proposed algorithm and four GA-based heuristics (for small-size problems)

Problem	Optimum solution	DGA [37]		SAGA [37]		TSGA [37]		PGA [6]		BA_OS	
$(n \times m)$		Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
$3 \times 2 - 1$	177	177	177	177	177	177	177	177	177	177	177
$3 \times 2 - 2$	109	109	109	109	109	109	109	109	109	109	109
$3 \times 2 - 3$	224	224	224	224	224	224	224	224	224	224	224
$3 \times 2 - 4$	241	241	241	241	241	241	241	241	241	241	241
$3 \times 3 - 1$	173	173	173	173	173	173	173	173	173	173	173
$3 \times 3 - 2$	193	193	193	193	193	193	193	193	193	193	193
$3 \times 3 - 3$	212	212	212	212	212	212	212	212	212	212	212
$3 \times 3 - 4$	255	255	255	255	255	255	255	255	255	255	255
$4 \times 2 - 1$	352	352	352	352	352	352	352	352	352	352	352
$4 \times 2 - 2$	393	393	393	393	393	393	393	393	393	393	393
$4 \times 2 - 3$	408	408	408	408	408	408	408	408	408	408	408
$4 \times 2 - 4$	556	556	556	556	556	556	556	556	556	556	556
$4 \times 3 - 1$	–	402	404.2	402	405.0	402	410.3	399	401	395	398
$4 \times 3 - 2$	–	487	487	489	492.2	487	493.1	483	484	479	481
$4 \times 3 - 3$	–	605	605.6	605	607.0	605	607.4	601	601	595	596
$4 \times 3 - 4$	–	388	388.6	388	389.2	388	388.8	382	382	375	375

"–" indicates that the optimum solutions cannot be obtained by the extended Lindo within 50 running hours

Table 12 Performance comparisons of the proposed algorithm and four GA-based heuristics (for large-size problems)

Problem	DGA [37]		SAGA [37]		TSGA [37]		PGA [6]		BA_OS	
$(n \times m)$	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
$10 \times 5 - 1$	3048	3058.4	3048	3098	3050	3138	3043	3045	3039	3042
$10 \times 5 - 2$	2926	2980.2	2932	3100.8	2932	3148.4	2911	2917	2902	2905
$10 \times 5 - 3$	3043	3061.7	3043	3084	3087	3114.8	3014	3019	3009	3011
$10 \times 5 - 4$	1965	1972.3	1968	1985.1	2000	2008.2	1922	1925	1910	1913
$10 \times 10 - 1$	3623	3650	3705	3760.8	3696	3812.6	3605	3612	3596	3600
$10 \times 10 - 2$	2457	2520.4	2516	2600.4	2532	2636.4	2419	2426	2409	2413
$10 \times 10 - 3$	1016	1056	1044	1092.7	1075	1121.5	1007	1014	1001	1007
$10 \times 10 - 4$	1455	1492.7	1455	1504.2	1457	1552	1418	1423	1410	1414
$30 \times 5 - 1$	4523	4537.4	4523	4602.7	4582	4652.8	4501	4509	4494	4500
$30 \times 5 - 2$	4587	4626.8	4590	5670.4	4601	4705	4532	4540	4527	4531
$30 \times 5 - 3$	3864	3880.5	3912	3958	3868	3984.3	3835	3842	3830	3836
$30 \times 5 - 4$	5137	5184.1	5137	5232.5	5193	5782.5	5117	5127	5108	5114
$30 \times 15 - 1$	6128	6188.8	6212	6280.6	6216	6312.8	6111	6134	6101	6116
$30 \times 15 - 2$	5042	5124	5120	5204.9	5120	5220.1	5013	5026	5006	5014
$30 \times 15 - 3$	4815	4846.2	4832	4920	4850	4950.8	4804	4832	4796	4802
$30 \times 15 - 4$	5284	5344.4	5291	5385	5291	5410.2	5233	5241	5222	5230
$50 \times 5 - 1$	6052	6140.8	6150	6318.2	6208	6338.4	6026	6076	6019	6038
$50 \times 5 - 2$	6615	6742.4	6692	6876.2	6680	6934.1	6603	6631	6546	6573
$50 \times 5 - 3$	5918	6035.1	6080	6290.2	6097	6298.2	5906	5945	5704	5741
$50 \times 5 - 4$	7422	7560.3	7510	7768.5	7590	7842	7409	7426	7217	7224

Table 12 (continued)

Problem	DGA [37]		SAGA [37]		TSGA [37]		PGA [6]		BA_OS	
($n \times m$)	Best	Average	Best	Average	Best	Average	Best	Average	Best	Average
$50 \times 15 - 1$	6485	6604.2	6540	6876.5	6605	6950.5	6419	6453	6402	6427
$50 \times 15 - 2$	8905	9015.7	8995	9250.3	9142	9390.8	8876	8895	8852	8821
$50 \times 15 - 3$	6624	6843.5	6708	6870	6708	6940.3	6604	6638	6546	6555
$50 \times 15 - 4$	7350	7413.6	7395	7522.6	7395	7560.7	7315	7352	7289	7299

close to the mean value. As much as the generated values are closer to the mean value, the stability of that method is higher. Box plot of several run of the proposed method on six sample problems of large-scale size is shown in Fig. 4.

Investigating the values of Fig. 4 shows that in many cases, the outputs of the proposed method are close to the mean value. Therefore, the proposed method produces balanced outputs. Among six examined test problems, in three problems $10 \times 10 - 1$, $30 \times 5 - 1$ and $30 \times 15 - 1$ only one solution was distant from the average value, and the rest ones were within an acceptable distance from the mean value. The minimum value of each box plot represents the best produced solution. The details of the results for this benchmark are shown in Tables 11 and 12. According to the tables, the proposed method produces a better solution in most of the problems.

For a better comparison between the proposed method and the four previous methods, the results of these algorithms are shown in Fig. 5 in the form of box plots. The values of box plots are obtained from test problem $50 \times 5 - 1$. Results show that the results of the proposed method have the most similarity to the mean value. The PGA [6] method, with BS close to the proposed method, is much weaker in terms of sustainability. Other methods are much weaker than the proposed method in terms of both the BS and sustainability.

6 Conclusion

The OSSP is a well-known scheduling problem with high application in industries. So, finding an optimal applicable scheduling in this environment would help in order to execute the best policy in industries. In this paper, a proposed bat algorithm is applied for solving OSSP. The aim of the proposed algorithm is to reduce production costs through minimizing makespan of such systems. Given that classical BA is presented for solving continuous problems and scheduling problem is a discrete problem, operations such as difference and bat's movement are defined so that proposed algorithm can operate in a discrete environment. Optimization of random bats is done by two heuristic functions named: *ColReuse* and *InactionDel*. To evaluate the performance of the proposed algorithm, we tested it on standard benchmark and compared it algorithm with other algorithms. This benchmark includes small-, medium- and large-scale problems. Experimental results show that proposed algorithm obtained better results in all three categories. This improvement can be more obvious in larger production systems. For future studies, it is suggested to consider more real assumptions in the problem such as setup times and maintenance of the machines. Also, studying the uncertain nature of the parameters would make the mode more real.

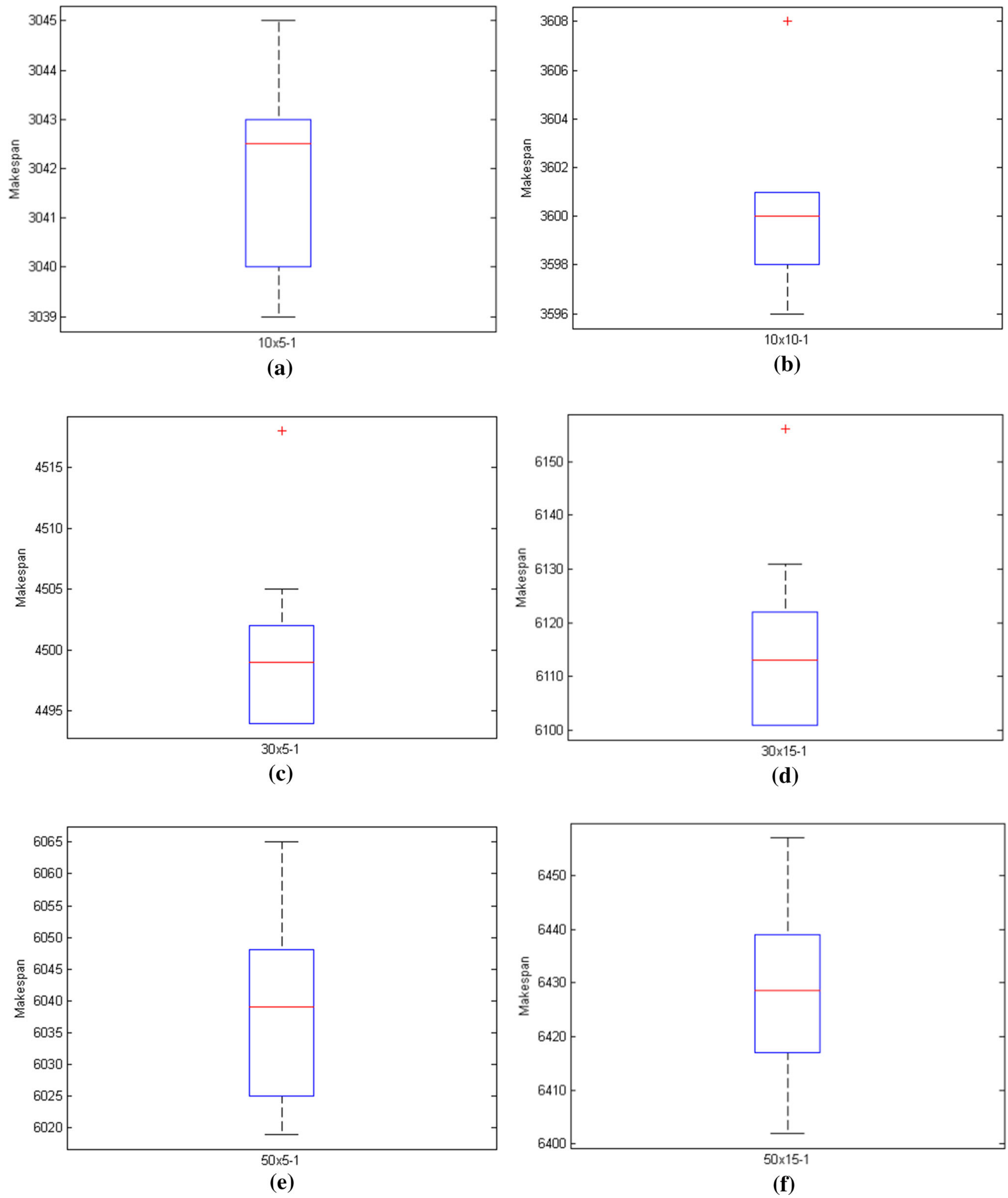

Fig. 4 Box plots multiple performances on six tests: $10 \times 5 - 1$ (**a**), $10 \times 10 - 1$ (**b**), $30 \times 5 - 1$ (**c**), $30 \times 15 - 1$ (**d**), $50 \times 5 - 1$ (**e**), $50 \times 15 - 1$ (**f**)

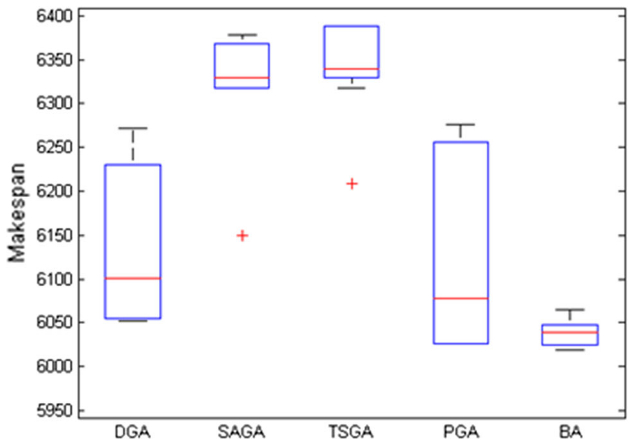

Fig. 5 Box plot of running of five different methods

Compliance with ethical standards

Conflict of interest In the present work, we have not used any material from previously published. So we have no conflict of interest.

References

1. Li X, Gao L (2016) An effective hybrid genetic algorithm and tabu search for flexible job shop scheduling problem. Int J Prod Econ 174:93–110
2. Tellache NEH, Boudhar M (2017) Open shop scheduling problems with conflict graphs. Discr Appl Math 227:103–120
3. Noori-Darvish S, Tavakkoli-Moghaddam R (2011) Solving a bi-objective open shop scheduling problem with fuzzy parameters. J Appl Oper Res 3(2):59–74
4. Ciro GC, Dugardin F, Yalaoui F, Kelly R (2016) A NSGA-II and NSGA-III comparison for solving an open shop scheduling problem with resource constraints. IFAC-PapersOnLine 49(12):1272–1277
5. TsoLin H, TauLee H, Pan W (2008) Heuristics for scheduling in a no-wait open shop with movable dedicated machines. Int J Prod Econ 111:368–377
6. Shamshirband S, Shojafar M, Hosseinabadi AR, Kardgar M, Nasir MN, Ahmad R (2015) OSGA: genetic-based open-shop scheduling with consideration of machine maintenance in small and medium enterprises. Ann Oper Res 229(1):743–758
7. Gonzalez T, Sahni S (1976) Open shop scheduling to minimize finish time. J ACM (JACM) 23(4):665–679
8. Kubiak W, Sriskandarajah C, Zaras K (1991) A note on the complexity of openshop scheduling problems. INFOR Inf Syst Oper Res 29(4):89–294
9. Liu CY, Bulfin RL (1987) Scheduling ordered open shops. Comput Oper Res 14(3):257–264
10. Prins C (1994) An overview of scheduling problems arising in satellite communications. J Oper Res Soc 45(6):611–623
11. Brucker P (2007) Scheduling algorithms. Springer, Berlin
12. Kolen AW, Lenstra JK, Papadimitriou CH, Spieksma FC (2007) Interval scheduling: a survey. Nav Res Logist (NRL) 54(5):530–543
13. Reddy MS, Ratnam C, Rajyalakshmi G, Manupati VK (2018) An effective hybrid multi objective evolutionary algorithm for solving real time event in flexible job shop scheduling problem. Measurement 114:78–90
14. Hosseinabadi AAR, Farahabadi AB, Rostami MHS, Lateran AF (2013) Presentation of a new and beneficial method through problem solving timing of open shop by random algorithm gravitational emulation local search. Int J Comput Sci Issues (IJCSI) 10(1):745–751
15. Tavakkolai H, Hosseinabadi AAR, Yadollahi M, Mohammadpour T (2015) Using gravitational search algorithm for in advance reservation of resources in solving the scheduling problem of works in workflow workshop environment. Indian J Sci Technol 8(11):1–16
16. Farahabadi AB, Hosseinabadi AR (2013) Present a new hybrid algorithm scheduling flexible manufacturing system consideration cost maintenance. Int J Sci Eng Res 4(9):1870–1875
17. Michael RG, David SJ (1979) Computers and intractability: a guide to the theory of NP-completeness. WH Free. Co., San Francisco
18. Ciro GC, Dugardin F, Yalaoui F, Kelly R (2015) A fuzzy ant colony optimization to solve an open shop scheduling problem with multi-skills resource constraints. IFAC-PapersOnLine 48(3):715–720
19. Tautenhahn T, Woeginger GJ (1997) Unit-time scheduling problems with time dependent resources. Computing 58(2):97–111
20. Brasel H, Herms A, Morig M, Tautenhahn T, Tusch J, Werner F (2008) Heuristic constructive algorithms for open shop scheduling to minimize mean flow time. Eur J Oper Res 189(3):856–870
21. Alcaide D, Sicilia J, Vigo D (1997) A tabu search algorithm for the open shop problem. Top 5(2):283–296
22. Liaw CF (2000) A hybrid genetic algorithm for the open shop scheduling problem. Eur J Oper Res 124(1):28–42
23. Blum C (2005) Beam-ACO-Hybridizing ant colony optimization with beam search: an application to open shop scheduling. Comput Oper Res 32(6):1565–1591
24. Huang YM, Lin JC (2011) A new bee colony optimization algorithm with idle-time-based filtering scheme for open shop-scheduling problems. Exp Syst Appl 38(5):5438–5447
25. Ahmadizar F, Farahani MH (2012) A novel hybrid genetic algorithm for the open shop scheduling problem. Int J Adv Manuf Technol 62(5–8):775–787
26. Chen Y, Zhang A, Chen G, Dong J (2013) Approximation algorithms for parallel open shop scheduling. Inf Process Lett 113(7):220–224
27. Naderi B, Zandieh M (2014) Modeling and scheduling no-wait open shop problems. Int J Prod Econ 158:256–266
28. Bai D, Zhang ZH, Zhang Q (2016) Flexible open shop scheduling problem to minimize makespan. Comput Oper Res 67:207–215
29. Tanimizu Y, Sakamoto M, Nonomiya H (2017) A co-evolutionary algorithm for open-shop scheduling with disassembly operations. Procedia CIRP 63:289–294
30. Gao KZ, Suganthan PN, Pan QK, Chua TJ, Chong CS, Cai TX (2016) An improved artificial bee colony algorithm for flexible

job-shop scheduling problem with fuzzy processing time. Exp Syst Appl 65:52–67
31. Zhao F, Liu H, Zhang Y, Ma W, Zhang C (2018) A discrete water wave optimization algorithm for no-wait flow shop scheduling problem. Exp Syst Appl 91:347–363
32. Marichelvam MK, Prabaharan T, Yang XS, Geetha M (2013) Solving hybrid flow shop scheduling problems using bat algorithm. Int J Logist Econ Glob 5(1):15–29
33. Oulamara A, Rebaine D, Serairi M (2013) Scheduling the two-machine open shop problem under resource constraints for setting the jobs. Ann Oper Res 211(1):333–356
34. Zaher H, Ragaa N, Sayed H (2017) A novel improved bat algorithm for job shop scheduling problem. Int J Comput Appl 164(5):24–30
35. Goldansaz SM, Jolai F, Anaraki AHZ (2013) A hybrid imperialist competitive algorithm for minimizing makespan in a multi-processor open shop. Appl Math Model 37(23):9603–9616
36. Koulamas C, Kyparisis GJ (2015) The three-machine proportionate open shop and mixed shop minimum makespan problems. Eur J Oper Res 243(1):70–74
37. Low C, Yeh Y (2009) Genetic algorithm-based heuristics for an open shop scheduling problem with setup, processing, and removal times separated. Robot Comput Integr Manuf 25(2):314–322
38. Zhang ZH, Bai D (2014) An extended study on an open-shop scheduling problem using the minimisation of the sum of quadratic completion times. Appl Math Comput 230:238–247
39. Chen Y, Goebel R, Lin G, Su B, Zhang A (2020) Open-shop scheduling for unit jobs under precedence constraints. Theor Comput Sci 803:144–151
40. Wei W, Song H, Li W, Shen P, Vasilakos A (2017) Gradient-driven parking navigation using a continuous information potential field based on wireless sensor network. Inf Sci 408:100–114
41. Mejía G, Yuraszeck F (2020) A self-tuning variable neighborhood search algorithm and an effective decoding scheme for open shop scheduling problems with travel/setup times. Eur J Oper Res 285(2):484–496
42. Wei W, Xu Q, Wang L, Hei XH, Shen P, Shi W, Shan L (2014) GI/Geom/1 queue based on communication model for mesh networks. Int J Commun Syst 27(11):3013–3029
43. Abdelmaguid TF (2020) Scatter search with path relinking for multiprocessor open shop scheduling. Comput Ind Eng 141:106292
44. Wei W, Zhou B, Polap D, Wozniak M (2019) A regional adaptive variational PDE model for computed tomography image reconstruction. Pattern Recognit 92:64–81
45. Yang XS (2010) A new metaheuristic bat-inspired algorithm. In: Proceedings of the of nature inspired cooperative strategies for optimization (NICSO 2010). Springer, Berlin, Heidelberg, pp 65–74
46. Yang XS (2011) Bat algorithm for multi-objective optimisation. Int J Bio-Inspired Comput 3(5):267–274
47. Yang XS, He X (2013) Bat algorithm: literature review and applications. Int J Bio-Inspired Comput 5(3):141–149
48. Basetti V, Chandel AK (2017) Optimal PMU placement for power system observability using Taguchi binary bat algorithm. Measurement 95:8–20
49. Chen S, Peng GH, He XS, Yang XS (2018) Global convergence analysis of the bat algorithm using a markovian framework and dynamical system theory. Exp Syst Appl 114:173–182
50. Taillard E (1993) Benchmarks for basic scheduling problems. Eur J Oper Res 64(2):278–285
51. Harmanani HM, Ghosn SB (2016) An efficient method for the open-shop scheduling problem using simulated annealing. In: Information technology: new generations. Springer, Cham, pp 1183–1193
52. Khuri S, Miryala SR (1999) Genetic algorithms for solving open shop scheduling problems. In: Proceedings of the of Portuguese conference on artificial intelligence. Springer, Berlin, Heidelberg, pp 357–368
53. Ross HLFP, Corne D (1994) A promising hybrid GA/heuristic approach for open-shop scheduling problems. In: Proceedings of the 11th European conference on artificial intelligence, pp 590–594
54. Prins C (2000) Competitive genetic algorithms for the open-shop scheduling problem. Math Methods Oper Res 52(3):389–411
55. Colak S, Agarwal A (2005) Non-greedy heuristics and augmented neural networks for the open-shop scheduling problem. Nav Res Logist (NRL) 52(7):631–644
56. Marrouche W, Harmanani HM (2018) Heuristic approaches for the open-shop scheduling problem. In: Information technology—new generations, pp 691–699
57. Bouzidi A, Essaid Riffi M, Barkatou M (2019) Cat swarm optimization for solving the open shop scheduling problem. J Ind Eng Int 15(2):367–378
58. Hosseinabadi AAR, Vahidi J, Saemi B, Sangaiah AK, Elhoseny M (2019) Extended genetic algorithm for solving open-shop scheduling problem. Soft Comput 23(13):5099–5116
59. Van Laarhoven PJ, Aarts EH (1987) Simulated annealing. In: Aarts EHL (ed) Simulated annealing: theory and applications. Springer, Dordrecht, pp 7–15
60. Glover F, Laguna M (1998) Tabu search. In: Pardalos PM, Du DZ, Graham RL (eds) Handbook of combinatorial optimization. Springer, Boston, pp 2093–2229