Hindawi
Complexity
Volume 2022, Article ID 5043058, 16 pages
https://doi.org/10.1155/2022/5043058

Research Article

Multiobjective Parallel Algorithms for Solving Biobjective Open Shop Scheduling Problem

Seyed Hassan Shams Lahroudi,[1] **Farzaneh Mahalleh**,[2] **and Seyedsaeid Mirkamali**[3]

[1]*Department of Industrial Management, Science and Research Branch, Islamic Azad University, Tehran, Iran*
[2]*Department of Computer, Faculty of Khoy, West Azarbijan Branch, Technical and Vocational University (TVU), Khoy, Iran*
[3]*Department of Computer Engineering and IT, Payame Noor University (PNU), Tehran, Iran*

Correspondence should be addressed to Seyedsaeid Mirkamali; s.mirkamali@pnu.ac.ir

Received 18 February 2022; Revised 15 May 2022; Accepted 5 July 2022; Published 31 August 2022

Academic Editor: Hui Xiao

Open Shop Scheduling Problem (OSSP) is one of the most important scheduling problems in the field of engineering and industry. This kind of problem includes m machines and n jobs, each job contains a certain number of operations, and each operation has a predetermined processing time on its corresponding machine. The order of processing of these operations affects the completion times of all jobs. Therefore, the purpose of OSSP is to achieve a proper order of processing of jobs using specified machines, so that the completion time of all jobs is minimized. In this paper, the strengths and limitations of three methods are evaluated by comparing the results of solving the OSSP in large-scale and small-scale benchmarks. In this case, the minimized completion time and total tardiness are considered the objective functions of the adapted methods. To solve small-scale problems, we adapt a mathematical model called Multiobjective Mixed Linear Programming (MOMILP). To solve large-scale problems, two meta-heuristic algorithms including Multiobjective Parallel Genetic Algorithm (MOPGA) and Multiobjective Parallel Simulated Annealing (MOPSA) are adapted. In experimental results, we randomly generated small-scale problems to compare MOMILP with the Genetic Algorithm (GA) and Simulate Annealing (SA). To compare MOPSA and MOPGA with the state of the art and show their strengths and limitations, we use a standard large-scale benchmark. The simulation results of the proposed algorithms show that although the MOPSA algorithm is faster, the MOPGA algorithm is more efficient in achieving optimal solutions for large-scale problems compared with other methods.

1. Introduction

Optimal job scheduling problems are classified into single machine scheduling, job shop scheduling, open shop scheduling, flow shop scheduling, and hybrid scheduling problems; where the single machine scheduling is the simplest problem of scheduling. The problem is to schedule the processing of n jobs with varying processing times on a single machine [1]. In the Job Shop Scheduling Problem (JSSP), the problem is more complex. Here, each job consists of a set of operations ($O1$, $O2$, . . ., O_n) that need to be processed in a specific order. Each operation has a unique machine on which it must be performed, and only one operation per task may be performed at the moment [2]. A more complex problem than JSSP is the Open Shop Scheduling Problem (OSSP). OSSP involves m machines ($M_1, M_2, \ldots, M_m$) to perform n jobs ($J_1, J_2, \ldots, J_n$), each job involving m operations. Jth operation of J_j is indicated by O_{ij} which should be processed on machine M_i for $p_{ij} \geq 0$ units of time. Therefore, we will have a total of $n \times m$ operations ($O_{11}, O_{12}, \ldots, O_{nm}$) [3]. Like OSSP, each job in the Flow Shop Scheduling Problem (FSSP) consists of precisely m operations. However, the ith operation of the job must be executed on the ith machine. No machine is capable of doing many operations simultaneously. The execution time for each action of each task is provided earlier [4]. The most complex problem in the class of job scheduling problems is the hybrid scheduling problem (HSFP). In this problem, n jobs ($J_1, J_2, \ldots, J_n$), must be processed in a multistage manufacturing facility in which each step has several concurrent and identical machines [5].

This paper addresses the problem of Biobjective Open Shop Scheduling. In case of complexity of the problem, the OSSP is in between the job shop scheduling and the flow shop scheduling problems. However, all the problems in this class are considered as NP-Hard problems. The OSSP has a broad range of applications across different sectors that include timetabling, Satellite communication, health care management, transportation, tourism, computer vision, and many other applications. In the following subsection, we introduce some related works that tried to solve the problem of open shop scheduling.

1.1. Related Works. Production scheduling problems have been widely studied in the past years. A detailed review of this type of research work can be found in [6, 7]. Breit et al. [8] solved the OSSP of two machines with a predetermined stop time for one of the machines and proposed a heuristic algorithm. Next, they set some predetermined stop times for both machines and again used heuristic algorithms to approximate the problem. Hsu et al. [9] examined two different periodic maintenance approaches simultaneously in the single-machine scheduling problem. Periodic maintenance approaches considered for their problem are such that the machine stops after a certain period of time (T) or after processing a certain number of jobs (K) in order to perform the maintenance process. Obviously, if either of these conditions occurs earlier, the machine will stop.

To quantitatively characterize the production and distribution optimization issue, Fu et al. [10] formulate a mixed-integer programming model to minimize the maximum completion time. To best handle the presented issue, an improved black widow optimization method is created to address the researched problem. In their suggested method, the solution representation, population initiation, procreation, cannibalism, and mutation, as well as a simulated annealing method, are uniquely constructed.

Another condition that arises in the presence of machines' unavailability constraint is the impermissibility of cutting work due to which in the absence of sufficient time to perform an operation and face the machine stop, the desired operation is not loaded on the machine and its processing is postponed until machine stop is finished. Transportation and movement times between machines are other criteria that have been considered in scheduling problems. Strusevich [11] considered a time interval between the completion of the processing of one activity of a job on one machine and the beginning of the next activity of the same job on another machine for two-machine OSSP and due to occurring transportation in real conditions named it transportation time. He also points out that in chemical and metallurgical applications, these transportation times are equivalent to the times of the heating or cooling processes.

Recently, Fu et al. [12] offer a stochastic biobjective two-stage open shop scheduling problem that mimics a car repair process where duties are delegated to different third-party organizations with specialized equipment. The optimization issue was formulated by reducing the overall lateness and processing costs, subject to different resource restrictions. In order to tackle the open shop issue, a hybrid multiobjective optimization of migratory birds, along with a genetic operation and discrete event system, is developed by considering problem features.

The OSSP is known as an NP-hard problem [13–15] and it is not possible to solve these problems in polynomial time except in small dimensions. Therefore, approximate solution achievements including heuristic and metaheuristic methods can be more efficient than exact methods. Panahi et al. [16] used multiobjective simulated annealing and Ant Colony Optimization (ACO) algorithm to solve Multiobjective Open Shop Scheduling (MOOSS) with the objectives of total delay and the longest completion time. Panahi and Tavakkoli-Moghaddam [17] combined general and multiobjective simulated annealing algorithms with ACO and used it to simultaneous optimization of two objective functions including the longest completion time and total delay in OSSP. Drezner [18] used the Genetic Algorithm (GA) and Simulated Annealing (SA) algorithm for batch scheduling in the case of multifunction parallel machines.

Task scheduling has been studied in much different research works with an array of different applications from task scheduling in cloud computing [19] to the flow shops in factories [20, 21]. Most studies on scheduling problems have been conducted under the basic assumption that machines are available on all planning horizons, while this is not the case in the real world. Machines may not be available for reasons such as preventive maintenance, failure, rest periods, and residual work from the previous planning period that should be processed at the beginning of the new period, [22]. In many cases, unavailability times are known in advance, so the decision-maker can make a more efficient decision by considering them. Karimi et al. [23] considered a predetermined time for machine unavailability in the single machine problem and proposed two heuristic methods to solve it.

In this paper, an OSSP is addressed in which stops occur at different times on machines and its duration is different but specific for different machines.

The intended objective functions include minimizing the longest completion time ($C_{\max}$) and the sum of delays, which are presented as a weight combination in a single function. Also, due to the possibility of using different devices to transport different items in a shop, the movement time in a fixed route is different for each job. Also, the working path from one machine to another is considered different from its return path due to the minimization of material flow interference. Therefore, the work transferring matrix is asymmetric between different workstations. In order to solve problems with small dimensions and when the weight of each target is known, the exact approach of the mixed biobjective linear mathematical model is adapted to use as a problem-solving method, and later the results are compared with the results of GA and SA in solving small-scale OSSP. On the contrary, when the weight of each target is unknown, the achievement of Pareto optimal solutions using metaheuristic algorithms including Multiobjective Parallel Genetic Algorithm (MOPGA) and Multiobjective Parallel Simulated Annealing (MOPSA) algorithm are adapted to solve large-scale problems. The parameters of the metaheuristic algorithms are adjusted using the Taguchi experimental design method, as a result of which, algorithms show a state of stability in the face of various problems.

1.2. The Contributions of the Study to the Literature. This study has various contributions to the literature from various aspects. These are as follows:

(i) The exact approach of the mixed multiobjective linear mathematical model is adapted to use as a problem-solving method, and later the results are compared with the results of GA and SA in solving small-scale OSSP.

(ii) Multiobjective Parallel Genetic Algorithm (MOPGA) and Multiobjective Parallel Simulated Annealing (MOPSA) methods are adapted to solve large-scale OSSP.

(iii) To improve the exploitation and exploration capabilities of MOPGA, the new candidate generation stage of MOPGA was developed by changing crossover and mutation operators.

(iv) Variations in MOPGA and MOPSA were compared in detail with various heuristic algorithms selected from the literature according to the average RPD, execution time, and best solution criteria.

In the following, Section 2 presents a mixed linear mathematical programming model. The adapted metaheuristic algorithms and their details are described in Section 3. Section 4 describes the design of experiments and computational evaluation. Section 5 provides a conclusion and future studies.

2. Mathematical Model

In this section, the problem is introduced using simplifying assumptions as well as mathematical symbols and then a multiobjective mixed linear programming model is presented and analyzed to accurately solve small-scale problems.

To achieve the mathematical model, the available time of each machine is considered as a packet of a certain length T_i and jobs should be placed in such a way that their total processing time does not exceed the length of the packet. To clarify the issues, see Figure 1, in which $J_{[j]}$ indicates j^{th} job in sequence and B_{il} indicates l^{th} packet of machine i.

2.1. Problem Assumptions. Simplifying assumptions are considered as follows:

(i) Each machine may process up to one task at a time.

(ii) Each machine may process up to one task at a time.

(iii) Jobs can be processed in any order by all or some machines.

(iv) All jobs are available in the shop from the beginning of the planning horizon.

(v) Job cutting is not allowed during processing.

(vi) There is only one machine of each type in the shop (operations of each job are performed by only one specific machine).

(vii) Unavailable times occur at specific times on each machine.

The length of available (and unavailable) times of machines is specific and depends on the machine and is fixed during the planning period.

2.2. Indexes and Parameters

j, k: indexes of job (j, $k = 1,2, \ldots, n$)

i, h: indexes of machines (i, $h = 1,2, \ldots, m$)

l, l': indexes of packets (l, $l' = 1,2, \ldots, b$)

p_{ij}: processing time of job j on machine i

Tr_{ihj}: job transferring time from machine i to machine h

Ti: the length of the available time of machine i

ti: the length of unavailable time of machine i

dj: delivery date of job j

2.3. Decision Variables

st_{ij}: start time of preparing job j on machine i

C_{ij}: completion time of job j on machine i

C_j: completion time of job j on the last machine according to the sequence.

Tard_j: tardiness of completion time of job j from the delivery date

$$Y_{ihj} = \begin{cases} 1 & \text{if job } j \text{ on machine } i \text{ is processed before machine } h \\ 0 & \text{otherwise} \end{cases}$$

$$X_{ijk} = \begin{cases} 1, & \text{if job } j \text{ is processed on machine } i \text{ before job } k \\ 0, & \text{otherwise} \end{cases}$$

$$A_{ijl} = \begin{cases} 1, & \text{if job } j \text{ is located in packet } l \text{ of machine } i \ (Bil) \\ 0, & \text{otherwise} \end{cases}$$

2.4. Multiobjective Mixed Linear Programming Model. In this section, using the above assumptions and symbols, a Multiobjective Mixed Linear Programming Model (MOMILP) is adapted to solve the open shop scheduling problem, taking into account the unavailability of machines, preparation and separation times of jobs and machines, and transmission times between machines to simultaneously minimize the criteria of the longest completion time and the sum of delays as a single function. The MOMILP model is as follows:

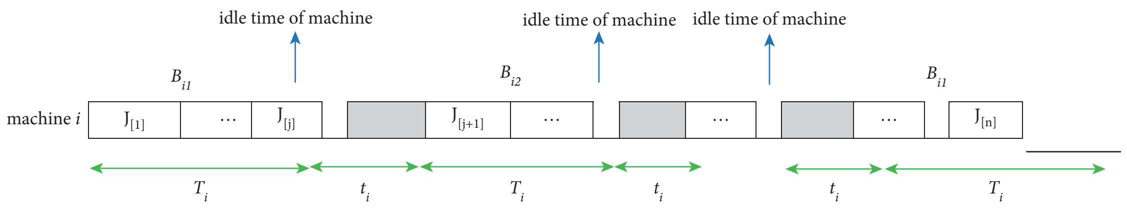

Figure 1: A sequence of jobs on the machine.

$$\min z = \theta_1 C_{\max} + \theta_2 \sum_{j=1}^{n} Tard_j, \tag{1}$$

$$st_{ij} + pi_j = c_{ij} \forall j, jst_{ij} + pi_j = Tr_{ihj} \forall j, h, j; \tag{2}$$

$$M\left(1 - Y_{ihj}\right) \le st_{hj} i \ne h, \tag{3}$$

$$\begin{aligned} &\le st_{ik} - M\left(1 - A_{ikl}\right) - M\left(1 - X_{ijk}\right) j \ne k\left(st_{ij} + p_{ij}\right) \\ &\quad - M\left(1 - A_{ijl}\right) \forall i, j, k, l; \end{aligned} \tag{4}$$

$$st_{ij} \ge (l-1)\left(T_i + t_i\right) A_{ijl} \forall_{i,j,l}, \tag{5}$$

$$C_{ij} - M\left(1 - A_{ijl}\right) \le (l-1)t_i + l.T_i \forall_{i,j,l}, \tag{6}$$

$$C_{ij} + M\left(1 - A_{ijl}\right) \ge (l-1)\left(t_i + T_i\right) \forall i, j, l, \tag{7}$$

$$M\left(1 - A_{ijl}\right) + M\left(1 - A_{ikl}{'}\right) + X_{ijk} \ge 1 \quad \begin{matrix} \forall i, j, k, l, l' \\ j \ne k, \\ l < b, l' \end{matrix} \tag{8}$$

$$X_{ijk} + X_{ikj} = 1 \begin{matrix} \forall i, k, j; \\ j < n, k < j \end{matrix}, \tag{9}$$

$$Y_{ihj} + Y_{hij} = 1 \begin{matrix} \forall i, j, h; \\ i \langle m, h \rangle i \end{matrix}, \tag{10}$$

$$\sum_{l=1}^{b} A_{ijl} = 1 \forall i, j, \tag{11}$$

$$\sum_{j=1}^{n} p_{ij}.A_{ijl} \le T_i \, \forall i, l, \tag{12}$$

$$C_i \ge C_{ij} \, \forall i, j, \tag{13}$$

$$C_{\max} \ge C_{ij} \forall i, \tag{14}$$

$$Tard_j \ge C_j - d_j \, \forall j, \tag{15}$$

$$st_{ij}, C_{ij}, C_j, C_{\max}, Tard_j \ge 0_{\forall i,j}, \tag{16}$$

$$X_{ijk}, Y_{ihj}, A_{ijl} \in \{0, 1\} \begin{matrix} \forall i, h, j, k, l \\ \ne h, j \\ \ne k \end{matrix} \tag{17}$$

Equation (1) is the objective function of the problem, which includes the convex linear combination of the two criteria: (1) the longest completion time and (2) the sum of the delays.

θ_1 and θ_2 are positive coefficients that represent the weight of each target and $\theta_1 + \theta_2 = 1$. The completion time of each operation (C_{ij}) is obtained by equation (2). If job j on machine i is processed before machine h, we have $Y_{ihj} = 1$, then the start time of preparing job j on machine h ($st_{hj} = 1$) will be larger than the completion time of that job on machine i, this is shown using equation (3). If job j is processed on machine i before job k ($X_{ijk} = 1$) and both jobs are located in the same packet l ($A_{ijl} = A_{ikl} = 1$), then the start time of preparing job k on machine i ($st_{ik} = 1$) will be larger than the completion time of job j on that machine, this criterion is shown in equation (4). Consequently, equation (5) ensures that no preparation takes place when machines are unavailable. Equations (6) and (7) together prevent the completion of operations during machines are unavailable. Equations (8) and (9) simultaneously specify the sequence of jobs on each machine and packets. Equation (10) determines the order of the machines for the operations of each job. Equation (11) ensures that each operation takes place in exactly one packet of each machine. Equation (12) controls the sum of the total processing times of the operations contained in a packet according to the maximum time of that packet, which is the access time of the relevant machine (T_i). The completion time of each job is calculated through equation (13) and according to the objective function. Equations (14) and (15) calculate the longest completion time and the delay time of each operation, respectively. Finally, equations (16) and (17) represent the negative variables and binary variables (0 or 1).

2.5. An Example.

Here, to clarify the concept of the problem, we provide an example and solve it using the above MOMILP model. This example includes five jobs ($n = 5$), two machines ($m = 2$), and five packets for each machine ($b = 5$). The rest of the parameters are as follows:

$$
\begin{aligned}
&p_{ij} = \begin{bmatrix} 6 & 11 \\ 12 & 16 \\ 15 & 14 \\ 11 & 15 \\ 9 & 15 \end{bmatrix}, \\
&Tr_{ih1} = \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix}, \\
&Tr_{ih2} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \\
&T_i = \begin{pmatrix} 26 & 16 \end{pmatrix}, t_i = \begin{pmatrix} 2 & 4 \end{pmatrix}, d_j, \\
&Tr_{ih3} = \begin{bmatrix} 0 & 2 \\ 2 & 0 \end{bmatrix}, Tr_{ih4} = \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix}, Tr_{ih5} = \begin{bmatrix} 0 & 0 \\ 2 & 0 \end{bmatrix} \\
&T_i = \begin{pmatrix} 26 & 16 \end{pmatrix}, t_i = \begin{pmatrix} 2 & 4 \end{pmatrix}, d_j \\
&T_i = \begin{pmatrix} 26 & 16 \end{pmatrix}, t_i = \begin{pmatrix} 2 & 4 \end{pmatrix}, d_j \\
&\quad = \begin{pmatrix} 19 & 48 & 60 & 37 & 28 \end{pmatrix}
\end{aligned}
\tag{18}
$$

Considering the weight of the targets $\theta_1 = \theta_2 = 0.5$, the global optimal solution $Z = 90.5$ is obtained by solving the MOMILP model in Lingo 8 software; the Gantt chart of which is depicted in Figure 2. As can be seen, no operation is performed during the unavailable times of machines, which are shown in gray. Thus, our main goal of modeling this design has been met. In addition, the sum of the total processing times associated with the operations contained in each packet does not exceed the time of that packet. Also, the obtained sequence indicates that the delivery date has been observed so that the jobs with an earlier delivery time are at the beginning of the sequence and the jobs with a later delivery time are at the end of the sequence. Another noteworthy point is the observance of transfer times; as an example, we can point to the distance between the completion of job J_1 on machine M_2 and starting this job on the next machine ($Tr_{211} = 2$).

2.6. Model Analysis. In this section, the sensitivity of the MOMILP model to the number of jobs (n), machines (m), and packets (b) is investigated. To do so, all indexes related to variables and constraints are shown in Tables 1 and 2, respectively. The effect of increasing the number of jobs, machines, and packets on the number of variables and constraints can also be seen in Table 3.

3. The Genetic Algorithm-Based Method

As mentioned in the introduction, the OSSP is known as an NP-hard problem and only small-scale problems can be solved accurately in a reasonable amount of time [24]. Although the proposed mathematical model achieves an accurate solution, its efficiency decreases as the dimensions of the problem increase. Hence, heuristic methods or approximate algorithms are more effective for solving medium- and large-scale problems that commonly occur in the real world.

Genetic algorithms (GAs) are a class of general-purpose search strategies based on natural selection and genetics. GAs have been used effectively to a wide array of optimization challenges [19, 25, 26]. In contrast to local search algorithms such as SA, which are often focused on manipulating a single viable solution and are extremely quick, GAs retain and modify a population of possible solutions. Despite the fact that GAs have shown to be a diverse and successful search tool for addressing optimization issues, there are still several scenarios in which the basic GA performs poorly. Therefore, several hybridization methods have been reported in the literature. In this section, we also propose to use the Multiobjective Parallel Genetic Algorithm to solve the OSSP.

3.1. Chromosome Display. Chromosomes are in fact encoded solutions and points in solution space. In this paper, each operation is considered as a gene and each chromosome (solution) is represented as a permutation of genes (operations). For example, consider a problem with five jobs and 2 machines, and so 10 operations. A chromosome (or an encoded solution) can be looked at in Table 4 for reference.where O_{ij} is the operation of job j on machine i. According to this chromosome, operation O_{21} will be processed before operation O_{11} (note that these two operations are related to the same job) and operation O_{14} will be processed before operation O_{11} (note that these two operations are related to the same machine).

3.2. Initial Population. Scattering in generations prevents rapid convergence and local optimums. Therefore, we randomly generate the initial population (initial generation) to maintain the scattering of solutions in the solution space as much as possible. That is, a random permutation from the set $\{1, 2, \ldots, m.n\}$ is considered as an individual where $m.n$ indicates the number of operations.

3.3. Objective Function. In order to calculate the objective function for each chromosome (solution space points), these so-called coded solutions should be decoded. For this purpose, we place the operations on the machines in ascending order from the first operation to the last one; taking into account their unavailable times as well, we calculate their completion time. Then, taking into account the completion time of all operations, the longest completion time is obtained $C_{\max} = \max_{i,j}\{C_{ij}\}$.

Also, in order to calculate the second expression of the objective function, i.e., the sum of the delays, first the completion time of each job is calculated as $C_{\max} = C_j = \max_i\{C_{ij}\}$ and then the delay of that job is obtained by $\text{Tard}_j = \max[0, C_j - d_j]$. Finally, the sum of delays is calculated as $\sum_{j=1}^{n} \text{Tard}_j$. How to determine the values of the coefficients θ_1 and θ_2 is explained in the next sections. Therefore, the objective function of each chromosome is calculated as $Z = \theta_1 C_{\max} + \theta_2 \sum_{j=1}^{n} \text{Tard}_j$.

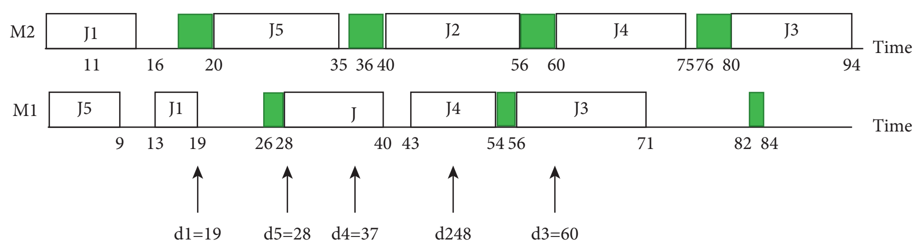

Figure 2: Gantt chart of optimal solving of the example.

Table 1: Number of variables.

Variable	Number
Z, $C_{\max}$	1
Cj, Tard_j	n
St_{ij}, C_{ij}	mn
X_{ijk}	$mn\ (n-1)$
Y_{ijk}	$mn\ (m-1)$
A_{ijl}	mnb

Table 2: Number of constraints.

Constraint	Number
(1)	1
(2), (11), (13)	mn
(3)	$mn\ (m-1)$
(4)	$mbn\ (n-1)$
(5), (6), (7)	mnb
(8)	$mn\ (n-1)\ b\ (b\text{-}1)/2$
(9)	$mn\ (n-1)/2$
(10)	$mn\ (m\text{-}1)/2$
(12)	mb
(14), (15)	n

Table 3: Number of variables and constraints according to the model MOMILP.

Problem size ($m.n.b$)	Number of variables	Number of constraints
2, 4, 4	61	393
3, 4, 4	95	607
2, 5, 5	91	826
3, 5, 5	141	1261
5, 30, 30	7156	2039926
10, 30, 30	15031	4082101
5, 40, 40	12541	6425901
10, 40, 40	26041	12854801

Table 4: An example of a chromosome for a problem with 5 jobs and 2 machines, and 10 operations.

O_{11}	O_{12}	O_{13}	O_{14}	O_{15}	O_{21}	O_{22}	O_{23}	O_{24}	O_{25}
3	6	8	2	5	1	9	10	4	7

3.4. Selection. Selection is the process by which individuals present in a generation are selected in pairs for mating. There are various approaches for selection in the literature [27]. In this paper, we use the roulette wheel selection approach, according to which individuals with higher fitness have a better chance of mating and this is in fact the basis of Darwin's theory. In order to apply the roulette wheel selection, two parameters are required: individual selection probability and individual cumulative probability, which are obtained from equations (18) and (19), respectively:

$$PS_i = \frac{f_i}{\sum_{j=1}^{\text{popsize}} f_j}, \tag{19}$$

$$CP_i = \sum_{j=1}^{i} PS_j. \tag{20}$$

According to the roulette wheel selection approach, a number is randomly generated in the range [0, 1], then individual i that meets the condition $CP_{i-1} < r < CP_i$ is selected as one of the pairs. This process is repeated for the second one.

3.5. Crossover Operation. After selecting a pair of parents using one of the selection methods, a genetic crossover operator with the probability p_c is used to combine the two parents and generate two children. There are several crossover techniques and combining parents: single-point crossover, two-point crossover, multipoint crossover, uniform crossover, and so on [28]. However, the main problem with the crossover operation is that the feasibility of new solutions (newborns) may not be guaranteed. In cases where new chromosomes or newborns produce infeasible solutions, corrective action is usually taken to turn them into feasible solutions, which will lengthen the solving time, by maintaining the feasibility of new chromosomes in each generation and without any modification process [29]. In the following, the crossover operation is described in four steps:

Step 1: Select two operations randomly from chromosome 1 (parent).

Step 2: Transfer the genes from the two selected operations along with all the genes between them from parent 1 to child 1 while maintaining their location.

Step 3: Select the rest of the genes needed by child 1 from left to right until creating a complete chromosome and place them the same way from left to right in the empty spaces of child 1.

Step 4: To create the second child, replace the two parents 1 and 2 and repeat steps 2 and 3 with the genes of the same operations selected in step 1.

Table 5: Chromosomes of parents 1 and 2.

Parent 1	3	6	8	2	5	1	9	10	4	7
Parent 2	4	9	5	6	2	1	7	10	8	3

Table 6: Transfer the genes from the two selected operations along with all the genes between them from parent 1 to child 1 while maintaining their location.

Parent 1	3	6	8	2	5	1	9	10	4	7
Parent 2	4	9	5	6	2	1	7	10	8	3

Child 1				2	5	1	9			

Table 7: Select the rest of the genes needed by child 1 from parent 2 and place them in the empty spaces of child 1.

Child 1	4	6	7	2	5	1	9	10	8	3

Table 8: Exchange the two parents 1 and 2 and repeat steps 2 and 3 until creating child 2.

Child 1	3	8	5	6	2	1	7	9	10	4

For example, consider the two parents in Tables 5–8.

Step 1. Suppose the two operations selected by the parent 1 are O_{14} and O_{22}, in other words, genes 2 and 9, respectively.

Step 2. Transfer genes 2, 5, 1, and 9 from parent 1 to child 1 while maintaining their locations.

Step 3. Select the rest of the genes needed by child 1 (from left to right) from parent 2 and place them (from left to right) in the empty spaces of child 1.

Step 4. Exchange the two parents 1 and 2 and repeat steps 2 and 3 until creating child 2 as follows:

3.6. Mutation

3.6. Mutation. In order to prevent the generated generations from being directed toward the local optimums, a mutation operator with a probability p_m is applied to each of the produced children. This operator exchanges the two randomly selected operations and their genes [29]. For example, suppose that operations O_{12} and O_{24} are selected from child 1. Then, genes 6 and 8 are exchanged and mutated child will be simillar to the result given in Table 9)

3.7. Termination Condition. The GA continues until a termination criterion is met, which is *m.n.εs* from the beginning and produces new generations [30]. Where ε is a coefficient whose value is adjusted through experiments. In the following sections, we explain how to set it up. Also, the reason for choosing this termination condition is that it gives the algorithm more time to solve larger problems.

Table 9: The genes 6 and 8 are exchanged and mutated is generated.

Child 1	4	6	7	2	5	1	9	10	8	3
Mutated Child 1	4	8	7	2	5	1	9	10	6	3

3.8. Primary Genetic Algorithm. Before presenting the GA for solving multiobjective problems, here we describe the structure and performance of the original GA using the above explanations. The following is how the basic genetic algorithm works in 12 steps:

Step 1. Generate popsize individuals (chromosome) randomly as the initial population.

Step 2. Calculate the fitness of each individual

Step 3. Select a pair of parents from individuals based on the selection strategy.

Step 4. Apply crossover operator with the probability p_c on the parents and generate two new children.

Step 5. Apply mutation operator with the probability p_m on each child.

Step 6. Calculate the fitness of children

Step 7. Repeat steps 3–6 until generating popsize new children.

Step 8. Arrange all existing individuals, both old (i.e., the initial population) and new (i.e., generated children), that reach $2 \times$ popsize according to their fitness.

Step 9. Transfer popsize number of individuals with the best fitness to the new generation.

Step 10. Check the termination condition, if it is not met, go to step 3; otherwise, step 11.

Step 11. Select the best solution (the most fitted one) of the last generation as the answer to the algorithm.

Step 12. End.

Values related to the parameters of the GA including p_c, p_m, popsize, and ε will be determined by designing statistical tests in future sections.

3.9. Multiobjective Parallel Genetic Algorithm. There is a lot of research in the literature in which GA has been used to solve multiobjective problems due to its high ability to deal with multiobjective optimization. Since the GAs deal with a population of points (solutions), Pareto optimal multiple solutions can be found in a population of GAs [27]. In the following, we explain how to deal with a multiobjective problem.

In the mathematical model MOMILP, θ_1 and θ_2 are introduced according to the weight of each objective $C_{\max}$ and $\sum_{j=1}^{n} \text{Tard}_j$, and the objective function is written as a single function (1). Now, our biobjective problem has become a single-objective problem. However, determining weights is a delicate and sensitive task. To tackle this problem, we can divide the population into several groups, with individuals in each group searching for one of the possible combinations of goal coefficients, while all groups search in parallel for Pareto optimal solutions. In this way, we can turn a multiobjective problem into a single-objective problem with different weights. Therefore, we consider *a* set of λ different weights with a little difference between the two consecutive weights. Equation (20) shows the set of possible combinations.

$$\theta = \left\{\left(\theta_1^1, \theta_2^1\right), \left(\theta_1^2, \theta_2^2\right), \ldots, \left(\theta_1^\lambda, \theta_2^\lambda\right)\right\}. \tag{21}$$

Similarly, we divide the population into λ subpopulations and assign weights of the above set to them. Thus, each subpopulation searches for Pareto optimal solutions in a separate path. Subpopulations act independently and unrelatedly as parallel GAs, but their other parameters and operators, including crossover and mutation, the size of the subpopulation, and termination conditions, are all the same.

In this way, the scattering of solutions is effectively strengthened. Also, Pareto solutions are stored to keep dominant solutions. This prevents losing dominant solutions during the optimization process. This set is updated frequently to get closer to Pareto's optimal solution. The process of MOPGA is as follows:

Step 1. Generate the initial population randomly, including $\lambda \times$ popsize members

Step 2. Divide the initial population into λ subpopulations

Step 3. Create the weights related to the single objective function and assign to the subpopulations

$$\theta = \{(\sup \times \rho, 1 - \text{sub} \times \rho)\colon \text{sub} = 0, 1, \ldots, \lambda - 1\}. \tag{22}$$

For instance, if we set $\lambda = 21$ and $\rho = 0.05$, we will have:

$$\theta = \{(0.0, 1.0), (0.05, 0.95), \ldots, (1.0, 0.0)\}. \tag{23}$$

Step 4. Perform the initial genetic algorithm process from step 2 for all subpopulations and save the dominant solutions

Step 5. Find the best solution among the dominant ones

Step 6. Finish

3.10. The Simulated Annealing (SA)-Based Method. Another common metaheuristic algorithm for solving NP-hard problems is Simulated Annealing. This algorithm is inspired by physical systems in which the energy of atoms is reduced through the cooling process to a minimum level [30–32]. SA analyzes the complete surface of the goal function and attempts to optimize it while going both uphill and downhill. Consequently, it is essentially independent of the initial values, which are often a crucial input in traditional algorithms. In addition, it can escape from local optimums and locate the global optimum via uphill and downhill maneuvers. In addition, SA makes fewer rigorous assumptions about the objective function than traditional algorithms. Because of these lenient assumptions, it can cope with objective functions with ridges and plateaus more readily. Finally, it is capable of optimizing objective functions that are not even specified for certain parameter values [33].

In this section, we first provide an overview of the SA and then describe the MOPSA.

3.10.1. The Primary Simulated Annealing (SA) Algorithm. SA starts from a solution (energy level) as the basis of an initial temperature (temp 0) and searches in the solution space in the neighborhood of that solution. The search continues until reaching a termination condition, which we set as $m.n.\varphi$ iterations. Parameter φ is a fixed coefficient in range (0, 1). In this algorithm, we also use the method of GA to encode the problem solutions and introduce chromosomes. We also considered a random permutation of operations as the base solution for the beginning. In order to move from one solution to another, the neighborhood search method described in Algorithm 1 is used. After completing the neighborhood search process, the current temperature is reduced by multiplying it by the parameter $\mu \epsilon (0, 1)$ and the neighborhood search process is repeated.

The algorithm continues until reaching the final temperature (temp_f). During iterations and searches, the new solution is accepted if it improves the value of the objective function, otherwise, the new solution is accepted with the probability $\exp(-\Delta/\text{temp}_i)$, in which $temp_i$ is the current temperature and $\Delta = 100 \times (Z_{\text{new}} - Z_{\text{old}})/Z_{\text{new}}$. SA is summarized in Algorithm 2. We consider the final temperature as *1* and the value of other parameters of the algorithm including temp, μ and φ set by the statistical experiment design method is discussed in Section 4.

3.10.2. Multiobjective Parallel Simulated Annealing (SA). MOPSA process is very similar to the process of MOPGA and both use the same approach to deal with multiobjective optimization. Here, the objective function is considered as a single function (1) and several SA examines the different weights of each target in parallel and independently. The difference between starting MOPSA and MOPGA is that, in MOPGA, the initial population is divided into several subpopulations, but since SA starts with only one chromosome, such a division is not possible. The process of MOPSA is as follows:

Step 1. Produce λ chromosomes randomly

Step 2. Produce the weights of the single objective function as equation (22) and assign them to the chromosomes

Step 3. Perform the primary SA as shown in Algorithm 2 for all chromosomes from step 2 and store the dominant solutions

Step 4. Find the best solution among the dominant ones

Step 5. End

4. Experimental Design and Computation Evaluation

In this section, we seek to establish the adapted algorithms through experiments and statistical analysis. For this purpose, different levels of the relevant factors are considered

```
(1) Choose a gene at random.
(2) If the selected gene is located in the first location of the chromosome, then replace it with the next gene.
    Otherwise,
(3) If the selected gene is located at the last location of the chromosome replace it with the previous gene.
    Otherwise,
(6) Evaluate exchanging with the two lateral genes (previous and next).
(7) Replace the selected gene with the lateral gene that gets the most improvement.
(8) End of condition.
```

Algorithm 1: Neighborhood search simulated annealing algorithm.

```
(1)  Generate a random permutation of operations.
(2)  Calculate the value of the objective function.
(3)  Set the parameters temp, $\text{temp}_f, \mu, \varphi$.
(4)  Check the termination condition, then go to step 5 if not met and go to step 15 if met.
(5)  Set repeat = 0.
(6)  repeat < Int$(m.n.\varphi)$ go to step 7, otherwise, go to step 14.
(7)  Call the neighborhood search process.
(8)  If the value of the objective function is improved, go to step 9; otherwise go to step 10.
(9)  Accept the new solution, update the value of the objective function, and go to step 13.
(10) Select a random number in (0, 1) and put it in rand.
(11) If $\exp(-\Delta/\text{temp}_i) < rand$ go to step 12; otherwise, go to step 13.
(12) Accept the new solution, and update the value of the objective function.
(13) Set repeat = repeat + 1.
(14) Modify the new temperature as $\text{temp}_{\text{new}} = \mu.\text{temp}_{\text{old}}$ and go to step 4.
(15) End.
```

Algorithm 2: Steps of the primary simulated annealing algorithm.

and their different combinations are investigated. Therefore, the Taguchi approach is used as an experimental design method in this paper.

In addition, after adjusting the parameters of the proposed algorithms using the Taguchi method and by generating random data, two problem groups, one with small dimensions and the other with large dimensions, are randomly generated and the results of the solution methods are compared.

We also compare the results of our experiments on a large-scale benchmark with other methods that solved the OSSP.

4.1. Taguchi Experiment Design. Taguchi method is one of the statistical analysis methods in which using orthogonal arrays the number of experiments is significantly reduced compared to complete factorial designs [30]. For example, to design an experiment with four factors, each including three levels, a complete factorial design performs $3^4 = 81$ experiments, but Taguchi reduces them to nine experiments.

Taguchi believed that the key to improving quality was to reduce deviations, not just controlling placement within the specified range. Therefore, he tried to reduce the deviations to zero and paid less attention to being on a fixed slope. This is the main reason why the Taguchi method is mentioned as a reliable method [34]. According to the Taguchi method, factors are divided into two categories: controllable factors or signals and uncontrollable factors or noise. In order to reduce the deviations, a fraction called signal to noise (S/N) is defined which indicates the sensitivity of the response variable to uncontrollable factors and shows the deviation around a certain amount of signal. High values of S/N indicate low deviation, which means that controllable factors are more effective than uncontrollable ones. Using the optimal levels of factors obtained by this method, close to the mean value with the least deviation is expected [30].

To convert the objective function to the fraction S/N, three types of criteria can be imagined in the Taguchi method: "the less the better," "the more the better," and "the closer to the nominal value the better" [30]. According to the measurement criterion of the problem discussed in this, which is expressed as a minimization expression, the "less is better" mode is appropriate. The relevant S/N formula is as follows:

$$\frac{S}{N} = -10 \log_{10}\left(Z^2\right). \tag{24}$$

Also, a well-known performance criterion named RPD is used:

$$\text{RPD} = \frac{\text{Alg}_{\text{sol}} - \text{Min}_{\text{sol}}}{\text{Min}_{\text{sol}}}.100, \tag{25}$$

Where Alg_{sol} is the value of an objective function of an example which is obtained from an algorithm of experiment design and Min_{sol} is the minimum value of the objective function of that example which is obtained from all the algorithms of experiments. So, it is obvious that low RPD values are more valuable.

4.1.1. Data Generation. To perform Taguchi experiments, a problem with dimensions $(m, n, b) = (10, 5, 20)$ is considered, and five iterations of the same are generated randomly. Also, other literature-inspired [9, 35–37] parameters of the problem are as follows:

(i) Processing times have a continuous uniform distribution in (1, 99).

(ii) Transfer times between machines have a continuous uniform distribution in (1) and (20).

(iii) Consecutive unavailable durations of each machine have a uniform distribution in (1, 50).

(iv) Consecutive machine access times (packet size of each machine) are calculated as follows:

$$T_i = \max\left\{ a\sum_{j=1}^{n} p_{ij}, \max_i p_{ij} \right\}, \tag{26}$$

where a is selected from $\{1/5, 1/4, 1/3\}$ randomly.

Delivery time is calculated as follows:

$$d_j = \sum_{i=1}^{m} p_{ij} + \frac{\left(\sum_{i=1}^{m}\sum_{h=1}^{m} Tr_{ihj}\right)}{m} + U(0\ \ 1).(n-1).\frac{\sum_{i=1}^{m} t_i}{m}. \tag{27}$$

In which d_j is the total time that a job is on all machines. $\sum_{i=1}^{m} p_{ij}$ is the average movement between all machines. $\sum_{i=1}^{m}\sum_{h=1}^{m} Tr_{ihj}/m$ is the average unavailable times of all machines.

4.1.2. Adjusting the Parameters of the MOPGA. Factors to be set in the MOPGA algorithm are as follows: crossover probability (p_c), mutation probability (p_m), population size (popsize), and termination criterion coefficient (ε). According to Table 10, three levels are considered for each factor [29, 30]. So, Orthogonal Array Design L_9 is the most appropriate case because it has exactly the same number of factors and levels as the experiment and meets all objectives. Combinations of different levels of factors in each experiment of the design L_9 for the MOPGA algorithm are shown in Table 11. MOPGA and all experiment designs are programmed in C#.Net visual language and run on a computer with GHz Intel® Core™ and 4 GB ram. According to the results, optimal levels of factors are as follows: $p_c = 0.8$, $p_m = 0.2$, popsize = 30 and $\varepsilon = 0.4$.

In order to find the effect of each factor on the response variable, we perform an analysis of variance (ANOVA). It is important to note that in the designed experiment, the degree of error freedom is 0. To deal with this problem, the factors that have the lowest mean squares are considered errors. In the above experiment, factors p_c and popsize with mean squares of 0.92 and 0.70, respectively, are of the least mean squares and so have the least effect on the response variable. Therefore, these two factors are considered errors. Analysis of the variance of the S/N fraction is given in Table 12. Therefore, the most effective factors are p_m and ε with the percentage of effect of 86.92% and 7.81%, respectively.

Table 10: Factors of MOPGA algorithm and their levels.

Factor	Level		
	(1)	(2)	(3)
Pc	0.4	0.6	0.8
Pm	0.0	0.1	0.2
popsize	20	30	40
ϵ	0.2	0.3	0.4

Table 11: Experiments related to the array L_9 in the MOPGA algorithm.

Experiment	Factor			
	Pc	*Pm*	popsize	ϵ
1	0.4	0.0	20	0.2
2	0.4	0.1	40	0.3
3	0.4	0.2	30	0.4
4	0.6	0.0	40	0.4
5	0.6	0.1	30	0.2
6	0.6	0.2	20	0.3
7	0.8	0.0	30	0.3
8	0.8	0.1	20	0.4
9	0.8	0.2	40	0.2

4.1.3. Adjusting the Parameters of the MOPSA. According to the description of the SA, three factors should be adjusted: initial temperature (temp_0), temperature reduction coefficient (μ), and the coefficient of the number of iterations per temperature (φ). Considering two levels for each factor, orthogonal array L_4 is suitable and its factors and levels are depicted in Table 13. Also, the combinations of different levels of factors in each experiment of design L_4 of the MOPSA are shown in Table 14. MOPSA and all experiment designs are programmed in C#.Net and run on a computer with GHz Intel® Core™ and 4 GB ram. Optimal levels of factors are as follows: $\text{temp}_0 = 100$, $\mu = 0.7$, and $\varphi = 0.3$.

An analysis of variance is performed to find the effect of each factor. Also, to prevent the error degree of freedom from zeroing, we consider the factor with the lowest mean squares as the error. The results indicate that the factor μ with the mean square 0.03 has the lowest effect on the response variable. Analysis of variance of S/N for the factors of the MOPSA algorithm is shown in Table 15 in which factor φ with the percentage of effect of 66.62% has the highest effect on the response variable. As well, temp_0 is the second effective factor with a percentage of effect of 21.89.

4.2. Computational Evaluation. In order to evaluate the performance of the MOMILP model and the proposed algorithms, two sets of problems including small-scale

Table 12: Analysis of variance of the S/N fraction for the factors of the MOPGA algorithm.

Factor	df	SS	MS	F	Percentage of effect (%)	Cumulative percentage (%)	p-value
popsize	2	108.56	54.28	67.07	86.93	86.93	0.00
ε	2	11.23	5.61	6.94	7.81	94.74	0.05
Error (p_c + popsize	4	3.24	0.81		5.26	100	
Total	8	123.02					

Table 13: Factors of MOPSA algorithm and their levels.

Factor	Level	
	(1)	(2)
temp_0	50	100
μ	0.7	0.8
φ	0.3	0.6

Table 14: Array experiments L_4 in MOPSA algorithm.

Experiment	Factor		
	temp_0	μ	ϕ
1	50	0.7	0.3
2	50	0.8	0.6
3	100	0.7	0.6
4	100	0.8	0.3

Table 15: Analysis of variance of the S/N fraction for the factors of the MOPSA algorithm.

Factor	df	SS	MS	F	Percentage of effect (%)	Cumulative percentage (%)	p-value
temp_0	1	0.17	0.17	6.72	21.89	21.89	0.23
ϕ	1	0.47	0.47	18.40	66.62	88.51	0.15
Error (μ)	1	0.03	0.03		11.49	100	
Total	3	0.66					

Table 16: Performance of MOMILP model and primary GA and SA algorithms in dealing with small-scale problems ($\theta_1 = \theta_2 = 0.5$.).

Problem		Z			Deviation from optimum (%)			Solving time (s)		
	m, n, b	MOMILP	GA	SA	MOMILP	GA (%)	SA (%)	MOMILP	GA	SA
1	2, 4, 8	449.74	449.74	510.53	0	0.00	13.52	11	84	1
2	3, 4, 8	347.31	350.83	668.90	0	1.02	92.60	60	105	1
3	2, 5, 10	492.95	492.95	726.71	0	0.00	47.42	145	106	1
4	3, 5, 10	382.65	466.38	737.00	0	21.88	92.61	1951	147	2
5	2, 6, 12	848.09	859.43	1415.23	0	1.34	66.87	1386	105	3
6	3, 6, 12	—	1026.97	2375.36	—	—	—	38744	168	4
Average					0	4.85	62.60			

Table 17: Average RPD for algorithms MOPGA and MOPSA in solving large-scale problems.

Problem	$m.n.b$	MOPGA	MOPSA	Problem	$m.n.b$	MOPGA	MOPSA
7	5, 10, 20	0.03	0.69	17	5, 20, 40	0.05	0.39
8	5, 10, 20	0.03	0.62	18	5, 20, 40	0.01	0.47
9	5, 10, 20	0.07	1.10	19	5, 20, 40	0.05	0.48
10	5, 10, 20	0.02	0.76	20	5, 20, 40	0.04	0.65
11	5, 10, 20	0.04	0.80	21	5, 20, 40	0.04	0.54
12	10, 10, 20	0.08	0.73	22	10, 20, 40	0.01	0.20
13	10, 10, 20	0.11	0.80	23	10, 20, 40	0.03	0.24
14	10, 10, 20	0.07	0.73	24	10, 20, 40	0.04	0.28
15	10, 10, 20	0.06	0.80	25	10, 20, 40	0.07	0.45
16	10, 10, 20	0.08	0.61	26	10, 20, 40	0.06	0.41
Average RPD of MOPGA: 0.05							
Average RPD of MOPSA: 0.59							

Table 18: Simulation results of the proposed algorithm and comparable algorithms for the Taillard [38] benchmarks.

Benchmarks	Optimal solution	SA	GA	Hybrid GA	Hybrid GA	GA	EGA_OS	ACS	CS	CSO	BA_OS	MOPSA	MOPGA
$4 \times 4 - 1$	193	193	193	213	193	193	193	193	193	193	193	193	193
$4 \times 4 - 2$	236	236	236	240	236	239	239	236	236	236	236	236	236
$4 \times 4 - 3$	271	271	271	293	271	271	271	271	271	271	271	271	271
$4 \times 4–4$	250	250	250	253	250	250	250	250	252	250	250	250	250
$4 \times 4 - 5$	295	295	295	303	295	295	295	295	295	295	295	295	295
$4 \times 4 - 6$	189	189	189	209	189	189	189	189	189	189	189	189	189
$4 \times 4 - 7$	201	201	201	203	201	201	201	201	201	201	201	201	201
$4 \times 4 - 8$	217	217	217	224	217	217	217	217	217	217	217	217	217
$4 \times 4 - 9$	261	261	261	281	261	261	261	261	261	261	261	261	261
$4 \times 4 - 10$	217	217	217	230	217	217	217	217	217	217	217	217	217
$5 \times 5 - 1$	300	300	301	323	300	301	300	301	301	300	300	300	300
$5 \times 5 - 2$	262	262	262	269	262	263	262	262	262	262	262	262	262
$5 \times 5 - 3$	323	323	331	353	323	335	323	331	335	323	323	323	323
$5 \times 5 - 4$	310	310	N/A	N/A	310	316	310	315	314	310	310	310	310
$5 \times 5 - 5$	326	326	N/A	N/A	326	330	326	331	329	326	326	326	326
$5 \times 5 - 6$	312	312	312	327	312	312	312	317	318	312	312	312	312
$5 \times 5 - 7$	303	303	N/A	N/A	303	308	303	308	305	303	303	303	303
$5 \times 5 - 8$	300	300	N/A	N/A	300	304	300	304	303	300	300	300	300
$5 \times 5 - 9$	353	353	353	373	353	358	353	358	358	353	353	353	353
$5 \times 5 - 10$	326	326	326	341	326	328	326	329	329	326	326	326	326
$7 \times 7 - 1$	435	435	438	447	435	436	435	435	436	435	435	435	435
$7 \times 7 - 2$	443	443	455	454	443	447	443	445	447	443	443	443	443
$7 \times 7 - 3$	468	468	N/A	N/A	468	472	468	479	472	468	468	468	468
$7 \times 7 - 4$	463	463	N/A	N/A	463	463	463	467	466	463	463	463	463
$7 \times 7 - 5$	416	416	N/A	N/A	416	417	416	419	416	416	416	416	416
$7 \times 7 - 6$	451	451	N/A	N/A	451	455	451	460	454	452	451	452	451
$7 \times 7 - 7$	422	422	443	450	422	426	422	435	425	422	422	422	422
$7 \times 7 - 8$	424	424	N/A	N/A	424	424	424	424	424	426	424	424	424
$7 \times 7 - 9$	458	458	465	467	458	458	458	458	458	458	458	459	458
$7 \times 7 - 10$	398	398	405	406	398	398	398	398	399	398	398	399	398
$10 \times 10 - 1$	637	637	667	655	637	637	637	638	639	645	637	641	637
$10 \times 10 - 2$	588	588	N/A	N/A	588	588	588	588	688	588	588	588	588
$10 \times 10 - 3$	598	598	N/A	N/A	598	598	598	599	600	599	598	598	598
$10 \times 10 - 4$	577	577	586	581	577	577	577	577	577	577	577	577	577
$10 \times 10 - 5$	640	640	N/A	N/A	640	640	640	640	640	640	640	640	640
$10 \times 10 - 6$	538	538	555	541	538	538	538	538	538	538	538	538	538
$10 \times 10 - 7$	616	616	N/A	N/A	616	616	616	616	616	616	616	616	616
$10 \times 10 - 8$	595	595	N/A	N/A	595	595	595	595	595	595	595	595	595
$10 \times 10 - 9$	595	595	627	598	595	595	595	595	595	595	595	595	595
$10 \times 10 - 10$	596	596	623	605	596	596	596	596	596	596	596	596	596
$15 \times 15 - 1$	937	937	967	937	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	918	919	918
$15 \times 15 - 3$	871	871	904	871	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	934
$15 \times 15 - 5$	946	946	N/A	N/A	946	946	946	946	946	946	946	948	946
$15 \times 15 - 6$	933	933	N/A	N/A	933	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	891
$15 \times 15 - 8$	893	893	928	893	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	899	913	899
$15 \times 15 - 10$	902	902	N/A	N/A	902	902	902	902	902	902	902	902	902
$20 \times 20 - 1$	1155	1155	1230	1165	1155	1155	1155	1155	1155	1155	1155	1166	1155
$20 \times 20 - 2$	1241	1241	N/A	N/A	1241	1241	1242	1241	1242	1243	1241	1260	1241
$20 \times 20 - 3$	1257	1282	1292	1257	1257	1257	1257	1257	1257	1257	1257	1257	1259
$20 \times 20 - 4$	1248	1274	N/A	N/A	1248	1248	1248	1248	1248	1248	1248	1253	1248
$20 \times 20 - 5$	1256	1289	1315	1256	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	1204
$20 \times 20 - 7$	1294	1294	N/A	N/A	1294	1294	1294	1294	1295	1294	1294	1310	1294
$20 \times 20 - 8$	1169	1169	N/A	N/A	1173	1171	1173	1170	1176	1175	1170	1173	1176
$20 \times 20 - 9$	1289	1307	1339	1289	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	1241

Table 19: Execution time of the proposed algorithm and compared algorithms in terms of seconds.

Benchmarks	CSO	MOPSA	MOPGA	Benchmarks	CSO	MOPSA	MOPGA
$4\times4-1$	0.001	0.001	0.003	$10\times10-1$	160.762	158.514	160.216
$4\times4-2$	0.001	0.001	0.007	$10\times10-2$	474.782	472.948	474.309
$4\times4-3$	0.001	0.001	0.018	$10\times10-3$	68.567	65.781	66.118
$4\times4-4$	0.156	0.101	0.148	$10\times10-4$	41.04	40.014	40.783
$4\times4-5$	0.125	0.097	0.129	$10\times10-5$	679.018	678.091	678.615
$4\times4-6$	0.003	0.001	0.009	$10\times10-6$	126.936	122.726	123.552
$4\times4-7$	0.109	0.074	0.101	$10\times10-7$	22.877	20.281	21.227
$4\times4-8$	0.359	0.231	0.287	$10\times10-8$	375.292	371.557	374.591
$4\times4-9$	0.219	0.142	0.186	$10\times10-9$	263.801	257.890	262.175
$4\times4-10$	0.156	0.099	0.121	$10\times10-10$	112.65	110.032	11.758
$5\times5-1$	0.610	0.451	0.573	$15\times15-1$	250.323	247.080	248.903
$5\times5-2$	0.735	0.618	0.691	$15\times15-2$	501.669	494.118	486.337
$5\times5-3$	2.235	1.876	2.116	$15\times15-3$	498.766	492.016	493.768
$5\times5-4$	8.906	6.569	8.592	$15\times15-4$	112.994	109.854	11.243
$5\times5-5$	1.344	1.123	1.338	$15\times15-5$	834.63	827.603	831.728
$5\times5-6$	13.813	11.097	12.589	$15\times15-6$	149.578	147.932	148.005
$5\times5-7$	14.688	13.458	13.976	$15\times15-7$	867.539	863.590	564.329
$5\times5-8$	35.236	31.231	34.811	$15\times15-8$	176.073	175.327	175.824
$5\times5-9$	4.032	2.642	3.628	$15\times15-9$	658.615	652.187	655.442
$5\times5-10$	2.453	2.321	2.404	$15\times15-10$	891.272	887.849	890.119
$7\times7-1$	24.375	21.968	23.211	$20\times20-1$	457.839	453.202	455.601
$7\times7-2$	26.065	24.376	24.897	$20\times20-2$	618.409	607.114	610.105
$7\times7-3$	51.658	47.612	50.626	$20\times20-3$	340.266	337.621	338.290
$7\times7-$	227.313	221.874	224.529	$20\times20-4$	816.467	811.479	814.471
$7\times7-5$	169.751	167.496	168.004	$20\times20-5$	432.391	426.719	431.772
$7\times7-6$	640.466	633.012	137.831	$20\times20-6$	806.064	805.431	805.712
$7\times7-7$	349.208	347.890	348.210	$20\times20-7$	859.499	857.386	859.116
$7\times7-8$	21.567	20.001	20.901	$20\times20-8$	695.772	694.329	694.707
$7\times7-9$	83.953	78.016	81.378	$20\times20-9$	207.641	203.989	206.752
$7\times7-10$	112.39	110.731	112.004	$20\times20-10$	489.458	484.307	488.262

problems ($n=4, 5, 6$; $m=2, 3$) and large-scale problems ($n=10, 20$; $m=5, 10$) have been generated [26]. Considering one iteration for each of the small-scale problems and five iterations for each of the large-scale problems, we will have a total of 26 problems. The rest of the parameters are considered exactly as in the Data Generation Section. Considering the weights as $\theta_1 = \theta_2 = 0.5$, small-scale problems are solved using the primary GA and SA and the results will be compared to the optimal solutions obtained by the MOMILP model with the mentioned weights. MOMILP model is coded in Lingo 8 and C#.Net is used to code the primary algorithms GA and SA.

The performance of the MOMILP model and primary SA and GA are shown in Table 16. As seen in the table, although the MOMILP model is a suitable method to solve small-scale problems, by increasing the computational time of the MOMILP model while magnifying the problem, the GA method becomes more efficient with a deviation of 4.85% from the optimum. As well, although the SA algorithm shows a relatively large deviation from the optimal solution, its time saving compared to GA is clear.

To solve large-scale problems, MOPGA and MOPSA algorithms are used and each problem is solved five times using each of the algorithms. In addition, in order to compare algorithms, obtained results are converted to relative percentage difference (RPD) criteria and are shown in Table 17. Therefore, MOPGA with an average RPD of 0.05% is more efficient and stable than MOPSA in solving large-scale problems.

In addition, MOPGA and MOPSA are programmed in C#.Net programming language, and all algorithms including models MOMILP, GA, SA, MOPGA, and MOPSA are performed on a computer with CPU 3 GHz Intel® Core™ I and 4 GB RAM.

In order to show the efficiency of the proposed algorithm (MOPGA), we tested the above algorithm on benchmarks by Taillard [38] and compared it with other metaheuristic algorithms including SA [39], GA [40], Hybrid GA [40], GA [41], Hybrid GA [42], Ant Colony System (ACS) [43], Cuckoo Search (CS) [43], Cat Swarm Optimization (CSO) algorithm [44], Extended Genetic Algorithm Open Shop (EGA_OS) [14], and Bat Algorithm Open Shop (BA_OS) [22]. Table 18 shows the simulation results of the proposed algorithm and the comparable algorithms for different benchmarks. As mentioned in this table, the proposed algorithms were able to achieve the optimal solution in most benchmarks and solve the problem well.

As you can see in Table 19, the MOPSA algorithm has less execution time than the other two algorithms and was able to respond in less time. The MOPGA algorithm was able to achieve the optimal answer in less time than the CSO algorithm.

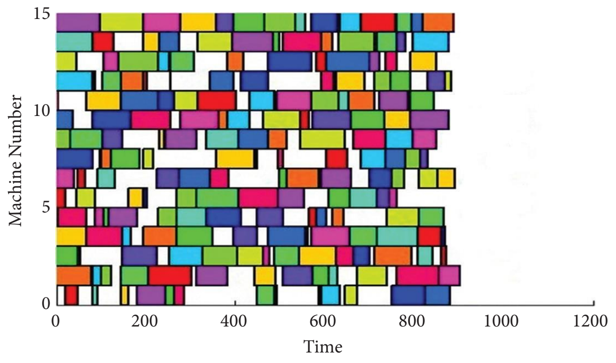

Figure 3: Gantt chart of the MOPGA for the 15 × 15 − 1 benchmark.

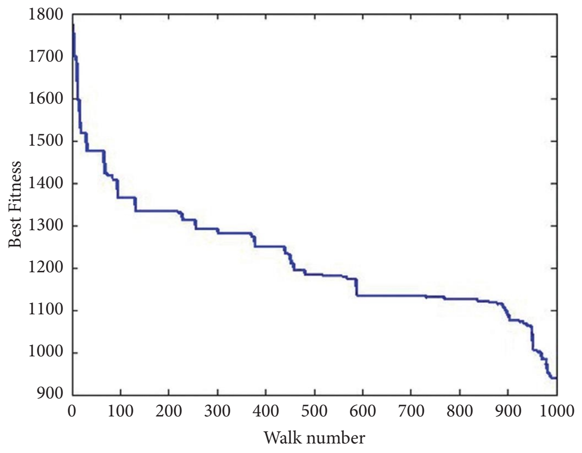

Figure 4: Fitness diagram of the MOPGA for the 15 × 15 − 1 benchmark.

The accuracy of the proposed algorithm in programming things well on machines is shown in Figure 3. This figure shows the Gantt chart for the 15 × 15 − 1 benchmark. In this case, a few free times left for the machine shows the accuracy of the proposed algorithm.

Figure 4 shows the fitness diagram of the MOPGA for the 15 × 15 − 1 benchmark. In this diagram, the best fitness is getting close to 900 while the walk number crosses 900 walks.

Figure 5 shows the dispersion diagram for the 15 × 15 − 1 benchmark. Population dispersion is always maintained in the MOPGA. As you can see in the above algorithm, premature convergence of chromosomes is prevented.

5. Conclusion

In this paper, a multiobjective mixed-integer linear programming model is adapted to solve biobjective OSSP considering unavailable times for machines. Also, "maximum completion time" and "total delays" are considered in optimization criteria simultaneously. The efficiency of the MOMILP model is investigated through a number of small-scale problems and compared with the efficiency and execution time of GA and SA. Although the results indicate the high ability of the MOMILP model to solve small-scale problems, the analyzes performed on the model highlight the need to develop other methods such as heuristic and metaheuristic methods to solve large-scale problems. Therefore, MOPGA and MOPSA algorithms were introduced for multiobjective problems with unknown weights for objectives. Also, the reliability of multiobjective algorithms is evaluated by designing Taguchi experiments. Results show that the MOPGA algorithm is more efficient than the MOPSA algorithm in dealing with large-scale problems. Finally, variations in MOPGA and MOPSA were compared in detail with various heuristic algorithms selected from the literature according to the Average RPD, execution time, and best solution criteria. Our experiments proved the success of the proposed solutions by running MOPGA and MOPSA not only on low- and medium-scaled open shop scheduling samples but also on large-scale open shop scheduling samples.

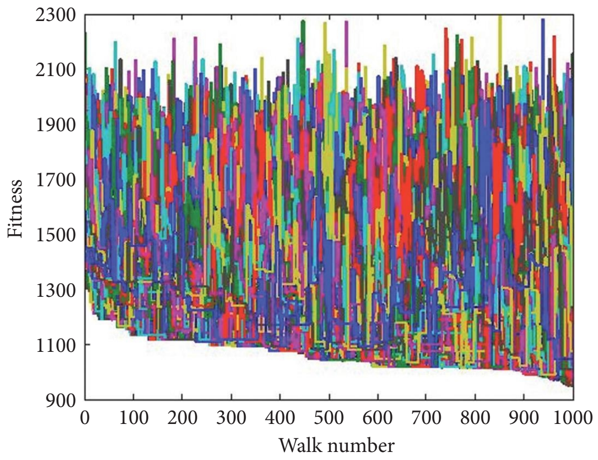

Figure 5: Dispersion diagram of the MOPGA for the 15 × 15 − 1 benchmark.

In the following, some areas of development and expanding the present study are mentioned:

(i) Developing other solution methods, including new heuristic and metaheuristic methods to solve large-scale problems and comparison with the algorithms proposed in this paper can be a good idea for making the model more practical.

(ii) Considering more diverse optimization criteria according to the needs of today's industry can also provide the basis for future studies.

(iii) In this paper, optimizing factors of the algorithms in discrete space are explored. Therefore, another area of development of the present paper can be the optimization of factors in the continuous space.

(iv) A sequence-dependent preparation and separation times can be considered and then a mathematical model can be developed.

(v) The number of vehicles can be used as a constraint on the problem.

(vi) Considering variable speeds for machines seems practical. That is, the processing speed of the machines decreases after performing a certain number of jobs or a certain period of time and then increases again after performing the maintenance process.

Data Availability

We used data from ref [26] and discussed the same in Section 4.2 of the manuscript.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

References

[1] D. Biskup, "Single-machine scheduling with learning considerations," *European Journal of Operational Research*, vol. 115, no. 1, pp. 173–178, 1999.

[2] A. S. Manne, "On the job-shop scheduling problem," *Operations Research*, vol. 8, no. 2, pp. 219–223, 1960.

[3] M. Sheikhalishahi, N. Eskandari, A. Mashayekhi, and A. Azadeh, "Multi-objective open shop scheduling by considering human error and preventive maintenance," *Applied Mathematical Modelling*, vol. 67, pp. 573–587, 2019.

[4] F. Zhao, R. Ma, and L. Wang, "A self-learning discrete jaya algorithm for multiobjective energy-efficient distributed no-idle flow-shop scheduling problem in heterogeneous factory system," *IEEE Transactions on Cybernetics*, pp. 1–12, 2021.

[5] B. Zhang, Qk. Pan, Ll. Meng, C. Lu, Jh Mou, and Jq Li, "Pan Q-k, Meng L-l, Lu C, Mou J-h, Li J-q: an automatic multi-objective evolutionary algorithm for the hybrid flowshop scheduling problem with consistent sublots," *Knowledge-Based Systems*, vol. 238, p. 107819, 2022.

[6] K. Gao, Z. Cao, L. Zhang, Z. Chen, Y. Han, and Q. Pan, "A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems," *IEEE/CAA Journal of Automatica Sinica*, vol. 6, no. 4, pp. 904–916, 2019.

[7] Y. Fu, Y. Hou, Z. Wang, X. Wu, K. Gao, and L. Wang, "Distributed scheduling problems in intelligent manufacturing systems," *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 625–645, 2021.

[8] J. Breit, G. Schmidt, and V. A. Strusevich, "Two-machine open shop scheduling with an availability constraint," *Operations Research Letters*, vol. 29, no. 2, pp. 65–77, 2001.

[9] C.-J. Hsu, C. Low, and C.-T. Su, "A single-machine scheduling problem with maintenance activities to minimize makespan," *Applied Mathematics and Computation*, vol. 215, no. 11, pp. 3929–3935, 2010.

[10] Y. Fu, Y. Hou, Z. Chen, X. Pu, K. Gao, and A. Sadollah, "Modelling and scheduling integration of distributed production and distribution problems via black widow optimization," *Swarm and Evolutionary Computation*, vol. 68, p. 101015, 2022.

[11] V. A. Strusevich, "A heuristic for the two-machine open-shop scheduling problem with transportation times," *Discrete Applied Mathematics*, vol. 93, no. 2-3, pp. 287–304, 1999.

[12] Y. Fu, H. Li, M. Huang, and H. Xiao, "Bi-objective modeling and optimization for stochastic two-stage open shop scheduling problems in the sharing economy," *IEEE Transactions on Engineering Management*, pp. 1–15, 2022.

[13] A. A. R. Hosseinabadi, H. Siar, S. Shamshirband, M. Shojafar, and M. H. N. M. Nasir, "Using the gravitational emulation local search algorithm to solve the multi-objective flexible dynamic job shop scheduling problem in Small and Medium Enterprises," *Annals of Operations Research*, vol. 229, no. 1, pp. 451–474, 2015.

[14] C.-F. Liaw, "Scheduling preemptive open shops to minimize total tardiness," *European Journal of Operational Research*, vol. 162, no. 1, pp. 173–183, 2005.

[15] A. Sedeño-Noda, D. Alcaide, and C. González-Martín, "Network flow approaches to pre-emptive open-shop scheduling problems with time-windows," *European Journal of Operational Research*, vol. 174, no. 3, pp. 1501–1518, 2006.

[16] H. Panahi, M. Rabbani, and R. Tavakkoli-Moghaddam, "Solving an open shop scheduling problem by a novel hybrid multi-objective ant colony optimization," in *2008 Eighth International Conference on Hybrid Intelligent Systems: 2008*, pp. 320–325, IEEE, 2008.

[17] H. Panahi and R. Tavakkoli-Moghaddam, "Solving a multi-objective open shop scheduling problem by a novel hybrid ant colony optimization," *Expert Systems with Applications*, vol. 38, no. 3, pp. 2817–2822, 2011.

[18] Z. Drezner, "A distance based rule for removing population members in genetic algorithms," *4OR*, vol. 3, no. 2, pp. 109–116, 2005.

[19] P. Pirozmand, A. A. R. Hosseinabadi, M. Farrokhzad, M. Sadeghilalimi, S. Mirkamali, and A. Slowik, "Multi-objective hybrid genetic algorithm for task scheduling problem in cloud computing," *Neural Computing & Applications*, vol. 33, no. 19, pp. 13075–13088, 2021.

[20] F. Zhao, X. He, and L. Wang, "A two-stage cooperative evolutionary algorithm with problem-specific knowledge for energy-efficient scheduling of no-wait flow-shop problem," *IEEE Transactions on Cybernetics*, vol. 51, no. 11, pp. 5291–5303, 2021.

[21] F. Zhao, L. Zhao, L. Wang, and H. Song, "An ensemble discrete differential evolution for the distributed blocking flowshop scheduling with minimizing makespan criterion," *Expert Systems with Applications*, vol. 160, p. 113678, 2020.

[22] M. B. Shareh, S. H. Bargh, A. A. R. Hosseinabadi, and A. Slowik, "An improved bat optimization algorithm to solve the tasks scheduling problem in open shop," *Neural Computing & Applications*, vol. 33, no. 5, pp. 1559–1573, 2021.

[23] S. Karimi, Z. Ardalan, B. Naderi, and M. Mohammadi, "Scheduling flexible job-shops with transportation times: mathematical models and a hybrid imperialist competitive algorithm," *Applied Mathematical Modelling*, vol. 41, pp. 667–682, 2017.

[24] D. Bai and L. Tang, "Open shop scheduling problem to minimize makespan with release dates," *Applied Mathematical Modelling*, vol. 37, no. 4, pp. 2008–2015, 2013.

[25] M. Ahmady, S. S. Mirkamali, B. Pahlevanzadeh, E. Pashaei, A. A. R. Hosseinabadi, and A. Slowik, "Facial expression recognition using fuzzified Pseudo Zernike Moments and structural features," *Fuzzy Sets and Systems*, vol. 443, pp. 155–172, 2022.

[26] J. Khodadoust, A. M. Khodadoust, S. S. Mirkamali, and S. Ayat, "Fingerprint indexing for wrinkled fingertips immersed in liquids," *Expert Systems with Applications*, vol. 146, p. 113153, 2020.

[27] C.-F. Liaw, "A hybrid genetic algorithm for the open shop scheduling problem," *European Journal of Operational Research*, vol. 124, no. 1, pp. 28–42, 2000.

[28] M. E. Matta, "A genetic algorithm for the proportionate multiprocessor open shop," *Computers & Operations Research*, vol. 36, no. 9, pp. 2601–2618, 2009.

[29] B. Naderi, S. Fatemi Ghomi, M. Aminnayeri, and M. Zandieh, "Scheduling open shops with parallel machines to minimize total completion time," *Journal of Computational and Applied Mathematics*, vol. 235, no. 5, pp. 1275–1287, 2011.

[30] R. Tavakkoli-Moghaddam, N. Safaei, and F. Sassani, "A memetic algorithm for the flexible flow line scheduling problem with processor blocking," *Computers & Operations Research*, vol. 36, no. 2, pp. 402–414, 2009.

[31] B.-W. Cheng and C.-L. Chang, "A study on flowshop scheduling problem combining Taguchi experimental design and genetic algorithm," *Expert Systems with Applications*, vol. 32, no. 2, pp. 415–421, 2007.

[32] R. Tavakkoli-Moghaddam, A. Rahimi-Vahed, A. Ghodratnama, and A. Siadat, "A simulated annealing method for solving a new mathematical model of a multi-criteria cell formation problem with capital constraints," *Advances in Engineering Software*, vol. 40, no. 4, pp. 268–273, 2009.

[33] A. Corana, M. Marchesi, C. Martini, and S. Ridella, "Minimizing multimodal functions of continuous variables with the "simulated annealing" algorithm—corrigenda for this article is available here," *ACM Transactions on Mathematical Software*, vol. 13, no. 3, pp. 262–280, 1987.

[34] B. Naderi, M. Zandieh, and S. M. T. Fatemi Ghomi, "Fatemi Ghomi S: scheduling sequence-dependent setup time job shops with preventive maintenance," *International Journal of Advanced Manufacturing Technology*, vol. 43, no. 1-2, pp. 170–181, 2009.

[35] G. J. Kyparisis and C. Koulamas, "Open shop scheduling with makespan and total completion time criteria," *Computers & Operations Research*, vol. 27, no. 1, pp. 15–27, 2000.

[36] R. Ruiz, J. Carlos García-Díaz, and C. Maroto, "Considering scheduling and preventive maintenance in the flowshop sequencing problem," *Computers & Operations Research*, vol. 34, no. 11, pp. 3314–3330, 2007.

[37] J. Jungwattanakit, M. Reodecha, P. Chaovalitwongse, and F. Werner, "Algorithms for flexible flow shop problems with unrelated parallel machines, setup times, and dual criteria," *International Journal of Advanced Manufacturing Technology*, vol. 37, no. 3-4, pp. 354–370, 2008.

[38] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operational Research*, vol. 64, no. 2, pp. 278–285, 1993.

[39] H. M. Harmanani and S. B. Ghosn, "An efficient method for the open-shop scheduling problem using simulated annealing. In: *Information technology: New generations.* edn.: Springer," pp. 1183–1193, 2016.

[40] S. Khuri and S. R. Miryala, "Genetic algorithms for solving open shop scheduling problems. In: Portuguese Conference on Artificial Intelligence: 1999: Springer," pp. 357–368, 1999.

[41] R. H-LFaP, "Corne D: a promising hybrid GA/heuristic approach for open-shop scheduling problems," *In: Proceedings of the 11th European conference on artificial intelligence: 1994: Citeseer*, pp. 590–594, 1994.

[42] C. Prins, "Competitive genetic algorithms for the open-shop scheduling problem," *Mathematical Methods of Operations Research*, vol. 52, no. 3, pp. 389–411, 2000.

[43] W. Marrouche and H. M. Harmanani, "Heuristic approaches for the open-shop scheduling problem. In: *information Technology-New Generations.* edn.: Springer," pp. 691–699, 2018.

[44] A. Bouzidi, M. E. Riffi, and M. Barkatou, "Cat swarm optimization for solving the open shop scheduling problem," *Journal of Industrial Engineering International*, vol. 15, no. 2, pp. 367–378, 2019.