Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2015, Article ID 612604, 10 pages
http://dx.doi.org/10.1155/2015/612604

Research Article

A Bee Colony Optimization Approach for Mixed Blocking Constraints Flow Shop Scheduling Problems

Mostafa Khorramizadeh[1] and Vahid Riahi[2]

[1]*Department of Mathematical Sciences, Shiraz University of Technology, Shiraz 71555-313, Iran*
[2]*Department of Industrial Engineering, Shiraz University of Technology, Shiraz 71555-313, Iran*

Correspondence should be addressed to Mostafa Khorramizadeh; m.khorrami@sutech.ac.ir

Received 11 June 2015; Revised 21 September 2015; Accepted 20 October 2015

Academic Editor: Marco Mussetta

The flow shop scheduling problems with mixed blocking constraints with minimization of makespan are investigated. The Taguchi orthogonal arrays and path relinking along with some efficient local search methods are used to develop a metaheuristic algorithm based on bee colony optimization. In order to compare the performance of the proposed algorithm, two well-known test problems are considered. Computational results show that the presented algorithm has comparative performance with well-known algorithms of the literature, especially for the large sized problems.

1. Introduction

Flow shop scheduling problem has been extensively studied for over 50 years because of its significance in both theory and industrial applications [1]. As an important branch of flow shop scheduling problems, the blocking flow shop scheduling has attracted much attention in recent years. In the blocking flow shop, because of the lack of intermediate buffer storage between consecutive machines, a job has to stay in the current machine until the next machine is available for processing, even if it has completed processing in this machine [2]. This increases the waiting time and thus influences the efficiency of production. In the classical flow shop problem, the buffer space capacity between machines is considered unlimited. Some flow shop problems are concerned with the blocking constraints such as RSb, RCb, and RCb* (these blocking constraints will be described in Section 2). For problems with classical blocking constraint (RSb), Wang et al. [3, 4] developed a hybrid genetic algorithm for flow shop scheduling with limited buffers and a hybrid harmony search algorithm and Ribas et al. [5] proposed an iterated greedy algorithm. RCb constraint was introduced for the first time by Dauzere-Peres [6]. Regarding RCb constraint, an integer linear programming model, lower bounds, and a metaheuristic are presented in [7]. These problems have also been solved in [8] by a metaheuristic algorithm. A new blocking constraint called RCb* has been proposed by Trabelsi et al. [9] which is an RCb constraint simultaneously subjected to different types of blocking constraints on successive machines in a process. In [10], authors proposed some heuristic for the flow shop problem with RSb, RCb, and RCb* constraints and solved it with genetic algorithm. In [11], the complexity of a production system where the RSb and RCb constraints are mixed is studied.

In this paper, we propose a constructive heuristic for minimizing the makespan in flow shop scheduling problems with mixed blocking constraints. We propose a new heuristic method for the problem, which is based on the ideas of the bee colony optimization. The presented algorithm is developed by using the ideas of the Taguchi orthogonal arrays and the path relinking method. Moreover, some heuristic local search methods and perturbation procedures are also designed to improve the efficiency of the resulting algorithm. To compare the presented algorithm with some efficient algorithms of

the literature, two sets of well-known test problems are used. The numerical results show that the presented algorithm is comparative with well-known algorithms of the literature.

This paper is structured as follows: the mixed blocking flow shop problem is described in Section 2. Section 3 is concerned with the description of the presented algorithm of this paper. The numerical results are given in Section 4. Finally, the paper is concluded in Section 5.

2. Problem Description

To describe the problem, we followed the same approach as [10]. Flow shop scheduling problems are a class of scheduling problems with a workshop or group shop in which the flow control will enable an appropriate sequencing for each job and for processing on a set of machines or with other resources $1, 2, \ldots, m$ in compliance with given processing orders. Particularly, the maintaining of a continuous flow of processing tasks is desired with a minimum of idle time and a minimum of waiting time. Flow shop scheduling is a special case of job shop scheduling where there is strict order of all operations to be performed on all jobs. Flow shop scheduling may apply as well to production facilities as to computing designs. The deterministic flow shop scheduling problem consists of a finite set J of n jobs to be processed on a finite set M of m machines. Each job J_i must be processed on every machine in its routing consisting of m operations $O_{i1}, O_{i2}, \ldots, O_{im}$. Operation O_{ij} needs an execution time P_{ij} on machine $Mj \in M$, performed in order. For $1 \leq j \leq M$, only one job can be executed on machine M_j, at any time. Here, preemptive operations are not authorized. Objective function is to reduce the time when all operations are completed, that is, makespan. Several different cases of flow shop can be considered such as the classical flow shop problem without any blocking constraint, the flow shop problem with only one blocking constraint between all machines, and the flow shop problems in which different blocking constraints are mixed. One constraint met in industrial problems is that a job blocks a machine until this job starts on next machine in routing. This classical blocking constraint is denoted by RSb (Release when Starting Blocking). Another constraint is that a machine will be immediately available to treat its next operation after its job on the following machine in process is finished without regard to whether or not it leaves the machine. This blocking constraint is denoted by RCb* (Release when Completing Blocking*). In a large production line, different types of blocking constraints can be encountered which depend on intermediate storage between machines, characteristics of machines, and technical constraints. To characterize a flow shop problem where different blocking constraints are mixed, a vector V is introduced that contains blocking constraints between machines. Element V_j is blocking constraint between machines M_j and M_{j+1}. This vector has $m - 1$ elements (as many elements as the number of transitions between machines). See [10] for details and a mathematical programming formulation of the problem. For the flow shop scheduling problem, we use a permutation p of jobs as a solution representation. For example, suppose there are six jobs and four machines in a flow shop scheduling problem. A permutation $p = (2, 3, 1, 6, 5, 4)$ is a permutation of six jobs and this solution represents a scheduling in which the sequence of jobs on each machine is J_2, J_3, J_1, J_6, J_5, J_4.

3. Bee Colony Algorithm

To describe the bee colony algorithm, we use the same approach as [12]. The bee colony algorithm is a stochastic P-metaheuristic that belongs to the class of swarm intelligence algorithms. BC is the one, which has been most widely studied and applied to solve the real-world problems. The basic artificial bee colony algorithm [12] classifies bees into three categories: employed bees, onlooker bees, and scout bees that both the onlookers and the scouts are also called unemployed bees. Employed bees are having no knowledge about food sources and looking for a food source to exploit. Onlooker bees are waiting for the waggle dances exerted by the other bees to establish food sources and scout bees carry out a random search in the environment surrounding the nest for new food sources. In the BC algorithm, each solution to the problem under consideration represented by an n-dimensional real-valued vector is called a food source and the nectar amount of the food resource is evaluated by the fitness value. The following steps show the template of the bee algorithm inspired by the food foraging behavior in a bee colony:

(1) Initialization: compute an initial population.

(2) Employed bee stage: find food sources by exploring the environment randomly.

(3) Onlooker bee stage: select the food sources after sharing the information with employed bees.

(4) Scout bee stage: select one of the solutions; then, replace it with a new randomly generated solution.

(5) Remember the best food source found so far.

(6) If a termination criterion has not been satisfied, go to step (2); otherwise, stop the procedure and report the best food source found so far.

Due to the fact that the basic BC algorithm was originally designed for continuous function optimization, for making it applicable for solving the mixed blocking flow shop problems with makespan criterion, a new variant of the BC algorithm is presented in this section.

3.1. Initialization. In the BC algorithm, initial population is often generated randomly. To guarantee an initial solution with certain quality and diversity, we generate one solution by using the NEH heuristic of Nawaz et al. [13]. Then, to maintain the diversity of the initial population, the other solutions are randomly generated in the entire search space. In Section 4, we show the impact of selecting NEH algorithm and some random solutions instead of selecting just random solution. The framework of the NEH heuristic is presented as follows:

(1) Compute the total processing time on all machines for each job. Then, obtain a sequence $\pi = (\pi_1, \ldots, \pi_n)$

by sorting jobs according to their nonincreasing total processing time.

(2) The first two jobs of π are taken, and the two possible subsequences of these two jobs are evaluated, and then select the better one as the current sequence β. Set $k = 2$.

(3) Set $k = k + 1$. Take the kth job of π and insert it into p ($p \in [1, k + 1]$) possible positions of the current sequence β to obtain p subsequences. Select the subsequence with the minimum makespan as the current sequence β for the next generation. Repeat this step until all jobs are sequenced and the final sequence is found.

3.2. Employed Bee Phase. According to the basic BC algorithm, the employed bees find new solutions in the neighborhood of their current positions. Let $P_0 = \{p_1, \ldots, p_{np}\}$ be the population. For each member p_i, at first, we apply a perturbation method to p_i and generate a new solution p_i'. Then, the path relinking procedure is executed on p_i and p_i' and produces a set of candidate solutions. Finally, a local search (which will be described later) is applied to the best candidate solution with a small probability p_{LS} and the current solution is replaced by the best candidate solution. The pseudocode of employed bee phase procedure is given as follows:

(1) Let $P = \{p_1, \ldots, p_{np}\}$ be the current population and set $i = 1$.

(2) Apply the perturbation method to p_i to obtain a new solution p_i'.

(3) Let $P = \{\overline{p}_1, \ldots, \overline{p}_k\}$ be the set of candidate solutions obtained by executing the path relinking method to p_i and p_i'.

(4) Use the local search method to improve the best candidate solution with probability p_{LS}.

(5) Replace p_i with the best candidate solution.

(6) Let $i = i + 1$. If $i > np$, then, stop. Otherwise, go to step (2).

To complete the discussion, we need to describe the procedures for perturbation, path relinking, and local search. The perturbation procedure is based on the insertion operator. In the insertion operator Ins(x, y) jobs x and y are selected, the job at position x is inserted into position y, and all jobs between positions x and y are shifted accordingly [14]. In the perturbation method, at first, a random position p is selected. Then, for all $p \neq k$, the operator Ins(p, k) is applied to the current solution and the best obtained solution is chosen as the perturbed solution. The path relinking is applied in order to explore the search space between p_i and p_i'. Path relinking is based on the interchange operator. Consider two distinct positions x and y. The operator Exch(x, y) exchanges the positions of the job at position x and the job at position y. In what follows, the path relinking procedure is explained by using an example. Let $p = (2, 1, 4, 5, 3)$ and $p' = (3, 4, 2, 1, 5)$. At first, job 2 which is in position 1 of p is chosen. Since job 2 is in position 3 in p', we apply the operator Exch(1, 3) on p and generate $\overline{p}_1 = (4, 1, 2, 5, 3)$. In the next step, job 4 which is in position 1 of p is selected. Since job 4 is in position 2 of p', we apply Exch(1, 2) on $\overline{p}_1$ to obtain $\overline{p}_2 = (1, 4, 2, 5, 3)$. Similarly, the next generated solution is $\overline{p}_3 = (5, 4, 2, 1, 3)$, obtained by executing Exch(1, 4) on $\overline{p}_2$. The set of candidate solutions obtained by using the path relinking method is $\{\overline{p}_1, \overline{p}_2, \overline{p}_3\}$. As mentioned earlier, the local search method is applied on the best candidate solution with probability p_{LS}. The local search method is also based on the exchange operator. In the local search method, at first, a position k is randomly selected. Then, for all $p \neq k$, the operator Exch(p, k) is applied to the current solution. The result is the best obtained solution. The local search method is repeated until no improvement is observed in a certain number of times (MaxCounter).

3.3. Onlooker Bee Phase. In the onlooker bee phase, we try to improve the quality of the members of the populations. For this purpose, we repeat the following procedure a certain number of times (MaxIter). At first, a member of the population is randomly chosen. Then, the perturbation method (as described in Section 3) is applied to the selected member. Finally, an improvement method (to be described later) is applied to the perturbed solution and if the resulting solution is better than the selected member, then the selected member is replaced with the resulting solution. The pseudocode of the onlooker bee phase is given as follows:

(1) Set $i = 1$.

(2) Let s be a random member of the population.

(3) Let s' be the solution obtained by running the perturbation method to s.

(4) Apply the improvement method to s' to generate a new solution s''.

(5) If s'' is better than s, then set $s = s''$.

(6) Let $i = i + 1$. If $i >$ MaxIter, then, stop. Otherwise, go to step (2).

Two local search methods LS1 and LS2 are used in the improvement method. LS1 is exactly the local search method described in Section 3 and is based on the exchange operator. LS2 is obtained by using the insertion operator instead of the exchange operator in LS1. In LS2, a job is removed from a permutation and inserted into other positions; then, the permutation with the best out of the insertions is retained for the next iteration. In the improvement method, at first, LS1 is applied on the given solution π and a new solution π' is generated. If π' is better than π, we set $\pi = \pi'$. Then, LS2 is applied on π and another solution π'' is produced. Similarly, if π'' is better than π, we set $\pi = \pi''$. If the current solution is not improved by using of LS1 and LS2, then the perturbation method (described in Section 3) is applied on π. This process is repeated until no improvement is observed in a certain number (MaxCnt) of iterations. The pseudocode of this improvement method is given as follows:

(1) Set $i = 1$ and let π be the current solution.

(2) Let π' be the solution obtained after an application of LS1 to π. If π' is better than π, then $\pi = \pi'$.

(3) Let π'' be the solution obtained after an application of LS2 to π'. If π'' is better than π, then $\pi = \pi''$.

(4) If π is not improved after the application of LS1 and LS2, then apply the perturbation method to π and replac π with the perturbed solution.

(5) Let $k = k + 1$. If $k >$ MaxCnt, then, stop. Otherwise, go to step (2).

3.4. Scout Bee Phase. In this phase, the algorithm tries to generate new solutions by executing the following procedure a certain number of times (MaxLoop). At first, two distinct solutions of the population p_1 and p_2 are randomly selected. Then, the better one is determined. Without less of generality, let p_1 be the better one. In the next step, a combination method is applied to the best solution found so far and p_1 to generate a new solution. Finally, p_2 is replaced with the new generated solution. The pseudocode of the scout bees phase procedure is given as follows:

(1) Set $i = 1$.

(2) Let p_1 and p_2 be two members of the population chosen randomly.

(3) If p_1 is better than p_2, then let $p_b = p_1$. Otherwise, $p_b = p_2$.

(4) Let p_* be the solution obtained after an application of the combination method to p_b and p_{best} (p_{best} is the best solution so far).

(5) If $p_b = p_1$, then $p_2 = p_*$; otherwise, let $p_1 = p_*$.

(6) Let $i = i + 1$. If $i >$ MaxLoop, then, stop. Otherwise, go to step (2).

In the following, we describe the combination method. The ideas of the combination method are usually taken from crossover operators of evolutionary algorithms. However, in this paper, the combination method is based on the Taguchi orthogonal arrays. For this reason, it is required to briefly describe the Taguchi orthogonal arrays. To describe the Taguchi orthogonal arrays, we follow the same approach as [15]. Taguchi orthogonal arrays are concerned with k factors, where each factor has q levels. The purpose is to determine the best setting of each factor's level. Clearly, examining all possible q^k combinations often needs a lot of computational effort and is inefficient. Therefore, a small but representative sample of whole combinations is considered for this purpose. Let n be the number of elements of this sample. This sample can be determined by an array with n rows and k columns and is denoted by $O(n, k, q, t)$. All members of the sample must satisfy three conditions. Every level must make the same number of times in any column occur. For every t factors in any t columns, every combination of q levels must make the same number of times occur. The members of the factor must be uniformly distributed over the whole space of all possible combinations. In this paper, the parameter t is always equal to 2 and can be omitted from this notation. In other words, the more simple notation $L_n(q^k)$ can be used instead of $O(n, k, q, t)$. To use $L_n(q^k)$ in the combination method, we need to explain how the best combination of each factor's level is determined. Let $F_{j,r} = \sum_{i=1}^{n} E_i A_{ijr}$, where E_i is the objective function value of the ith member of the sample; if the jth level of the factor in $L_n(q^k)$ is r, then $A_{ijr} = 1$ and otherwise $A_{ijr} = 0$. The sth level of the factor j is the best level of this factor, if $s = \arg\min_{1 \le r \le q} F_{j,r}$.

In the following, an example is presented to explain how Taguchi orthogonal arrays are used to combine some members of the population. Consider the orthogonal array $L_4(2^3)$. Note that $q = 2$, $k = 3$, and $n = 4$. In the combination method, 5 solutions are generated and the best one is returned as the result of the combination method. From these 5 solutions, 4 solutions are generated by using the rows of $L_4(2^3)$ and a solution is generated by using the best level of each factor. At first, we explain how solutions are generated by using the rows of $L_4(2^3)$. Assume that we want to combine two solutions $p_1 = (1, 4, 6, 2, 5, 0, 7, 3)$ and $p_2 = (4, 6, 3, 2, 1, 7, 0, 5)$ by using the third row of $L_4(2^3)$. In the first step, 2 cut points 2 and 5 are randomly chosen and p_1 and p_2 are cut into 3 pieces. These 3 pieces are characterized by grey and white colors. Since the first component of the third row of $L_4(2^3)$ is 1, the first piece of the combined solution is taken from the first piece of p_2. Therefore, the first piece of the combined solution is $(4, 6)$. The second piece of the combined solution is $(6, 2, 5)$ which is taken from p_1, because the second member of the third row of $L_4(2^3)$ is 0. But here, 6 is repeated in the first piece of the combined solution as well. Therefore, only 2 and 5 appeared in the combined solution and the entry corresponding to 6 is left blank. Similarly, as the third member of the third row of $L_4(2^3)$ is 1, the third piece of the combined solution is taken from p_2 and the entry corresponding to the repeated number 5 is left blank. The outcome of this procedure is $(4, 6, ?, 2, 5, 7, 0, ?)$. Note that the missing components of the combined solution are 1 and 3. Now, we consider the first member p_1. The first blank position belongs to 1, because in p_1 at first 1 appeared. Finally, the second blank position is devoted to 3 and the resulting solution is $(4, 6, 1, 2, 5, 7, 0, 3)$. This procedure is depicted in Figure 1. Now, we explain how the best level of each factor is used to construct a solution. As we noted before, to compute the best level of each factor, we have to compute $F_{j,r}$ for $1 \le j \le (k = 3)$ and $1 \le r \le (q = 2)$. Consider Figure 2. $F_{1,0}$ is constructed by using the first column of $L_4(2^3)$. Since level 0 appeared in the first and second row of the first column of $L_4(2^3)$, we have $F_{1,0} = 14 \times 1 + 18 \times 1 + 20 \times 0 + 10 \times 0 = 32$. Similarly, $F_{1,1}$ is also computed by using the first column of $L_4(2^3)$. Since level 1 appeared in the third and fourth row of the first column of $L_4(2^3)$, we have $F_{1,1} = 14 \times 0 + 18 \times 0 + 20 \times 1 + 10 \times 1 = 30$. We have $\arg\min_{0 \le r \le 1} F_{1,r} = 1$. Therefore, the best level of the first factor is 1. By following the same procedure, the best levels of the second and third factors are 1 and 0, respectively. Now, since the best level of the first factor is 0, the first piece of the new solution w_5 is taken from p_1. Similarly, the second and third factors of w_5 are taken from p_2 and p_1, respectively. Finally, from $w_1, w_2, \ldots, w_5$, the best

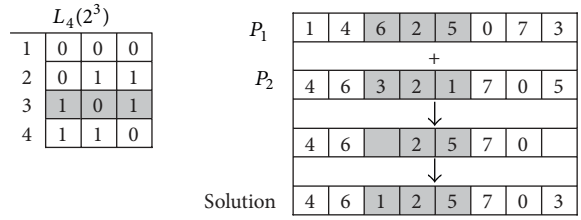

Figure 1: Solution generation by using the third row of $L_4(2^3)$.

	Partial solution							
	Factor 1		Factor 2			Factor 3		
W_1	1	4	6	2	5	0	7	3
W_2	1	4	3	2		7	0	5
W_3	4	6		2	5	7	0	
W_4	4	6	3	2	1	0	7	
Level 0	$F_{1,0} = 14 + 18$		$F_{2,0} = 14 + 20$			$F_{3,0} = 14 + 10$		
Level 1	$F_{1,1} = 20 + 10$		$F_{2,1} = 18 + 10$			$F_{3,1} = 18 + 20$		
Best level	1		1			0		
W_5	4	6	3	2	1	0	7	
Best solution								

Solution								
Factor 1		Factor 2			Factor 3			Fitness
1	4	6	2	5	0	7	3	$E_1 = 14$
1	4	3	2	6	7	0	5	$E_2 = 18$
4	6	1	2	5	7	0	3	$E_3 = 20$
4	6	3	2	1	0	7	5	$E_4 = 10$
4	6	3	2	1	0	7	5	$E_5 = 10$
4	6	3	2	1	0	7	5	10

Figure 2: Combination method.

one is w_5 and is returned as the output of the combination method.

In general case, suppose that we want to use the orthogonal array $L_n(q^k)$ to combine q members $p_1, p_2, \ldots, p_q$ of the population and generate some new solutions. At first, $k-1$ cut points are randomly chosen and $p_1, p_2, \ldots, p_q$ are cut into k pieces. Then, for $1 \le i \le n$, the ith row of the orthogonal array corresponding to $L_n(q^k)$ is considered and a new solution w_i is generated as follows. If the level of the jth factor in the ith row of the orthogonal matrix is $1 \le s \le q$, then the jth part of w_i is taken from the jth part of p_s. Here, missing components of w_i are inserted as described before. In the next step, the fitness value E_i of each w_i is computed. Then, the best level of each factor is determined, as described before, and a new solution w_{n+1} is also generated as follows. If the best level of the jth factor is $1 \le s \le q$, then the jth part of w_i is taken from the jth part of p_s and missing components of w_{n+1} are inserted as described before. Finally, from $w_1, w_2, \ldots, w_{n+1}$, the best one is chosen as the output of the combination method. The pseudocode of the described combination method is given as follows:

(1) Select q members $p_1, p_2, \ldots, p_q$ of the population, randomly.

(2) Randomly, choose $k-1$ cut points and cut $p_1, p_2, \ldots p_q$ into k pieces.

(3) Use the ith row of $L_n(q^k)$ to generate a new member w_i, for $1 \le i \le n$.

(4) Compute the fitness value E_i of each new member w_i, for $1 \le i \le n$.

(5) Compute the value of F_{jr} of factor j with level r, for $1 \le j \le k$, $1 \le r \le q$.

(6) Use the best levels of all factors to generate a new member w_{n+1} and calculate the fitness value E_{n+1} of w_{n+1}.

(7) From $w_1, w_2, \ldots, w_{n+1}$, choose the best one as the output of the combination method.

4. Numerical Results

In this section, we examine the practical efficiency of the BC algorithm. The implementation was performed by using the C programming language on a system with a dual core 3.6 GHz CPU and 2 GB memory. All of the parameters in this study were determined experimentally. The parameters of BC algorithm were set as follows: $P_{\text{size}} = 10$, $p_{\text{LS}} = 0.05$. Two sets of test problems were used to examine the efficiency of the presented algorithm. The first set of test problems is presented in Trabelsi et al. [10], and the second set is given by Taillard [16]. The numerical results of the test problems of Trabelsi are recorded in Tables 2, 3, and 4 and the numerical results of the test problems of Taillard are given in Tables 5(a), 5(b), and 5(c).

Table 1 is concerned with the properties of the 5680 different benchmark problems presented by Trabelsi et al. [10]. The first column and row of this table are concerned with the number of jobs and machines, respectively. The number in the ith row and jth column denotes the number of test problems with m_i machines and n_j jobs, where m_i denotes the ith number of the first column and n_j is the jth number of the first row. For example, the number of test problems with $n_1 = 5$ jobs and $m_5 = 15$ machines is 100.

In Table 2, the computing time of the BC is compared with that of the genetic algorithm (GA) of Trabelsi et al. [10] which is an efficient algorithm of the literature (perhaps

Table 1: Characteristics of the set of test problems presented by Trabelsi.

Size	$m_1 = 5$	$m_2 = 6$	$m_3 = 7$	$m_4 = 10$	$m_5 = 15$	$m_6 = 20$	$m_7 = 50$	$m_8 = 100$
$n_1 = 5$	100	100	100	100	100	100	100	100
$n_2 = 6$	100	100	100	100	100	100	100	100
$n_3 = 7$	100	100	100	100	100	100	100	100
$n_4 = 8$	100	100	100	100	100	100	100	100
$n_5 = 9$	100	100	100	100	100	100	100	100
$n_6 = 10$	100	100	100	100	100	100	100	100
$n_7 = 11$	100	100	100	100	20	20	20	20
$n_8 = 12$	100	100	100	20	20	20	20	20

Table 2: Comparison of BC and GA with respect to computing time.

Alg.	n	$m = 5$	$m = 6$	$m = 7$	$m = 10$	$m = 15$	$m = 20$	$m = 50$	$m = 100$
GA	5	0.00	0.00	0.00	0.00	0.00	0.00	1.00	2.06
BC	5	0.00	0.00	0.00	0.00	0.00	0.01	0.04	0.06
GA	6	0.00	0.00	0.00	0.00	0.00	0.02	1.03	3.01
BC	6	0.00	0.00	0.00	0.00	0.01	0.03	0.06	0.09
GA	7	0.00	0.00	0.00	0.02	0.01	0.01	0.00	0.00
BC	7	0.01	0.01	0.01	0.03	0.03	0.06	0.09	0.17
GA	8	0.00	0.01	0.05	0.25	0.40	0.59	1.97	5.06
BC	8	0.01	0.01	0.03	0.03	0.04	0.06	0.12	0.23
GA	9	0.05	0.08	0.14	0.51	0.85	0.00	3.61	7.15
BC	9	0.03	0.03	0.04	0.04	0.06	0.07	0.15	0.32
GA	10	0.13	0.05	0.13	0.44	0.76	1.10	3.23	6.10
BC	10	0.03	0.03	0.03	0.04	0.07	0.09	0.23	0.45
GA	11	0.48	0.23	0.69	0.82	2.15	2.95	6.5	9.95
BC	11	0.04	0.04	0.04	0.06	0.07	0.14	0.26	0.59
GA	12	0.56	0.39	0.61	1.00	2.35	2.65	5.30	11.15
BC	12	0.04	0.04	0.06	0.06	0.11	0.14	0.32	0.76

Table 3: Comparison of BC and GA with respesct to quality of solutions.

Alg.	n	$m = 5$	$m = 6$	$m = 7$	$m = 10$	$m = 15$	$m = 20$	$m = 50$	$m = 100$
GA	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BC	5	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
GA	6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BC	6	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
GA	7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
BC	7	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
GA	8	0.00	0.00	0.00	0.00	0.00	0.01	0.00	0.00
BC	8	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.01
GA	9	0.00	0.00	0.00	0.02	0.04	0.01	0.02	0.01
BC	9	0.00	0.00	0.00	0.02	0.03	0.02	0.03	0.00
GA	10	0.04	0.02	0.06	0.05	0.10	0.04	0.05	0.05
BC	10	0.02	0.02	0.06	0.04	0.08	0.04	0.04	0.02
GA	11	0.02	0.04	0.09	0.10	0.20	0.29	0.12	0.04
BC	11	0.04	0.02	0.05	0.11	0.15	0.10	0.05	0.05
GA	12	0.06	0.06	0.17	0.25	0.34	0.27	0.23	0.13
BC	12	0.04	0.05	0.11	0.18	0.18	0.28	0.17	0.11

Table 4: Success rates of BC in finding the optimal solution.

$n \times m$	NoI	NS	$n \times m$	NoI	NS	$n \times m$	NoI	NS
5 × 5	100	All	7 × 50	100	All	10 × 15	100	78
5 × 6	100	All	7 × 100	100	All	10 × 20	100	86
5 × 7	100	All	8 × 5	100	All	10 × 50	100	80
5 × 10	100	All	8 × 6	100	All	10 × 100	100	85
5 × 15	100	All	8 × 7	100	All	11 × 5	100	86
5 × 20	100	All	8 × 10	100	All	11 × 6	100	91
5 × 50	100	All	8 × 15	100	All	11 × 7	100	78
5 × 100	100	All	8 × 20	100	All	11 × 10	100	71
6 × 5	100	All	8 × 50	100	99	11 × 15	20	10
6 × 6	100	All	8 × 100	100	99	11 × 20	20	14
6 × 7	100	All	9 × 5	100	All	11 × 50	20	14
6 × 10	100	All	9 × 6	100	All	11 × 100	20	12
6 × 15	100	All	9 × 7	100	All	12 × 5	100	85
6 × 20	100	All	9 × 10	100	95	12 × 6	100	85
6 × 50	100	All	9 × 15	100	94	12 × 7	100	71
6 × 100	100	All	9 × 20	100	93	12 × 10	20	9
7 × 5	100	All	9 × 50	100	90	12 × 15	20	13
7 × 6	100	All	9 × 100	100	96	12 × 20	20	8
7 × 7	100	All	10 × 5	100	92	12 × 50	20	8
7 × 10	100	All	10 × 6	100	94	12 × 100	20	8
7 × 15	100	All	10 × 7	100	89			
7 × 20	100	All	10 × 10	100	89			

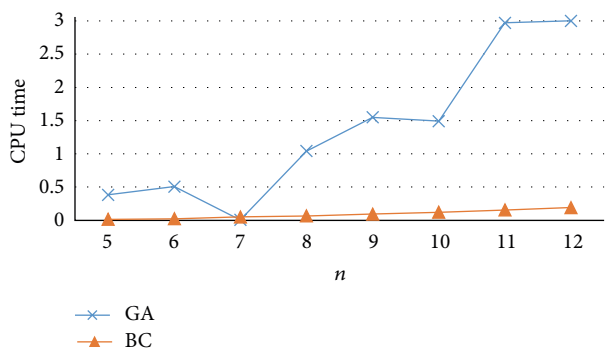

Figure 3: Comparison of BC and GA with respect to computing time.

the best one). In this table, the names of the algorithms are given in the first column. The numbers of jobs and machines are given in the second column and first row, respectively. For each problem instance BC was run 25 times and the average of the computing times is recorded in the column below AvgT. The numerical results of this table show that for large instances the computing time of BC is less than that of GA. From Figure 3, we can see that, with increasing the number of jobs, the difference of computing time of presented algorithm has been raised with GA that shows our algorithm is significantly faster than GA.

In Table 3, the quality of solutions obtained by BC is compared with those of GA. Here, the error percentages are obtained by using the formula %err = $(C_{\max} - \text{Opt})/\text{Opt} \times 100$ and are recorded below the column %err. It can be observed

Table 5: The results of the BC on Taillards benchmark problems.

(a)

Instance	$n * m$	Min	AvgValue	Max	SD	AvgTime
ta001	20 × 5	2265	2268.2	2273	3.92	0.094
ta002	20 × 5	2180	2185.6	2187	3.17	0.109
ta003	20 × 5	2095	2095	2095	0.00	0.093
ta004	20 × 5	2417	2420.7	2438	5.63	0.112
ta005	20 × 5	2097	2106.6	2108	4.26	0.150
ta006	20 × 5	2265	2265	2265	0.00	0.120
ta007	20 × 5	2198	2198	2198	0.00	0.125
ta008	20 × 5	2261	2269.4	2288	13.59	0.098
ta009	20 × 5	2253	2257.52	2270	9.65	0.108
ta010	20 × 5	2042	2065.04	2089	8.13	0.102
ta011	20 × 10	2596	2609.3	2619	10.95	0.18
ta012	20 × 10	2685	2737.72	2805	39.62	0.19
ta013	20 × 10	2456	2486.44	2576	28.64	0.21
ta014	20 × 10	2186	2237.88	2286	42.63	0.22
ta015	20 × 10	2416	2464.88	2509	24.39	0.17
ta016	20 × 10	2289	2324.52	2365	27.64	0.18
ta017	20 × 10	2391	2407.92	2431	19.62	0.20
ta018	20 × 10	2541	2565.6	2605	19.86	0.23
ta019	20 × 10	2571	2604.72	2658	26.99	0.22
ta020	20 × 10	2684	2704.84	2736	16.45	0.19
ta021	20 × 20	3217	3248.6	3269	19.01	0.39
ta022	20 × 20	3010	3051.24	3092	39.59	0.41
ta023	20 × 20	3278	3331.00	3390	38.27	0.43
ta024	20 × 20	3164	3200.76	3234	24.16	0.42
ta025	20 × 20	3288	3338.40	3488	76.42	0.38
ta026	20 × 20	3155	3196.16	3238	30.66	0.39
ta027	20 × 20	3208	3259.48	3296	34.63	0.40
ta028	20 × 20	3127	3185.04	3236	44.78	0.44
ta029	20 × 20	3171	3235.52	3309	41.29	0.38
ta030	20 × 20	3140	3194.6	3256	29.88	0.46
ta031	50 × 5	5197	5210.10	5231	10.80	0.70
ta032	50 × 5	5430	5446.70	5479	13.49	0.72
ta033	50 × 5	5037	5046.00	5060	7.67	0.81
ta034	50 × 5	5368	5393.16	5418	7.28	0.79
ta035	50 × 5	5481	5391.40	5405	12.58	0.86
ta036	50 × 5	5430	5448.20	5473	14.54	0.76
ta037	50 × 5	5123	5146.90	5169	14.00	0.78
ta038	50 × 5	5214	5243.1	5260	11.86	0.73
ta039	50 × 5	5210	5212.40	5218	2.54	0.70
ta040	50 × 5	5351	5366.00	5379	9.47	0.75
ta041	50 × 10	5796	5826.30	5878	23.09	1.43
ta042	50 × 10	5445	5488.40	5523	23.40	1.67

(b)

Instance	$n * m$	Min	AvgValue	Max	SD	AvgTime
ta043	50 × 10	5500	5532.30	5584	24.16	1.51
ta044	50 × 10	5785	5811.60	5834	16.04	1.42
ta045	50 × 10	5790	5816.40	5859	22.89	1.39
ta046	50 × 10	5662	5694.00	5739	25.47	1.48
ta047	50 × 10	5802	5836.50	5868	18.55	1.43

(b) Continued.

Instance	$n * m$	Min	AvgValue	Max	SD	AvgTime
ta048	50 × 10	5668	5699.30	5730	17.89	1.53
ta049	50 × 10	5610	5646.40	5683	21.48	1.48
ta050	50 × 10	5707	5780.60	5825	29.76	1.59
ta051	50 × 20	6820	6873.40	6935	31.94	3.12
ta052	50 × 20	6550	6592.50	6644	25.28	2.95
ta053	50 × 20	6389	6432.90	6465	27.82	3.18
ta054	50 × 20	6504	6558.52	6649	46.62	3.17
ta055	50 × 20	6447	6475.30	6505	16.04	3.10
ta056	50 × 20	6442	6488.20	6530	26.30	3.09
ta057	50 × 20	6507	6544.8	6572	17.70	3.14
ta058	50 × 20	6502	6572.16	6676	29.87	2.95
ta059	50 × 20	6512	6562.10	6631	30.74	2.98
ta060	50 × 20	6675	6728.40	6746	21.32	3.03
ta061	100 × 5	10601	10640.20	10678	22.54	5.71
ta062	100 × 5	10462	10477.9	10505	16.57	3.75
ta063	100 × 5	10216	10233.7	10259	23.56	4.46
ta064	100 × 5	9949	9996.00	10018	37.49	4.68
ta065	100 × 5	10339	10374.5	10432	33.69	5.06
ta066	100 × 5	10039	10079.6	10120	28.46	4.70
ta067	100 × 5	10396	10453.5	10491	31.26	4.75
ta068	100 × 5	10011	10038.50	10072	19.64	4.53
ta069	100 × 5	10651	10681.1	10702	14.96	4.50
ta070	100 × 5	10556	10572.6	10605	22.90	4.68
ta071	100 × 10	11283	11374.1	11446	41.14	9.45
ta072	100 × 10	10779	10846.2	10888	38.55	8.79
ta073	100 × 10	11147	11211.3	11248	27.46	8.87
ta074	100 × 10	11468	11492.80	11534	19.72	7.78
ta075	100 × 10	10937	10994.40	11036	25.33	7.98
ta076	100 × 10	10486	10540.00	10613	36.26	8.89
ta077	100 × 10	10927	10977.9	11019	42.86	7.81
ta078	100 × 10	10938	11015.1	11080	46.44	8.06
ta079	100 × 10	11239	11279.40	11347	35.45	8.76
ta080	100 × 10	11262	11329.90	11386	39.27	8.57
ta081	100 × 20	12180	12237.50	12344	46.28	15.18
ta082	100 × 20	12114	12177.30	12246	39.95	16.31
ta083	100 × 20	12145	12232.70	12299	47.09	14.98
ta084	100 × 20	12158	12211.00	12261	34.14	15.4

(c)

Instance	$n * m$	Min	AvgValue	Max	SD	AvgTime
ta085	100 × 20	12085	12196.80	12264	48.26	15.56
ta086	100 × 20	12244	12300.00	12369	36.41	15.16
ta087	100 × 20	12339	12376.10	12409	19.12	15.43
ta088	100 × 20	12502	12565.50	12652	38.49	15.29
ta089	100 × 20	12295	12352.10	12433	33.47	15.98
ta090	100 × 20	12423	12482.00	12525	37.19	15.04
ta091	200 × 10	21953	22056.70	22136	54.33	46.65
ta092	200 × 10	21570	21607.40	21707	45.16	46.70
ta093	200 × 10	21918	22012.20	22147	73.07	43.73
ta094	200 × 10	21626	21705.5	21768	42.83	41.07
ta095	200 × 10	21683	21822.40	21929	75.52	45.54
ta096	200 × 10	21128	21239.50	21319	60.27	45.62

(c) Continued.

Instance	$n * m$	Min	AvgValue	Max	SD	AvgTime
ta097	200 × 10	21851	21954	22052	58.02	43.28
ta098	200 × 10	21697	21867.40	21978	78.91	46.28
ta099	200 × 10	21512	21555.90	21595	27.84	44.18
ta100	200 × 10	21565	21658.40	21730	45.61	46.96
ta101	200 × 20	23150	23240.80	23299	40.36	102.98
ta102	200 × 20	23479	23574.30	23674	62.57	103.9
ta103	200 × 20	23541	23720.60	23795	76.31	87.84
ta104	200 × 20	23629	23705.00	23793	57.44	93.07
ta105	200 × 20	23276	23351.70	23433	52.30	92.68
ta106	200 × 20	23492	23630.10	23692	57.17	90.36
ta107	200 × 20	23715	23812.10	23875	49.46	90.14
ta108	200 × 20	23575	23686.60	23797	56.42	85.95
ta109	200 × 20	23522	23621.50	23707	52.84	91.00
ta110	200 × 20	23547	23614.70	23685	49.68	93.74

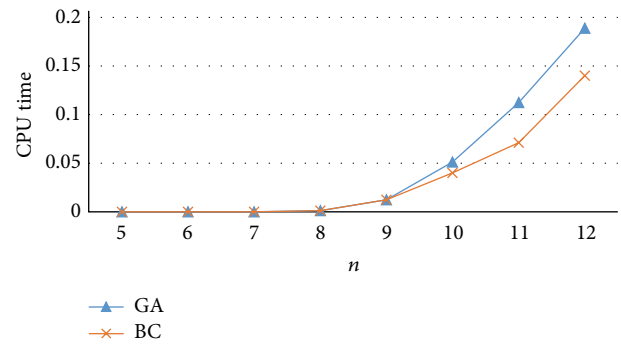

Figure 4: Comparison of BC and GA with respect to quality of solutions.

that the overall mean error value yielded by the BC is equal to 0.0328 which is smaller than 0.0458 generated by the GA. As the problem size increases, the superiority of the BC over GA increases. For the biggest problem analyzed (12 × 100), the average error of the GA was nearly 0.189, while it was 0.14 for the presented BC. Based on Figure 4, we can observe that our algorithm reaches better solution in large problems which indicates the superiority of presented algorithm for mixed blocking flow shop than GA. Therefore, we conclude that the BC is competitive with GA, with respect to the computing time and the quality of the computed solution.

In Table 4, the rates of success of the presented algorithm, in finding the optimal solution of the test problems of Trabelsi, are given. In this table, n is the number of jobs, m is the number of machines, NoI is the number of problem instances (given in Table 1), and NS is the number of times in which BC computes the optimal solution. The numerical results of Table 4 verify that, from 64 cases, in 33 cases, BC computes the optimal solution of all problem instances. In 31 cases, the BC could not find the optimal solution of all of test problems. However, from these 31 cases, in 10 cases, the BC finds the optimal solution of more than (or equal to) 90 percent of test problems. Moreover, in 8 cases, the

Table 6: Comparison of BC-NEH and BC-Random.

Instance	BC-Random	BC-NEH	AIR (%)
20 ∗ 5	2215.206	2213.106	0.095
20 ∗ 10	2516.782	2514.382	0.095
20 ∗ 20	3229.32	3224.08	0.162
50 ∗ 5	5298.458	5290.396	0.152
50 ∗ 10	5721.72	5713.18	0.149
50 ∗ 20	6583.931	6582.828	0.017
100 ∗ 5	10389.03	10354.76	0.330
100 ∗ 10	11201.36	11106.11	0.850
100 ∗ 20	12339.51	12313.1	0.214
200 ∗ 10	21827.46	21747.94	0.364
200 ∗ 20	23792.03	23635.18	0.659
Average	NA	NA	0.281

BC finds the optimal solution of 80 to 90 percent of the test problems. Only in 8 cases, the optimal solution of 70 to 80 percent of test problems is computed. Indeed, from 5680 test problems, only the optimal solution of 346 problems could not be found by using BC. These results show the efficiency of the presented algorithm in finding the optimal solution of the test problems.

For the first time, the BC was tested in the well-known test bed of Taillard [16]. Due to the fact that the largest number of jobs in Trabelsi's benchmark problems is 12, in order to prove the efficiency of our algorithm according to competing approaches of this problem, we conducted still another experiment to test the performance of the BC on the 110 problems presented by Taillard that are larger than Trabelsi's benchmark. In Table 4, the first column is concerned with the problem name. In the second column, the size of the test problems is recorded. Here, the number of jobs is denoted by n and the number of machines is denoted by m. Moreover, the column under Min (Max) is for the objective value of the best (worst) and the standard deviation (SD) of solution found by the presented algorithm. So, all results were obtained after 10 runs for each instance and are given in Table 2. The experimental results reveal that BC has the capability to solve large scale problems and it is effective and has reliable performance.

In addition, in order to prove the efficiency of applying NEH heuristic algorithm as one of the solutions instead of using just random ones as initial solutions, we compare the results of algorithms in these two cases that can be seen in Table 6. In that table, BC-NEH donates BC algorithm when NEH was used as one of the initial solutions, while BC-Random means BC algorithm with only random solutions as initialization phase. Comparison results indicate BC-NEH is 0.281 percent better than BC-Random that shows the efficiency of applying NEH as a solution and increases its effectiveness especially in large instances. It is also worth pointing out that, owing to $P_{\text{size}} = 10$, to maintain the diversity of solutions and decrease the computing time, we just use a heuristic solution instead of applying different heuristic solutions [17] in initialization step.

5. Conclusion

In the current research, a mixed blocking flow shop scheduling problem for minimizing the makespan was discussed. This problem is known to be strongly NP-hard. Here, an efficient bee colony algorithm was proposed to solve this problem. The novelty of the presented method was mainly relevant to the way we applied the ideas of Taguchi orthogonal arrays, path relinking, and some local search methods within the bee colony algorithm, to obtain an efficient algorithm for mixed blocking constraints flow shop scheduling problems. Comparison of the proposed algorithm with an efficient algorithm of the literature demonstrates the efficiency of the proposed BC algorithm. For the first time, the presented algorithm was also tested on well-known test bed of Taillard.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] J. N. D. Gupta and E. F. Stafford, "Flowshop scheduling research after five decades," *European Journal of Operational Research*, vol. 169, no. 3, pp. 699–711, 2006.

[2] L. Wang, Q.-K. Pan, P. N. Suganthan, W.-H. Wang, and Y.-M. Wang, "A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems," *Computers & Operations Research*, vol. 37, no. 3, pp. 509–520, 2010.

[3] L. Wang, L. Zhang, and D.-Z. Zheng, "An effective hybrid genetic algorithm for flow shop scheduling with limited buffers," *Computers & Operations Research*, vol. 33, no. 10, pp. 2960–2971, 2006.

[4] L. Wang, Q.-K. Pan, and M. F. Tasgetiren, "A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem," *Computers and Industrial Engineering*, vol. 61, no. 1, pp. 76–83, 2011.

[5] I. Ribas, R. Companys, and X. Tort-Martorell, "An iterated greedy algorithm for the flowshop scheduling problem with blocking," *Omega*, vol. 39, no. 3, pp. 293–301, 2011.

[6] S. Dauzere-Peres, C. Pavageau, and N. Sauer, "Modelisation et resolution par PLNE dun probleme reel dordonnancement avec contraintes de blocage," in *3eme Congres ROADEF*, Nantes, France, January 2000.

[7] S. Martinez, *Ordonnancement de systems de production avec contraintes de blocage [Ph.D. thesis]*, University of Nantes, Nantes, France, 2005.

[8] G. I. Zobolas, C. D. Tarantilis, and G. Ioannou, "Minimizing makespan in permutation flow shop scheduling problems using a hybrid metaheuristic algorithm," *Computers and Operations Research*, vol. 36, no. 4, pp. 1249–1267, 2009.

[9] W. Trabelsi, C. Sauvey, and N. Saur, "Heuristic methods for problems with blocking constraints solving jobshop scheduling," in *Proceedings of the 8th International Conference on Modeling and Simulation (MOSIM '10)*, Hammamet, Tunisia, May 2010.

[10] W. Trabelsi, C. Sauvey, and N. Sauer, "Heuristics and metaheuristics for mixed blocking constraints flowshop scheduling problems," *Computers and Operations Research*, vol. 39, no. 11, pp. 2520–2527, 2012.

[11] S. Martinez, S. Dauzère-Pérès, C. Guéret, Y. Mati, and N. Sauer, "Complexity of flowshop scheduling problems with a new blocking constraint," *European Journal of Operational Research*, vol. 169, no. 3, pp. 855–864, 2006.

[12] D. Karaboga and B. Basturk, "A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm," *Journal of Global Optimization*, vol. 39, no. 3, pp. 459–471, 2007.

[13] M. Nawaz, E. E. Enscore Jr., and I. Ham, "A heuristic algorithm for the *m*-machine, *n*-job flow-shop sequencing problem," *Omega*, vol. 11, no. 1, pp. 91–95, 1983.

[14] B. Jarboui, M. Eddaly, and P. Siarry, "An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems," *Computers & Operations Research*, vol. 36, no. 9, pp. 2638–2646, 2009.

[15] J.-T. Tsai, T.-K. Liu, and J.-H. Chou, "Hybrid Taguchi-genetic algorithm for global numerical optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 4, pp. 365–377, 2004.

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

[17] Y.-Y. Han, Q.-K. Pan, J.-Q. Li, and H.-Y. Sang, "An improved artificial bee colony algorithm for the blocking flowshop scheduling problem," *International Journal of Advanced Manufacturing Technology*, vol. 60, no. 9–12, pp. 1149–1159, 2012.