Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2016, Article ID 7373617, 11 pages
http://dx.doi.org/10.1155/2016/7373617

Research Article

An Enhanced Discrete Artificial Bee Colony Algorithm to Minimize the Total Flow Time in Permutation Flow Shop Scheduling with Limited Buffers

Guanlong Deng, Hongyong Yang, and Shuning Zhang

School of Information and Electrical Engineering, Ludong University, Yantai 264025, China

Correspondence should be addressed to Guanlong Deng; dglag@163.com

Received 25 January 2016; Accepted 18 May 2016

Academic Editor: Vladimir Turetsky

This paper presents an enhanced discrete artificial bee colony algorithm for minimizing the total flow time in the flow shop scheduling problem with buffer capacity. First, the solution in the algorithm is represented as discrete job permutation to directly convert to active schedule. Then, we present a simple and effective scheme called best insertion for the employed bee and onlooker bee and introduce a combined local search exploring both insertion and swap neighborhood. To validate the performance of the presented algorithm, a computational campaign is carried out on the Taillard benchmark instances, and computations and comparisons show that the proposed algorithm is not only capable of solving the benchmark set better than the existing discrete differential evolution algorithm and iterated greedy algorithm, but also capable of performing better than two recently proposed discrete artificial bee colony algorithms.

1. Introduction

In the scope of scheduling problem, the permutation flow shop scheduling problem (PFSP) is one of the most important and studied issues because of its theoretical complexity and practical application. The traditional permutation flow shop model is not concerned with the capacity for buffer between two consecutive machines, and once its processing on a machine is finished, a job waits till the next machine is available to process it. However, in real production environments, the buffers are usually limited. Examples lie in the petrochemical processing industries and cell manufacturing [1]. In such a scheduling problem, after finishing its operation on a machine, if the next machine is not available, a job is allowed to store in a buffer only if the buffers are not full. If the buffers are full, the job must wait on the incumbent machine, which may make the machine unable to process other jobs. One special case in the permutation flow shop scheduling problem with limited buffers (LBPFSP) is with no buffer, and the problem is called the blocking flow shop scheduling problem (BPFSP). The BPFSP has gained much attention in the past decades [2, 3] and its strong NP-hard characteristics were validated for the case with more than two machines [4]. Besides, the LBPFSP is also strongly NP-hard even for only two machines [5].

A great amount of research work has been carried out for the BPFSP. Many heuristics were introduced or proposed for the problem [6–9], but they are not good enough, especially for problem instance with big size. In recent years, lots of sophisticated metaheuristics have been developed for the problem. For the makespan criterion, the developed metaheuristics include genetic algorithm (GA) [10], tabu search (TS) algorithm [11], hybrid discrete differential evolution (HDDE) algorithm [12], iterated greedy (IG) algorithm by [2], hybrid modified global-best harmony search (hmgHS) algorithm [13], and variable neighborhood search (VNS) [14]. Recently, some researchers also proposed algorithms to minimize the total flow time (TFT) of the BPFSP. Wang et al. [15] developed an hmgHS algorithm and Deng et al. [16] put forward a discrete artificial bee (DABC) algorithm.

As a more general problem, the LBPFSP received increasing attention in recent years. An early overview article was provided by Leisten [7], and the article concluded that the NEH heuristic is competitive. Smutnicki [17] presented a TS

algorithm for the case with two machines, and the TS algorithm was later generalized to the case with more machines by Nowicki [18]. Also, an effective TS algorithm was developed by Brucker et al. [19]. Later, a hybrid genetic algorithm (HGA) by Wang et al. [20] was shown to outperform the TS algorithm. Further, Liu et al. [21] presented a hybrid particle swarm optimization (HPSO) algorithm that yielded better results than HGA. Qian et al. [22] investigated a hybrid differential evolution (HDE) algorithm for not only the finite buffer case but also the blocking and infinite buffer case. An immune based approach (IA) was developed by Hsieh et al. [23] and its superiority over the HGA was asserted. Recently, in two papers, Pan et al. [24, 25] proposed two metaheuristics, chaotic harmony search (CHS) and HDDE, and showed their superiority over the HGA and HPSO algorithm, respectively. More recent work was developed by Zhao et al. [26] and Moslehi and Khorasanian [27]. The former proposed an improved PSO algorithm while the latter presented a hybrid variable neighborhood search (HVNS) hybridizing variable neighborhood search and simulated annealing algorithm. In the HVNS algorithm, a speed-up method was developed for several kinds of local search methods.

In the past decades, a bunch of metaheuristics based on swarm intelligence has been proposed and applied to scheduling problems [28, 29]. Among them, the artificial bee colony (ABC) algorithm [30–33] performed well in continuous function optimization, and Pan et al. [34] firstly proposed a discrete version of the ABC (DABC) algorithm for the lot-streaming flow shop scheduling. Then, Tasgetiren et al. [35] and Deng et al. [16] also developed a DABC algorithm for the PFSP and BPFSP, respectively. However, to the best of our knowledge, there is no published study on solving the LBPFSP using this algorithm. As for the LBPFSP, the existing work all focused on the makespan minimization, and no research work has been done with the TFT criterion, despite the prominence of the TFT criterion. Therefore, this paper aims to present a simple and effective DABC algorithm for the LBPFSP with the TFT criterion, which is not a well-studied scheduling problem. The developed DABC algorithm is based on the hybridization of ABC algorithm paradigm and local search methodology, and its performance is investigated by extensive experiments.

The rest of the paper is organized as follows. In Section 2, the considered problem with the TFT criterion is introduced and formulated. The proposed DABC algorithm is then presented as a simple and effective method for the TFT criterion case in Section 3. Section 4 provides the parameter calibration and performance investigation based on computational experiments. Finally, Section 5 gives out the conclusions and future work of the paper.

2. Problem Formulation

In the LBPFSP with the TFT criterion, there are a set of n jobs $N = \{1, 2, \ldots, n\}$ and a set of m machines $M = \{M_1, M_2, \ldots, M_m\}$. The operation of job j ($j = 1, 2, \ldots, n$) on machine M_i ($i = 1, 2, \ldots, m$) requires a nonnegative time given as p_{ij}. Every job has to be processed consecutively from the first machine M_1 to the last machine M_m. The following traditional flow shop assumptions apply. (1) All jobs are independent and available for processing at time zero. (2) At any time, each job is being processed at most on one machine and each machine is processing at most one job. (3) There is no breaking down in machines. (4) An operation can not be interrupted or split. (5) The setup and release times are ignored. Besides, the "permutation" requires that the job processing sequence must be the same on all machines. Between two consecutive machines M_i and M_{i+1}, there is a buffer with the capacity equal to B_i ($B_i \geq 0$, $i = 1, 2, \ldots, m-1$). Therefore, the number of stored jobs between two consecutive machines is at most B_i. If no buffer exists and the downstream machine is busy, a completed job has to stay on the current machine and thus may block it. The TFT is defined as $\sum_{j=1}^{n} C_j$, where C_j is the time when job j is finished. The objective is to minimize the TFT.

Since the TFT belongs to regular optimality criteria, there exists at least one active schedule that is optimal, and thus each schedule can be represented as a job permutation $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$, where the job is processed as early as possible with respect to the given sequence in π. Let TFT(π) denote the total flow time of π and let $d_{\pi(j),i}$ denote the leaving time of job $\pi(j)$ from machine M_i. The values of $d_{\pi(j),i}$ can be calculated as follows [25]:

$$
\begin{aligned}
&d_{\pi(1),1} = p_{\pi(1),1},\\
&d_{\pi(1),i} = d_{\pi(1),i-1} + p_{\pi(1),i}, \quad i = 2, \ldots, m,\\
&d_{\pi(j),1} = d_{\pi(j-1),1} + p_{\pi(j),1}, \quad j = 2, \ldots, B_1 + 1,\\
&d_{\pi(j),i} = \max\left\{d_{\pi(j-1),i}, d_{\pi(j),i-1}\right\} + p_{\pi(j),i},\\
&\qquad\qquad j = 2, \ldots, B_i + 1,\ i = 2, \ldots, m-1,\\
&d_{\pi(j),1} = \max\left\{d_{\pi(j-1),1} + p_{\pi(j),1}, d_{\pi(j-B_1-1),2}\right\},\\
&\qquad\qquad j > B_1 + 1,\\
&d_{\pi(j),i} = \max\left\{\max\left\{d_{\pi(j-1),i}, d_{\pi(j),i-1}\right\}\right.\\
&\quad \left. + p_{\pi(j),i}, d_{\pi(j-B_i-1),i+1}\right\},\\
&\qquad\qquad j > B_i + 1,\ i = 2, \ldots, m-1,\\
&d_{\pi(j),m} = \max\left\{d_{\pi(j-1),m}, d_{\pi(j),m-1}\right\} + p_{\pi(j),m},\\
&\qquad\qquad j = 2, \ldots, n.
\end{aligned}
\tag{1}
$$

Using the above recursion, we can calculate the TFT with time complexity $O(mn)$:

$$\mathrm{TFT}(\pi) = \sum_{j=1}^{n} d_{\pi(j),m}. \tag{2}$$

If all permutations are denoted as set Π, then we have to find a permutation π^* in Π such that

$$\mathrm{TFT}(\pi^*) \leq \mathrm{TFT}(\pi) \quad \forall \pi \in \Pi. \tag{3}$$

Clearly, if $B_i = 0$, then the problem is the same as BPFSP. If $B_i \geq n-1$, then the problem can be treated as PFSP. Due to

the extensive work carried out for the BPFSP and PFSP, we will investigate the not-well-studied case; namely, the problem with the buffer size is finite.

3. Discrete Artificial Bee Colony Algorithm

According to the framework of the ABC algorithm, the algorithm includes three kinds of bees, namely, employed bee, onlooker bee, and scout bee. The solutions (called food sources) of the algorithm form a population with size NP. After initialization of the population, the algorithm goes into an iteration till the stopping criterion is satisfied. In the iteration, the algorithm sends first each employed bee, then each onlooker bee, and finally each scout bee to explore food sources. Since the ABC algorithm is originally proposed for continuous function optimization, it needs the conversion from real domain to discrete domain if the continuous coding solution is used. Due to the discrete characteristic of the considered problem, this paper uses job permutation as solution representation and puts forward a discrete ABC algorithm. To make the algorithm simple yet effective, we adopt the idea of iterated greedy (IG) algorithm of Ruiz and Stützle [36]. The IG algorithm mainly includes two important procedures. First, the destruction and construction procedure produce a new solution by perturbing the incumbent solution which is usually a local optimum. By iteratively searching the insertion neighborhood of the new solution, a local search is imposed on the new solution. These two procedures are modified or improved in the new DABC algorithm to design the operators of the employed, onlooker, and scout bees. All the elements are elaborated in the following subsections.

3.1. Initialization. As mentioned above, the DABC algorithm consists of NP food sources, where NP is a parameter controlling population size. For each food source, we need to generate a job sequence $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$. The NEH heuristic and its variants are developed to construct the initial population with both quality and diversity. Wang et al. [15] pointed out that if the jobs are sequenced in increasing order rather than decreasing order in NEH, the obtained heuristic performs better than NEH heuristic for BPFSP with the TFT criterion. They denoted the variant as NEH_WPT heuristic. Besides, if the jobs are sequenced in random order in NEH, the obtained heuristic is a randomized heuristic, and it also works well according to our pilot experiments. We denote this randomized heuristic NEH_RAN. In our proposed algorithm, the solutions generated by both the NEH and NEH_WPT heuristics are included in the initial population, and the remaining NP-2 solutions of the initial population are generated by the NEH_RAN heuristic. Such an initialization scheme gives a guarantee of the population with good quality and diversity.

3.2. Employed Bee. For each solution in the population, the employed bee is firstly applied. Thus there are also NP employed bees. In the employed bee phase, a procedure, bestinsert, is presented to find a neighboring food source from the incumbent food source.

Suppose that a permutation is denoted as $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ and $s = \pi(j)$ is a job with position index j. By inserting job s into kth ($k \in \{1, \ldots, n\} \setminus \{j\}$) position, we will get a permutation $\omega(s, k)$. Let π^s_{binsert} denote the permutation resulting in the minimum objective value among all $\omega(s, k)$ permutations. The bestinsert procedure is illustrated in Algorithm 1.

The bestinsert procedure is designed as a perturbation operator to escape from local optima. The idea behind the bestinsert procedure is that making several compulsory insert moves would result in a solution that is usually different from but keeps probably the good characteristics of the incumbent solution. The setting of parameter d determines the degree of perturbation.

Each employed bee employs the bestinsert procedure to generate a new food source. This generated food source is not directly put into the population but used by its corresponding onlooker bee.

3.3. Onlooker Bee. Before describing the design of the onlooker bee phase, we introduce several local search methods and present the combined local search.

For the PFSP, most of the excellent local search methods consider the insertion neighborhood. The superiority of this neighborhood structure has been shown in lots of papers, such as [36–41]. In the insertion-based local search methods embedded in IG algorithms by Ruiz and Stützle [36], a job s is randomly chosen, and its π^s_{binsert} with respect to the incumbent solution π is then identified. If the solution π^s_{binsert} is better than the incumbent solution, the incumbent solution is replaced. The above process is repeated for all n jobs, which means that s is randomly and unrepeated chosen for n times. Furthermore, once the incumbent solution is updated for a job's process, the processes of all n jobs need to be performed. The local search terminates when no improvement occurs for the processes of all n jobs. Pan et al. [39] improved this local search and presented the referenced local search (RLS). In RLS, jobs to be inserted are selected not randomly but according to the precedence of a referenced solution. Besides, the local search is optimized and the redundant process of finding π^s_{binsert} may be avoided. Similarly, Deng and Gu [40] also improved this local search but used a random order in which jobs are to be inserted. Their insertion-based local search (ILS) is shown in Algorithm 2.

It can be seen from Algorithm 2 that the job s to be inserted is chosen according to a random order π_D, and the procedure terminates once the process of finding π^s_{binsert} causes no improvement of π for consecutive n times. The effectiveness of the ILS inspired us to present a swap-based local search (SLS) with homogeneous structure. The SLS uses the swap neighborhood, and π^s_{bswap} is defined like π^s_{binsert}. Let $s = \pi(j)$ be a job scheduled in $\pi = (\pi(1), \pi(2), \ldots, \pi(n))$ and let $\nu(s, k)$ denote the sequence generated by swapping job s with the job occupying kth ($k \in \{1, \ldots, n\} \setminus \{j\}$) position of π. π^s_{bswap} is the permutation resulting in the minimum objective value among all $\nu(s, k)$ permutations. The procedure of SLS is illustrated in Algorithm 3.

It should be pointed out that there is a possibility that a local optimum provided by ILS is not a local optimum when

(1) choose d unrepeated jobs $J_1, \ldots, J_d$ randomly and let IL = $\{J_1, \ldots, J_d\}$
(2) **while** (IL is not empty)
(3) take out the front job s from IL and delete it from IL
(4) j = the position index of job s in π
(5) $W = \varnothing$
(6) **for** $k = 1$ to $j - 1$
(7) add $\omega(s, k)$ into W
(8) **endfor**
(9) **for** $k = j + 1$ to n
(10) add $\omega(s, k)$ into W
(11) **endfor**
(12) π^s_{binsert} = the best permutation in W
(13) $\pi = \pi^s_{\text{binsert}}$
(14) **endwhile**

Algorithm 1: Bestinsert procedure.

(1) π_D = a permutation generated randomly
(2) $i = 0, h = 1$
(3) **while** $(i < n)$
(4) let $s = \pi_D(h)$
(5) j = the position index of job s in π
(6) $W = \varnothing$
(7) **for** $k = 1$ to $j - 1$
(8) add $\omega(s, k)$ into W
(9) **endfor**
(10) **for** $k = j + 1$ to n
(11) add $\omega(s, k)$ into W
(12) **endfor**
(13) π^s_{binsert} = the best permutation in W
(14) **if** (π^s_{binsert} is better than π)
(15) $\pi = \pi^s_{\text{binsert}}$
(16) $i = 1$
(17) **else**
(18) $i = i + 1$
(19) **endif**
(20) $h = (h + 1)\%\ n$
(21) **endwhile**

Algorithm 2: Insertion-based local search.

(1) π_D = a permutation generated randomly
(2) $i = 0, h = 1$
(3) **while** $(i < n)$
(4) let $s = \pi_D(h)$
(5) j = the position index of job s in π
(6) $W = \varnothing$
(7) **for** $k = 1$ to $j - 1$
(8) add $v(s, k)$ into W
(9) **endfor**
(10) **for** $k = j + 1$ to n
(11) add $v(s, k)$ into W
(12) **endfor**
(13) π^s_{bswap} = the best permutation in W
(14) **if** (π^s_{bswap} is better than π)
(15) $\pi = \pi^s_{\text{bswap}}$
(16) $i = 1$
(17) **else**
(18) $i = i + 1$
(19) **endif**
(20) $h = (h + 1)\%\ n$
(21) **endwhile**

Algorithm 3: Swap-based local search.

SLS is applied. So, we present the combined local search (CLS) by applying ILS and SLS iteratively till a local optimum is reached. The procedure is given in Algorithm 4.

The number of onlooker bees is also NP. The onlooker bee applies the CLS to the food source returned by the employed bee. If the solution returned by CLS is not worse than the corresponding food source in the population, the corresponding food source in the population is replaced, or else it does not change. Note that the NP food sources in the population and the NP onlooker bees correspond one to one, which means whether ith food source is updated only depends on the solution found by ith onlooker bee. Setting the number of onlooker bees as NP can keep the parallel paradigm of the algorithm and benefit the depth and breadth

(1) apply ILS to π
(2) **while** (**true**)
(3) apply SLS to π
(4) **if** (π is not improved during the previous Step)
(5) break
(6) **endif**
(7) apply ILS to π
(8) **if** (π is not improved during the previous Step)
(9) break
(10) **endif**
(11) **endwhile**

Algorithm 4: Combined local search.

```
(1) set parameters NP, d, ds
(2) generate the initial population
(3) π_b = the best solution in the population
(4) while (not termination)
(5)     for (each employed bee)
(6)         apply bestinsert to its solution in the population
(7)     endfor
(8)     for (each onlooker bee)
(9)         apply CLS to the food source found by its employed bee
(10)            update π_b if possible
(11)    endfor
(12)    for (each scout bee)
(13)        produce a food source based on π_b
(14)        put the food source in the population by tournament selection
(15)        update π_b if possible
(16)    endfor
(17) endwhile
```

Algorithm 5: Procedure of the DABC algorithm.

of the algorithm's search. Additionally, it can decrease the number of the algorithm's parameters to be calibrated.

3.4. Scout Bee. There are two choices for a scout bee. It can either generate a food source randomly or produce a food source based on the best solution π_b. The latter tends to be more effective since the best solution in the current population often maintains better characteristics than others and the solution region around it could be more promising than others. Therefore, in the proposed DABC algorithm, the scout bee is designed to produce a food source by performing the bestinsert procedure and the ILS on the best solution π_b. First, the bestinsert procedure with parameter ds is performed on π_b and generates a new food source, and then the new food source is further searched by the ILS. The finally obtained food source by the scout bee is put in the population through a tournament selection. The tournament selection randomly chooses two solutions in the population, and the worse one is replaced with the considered food source. For simplicity of the parameter setting, the number of the onlooker bees is set to 0.1NP.

3.5. Proposition of the DABC Algorithm. Since the details of all components of the DABC algorithm have been given out, the whole computational procedure is outlined in Algorithm 5. Such an algorithm is expected to solve the LBPFSP with the TFT criterion effectively and efficiently.

4. Computations and Comparisons

A large amount of computational experiments is carried out to test the performance of the presented DABC algorithm. The well-known Taillard benchmark instances with different sizes are used. In this paper, Taillard benchmark instances originally produced for the PFSP are treated as the LBPFSP with the TFT criterion. All the tested algorithms are programmed in C++ language and the running environment is a PC with Intel Core (TM) i5-2400 3.1 GHz processor. The relative percentage deviation (RPD) is calculated to indicate the amount of improvement over the reference solution. Consider

$$\mathrm{RPD} = \frac{\mathrm{TFT}_A - \mathrm{TFT}_{\mathrm{ref}}}{\mathrm{TFT}_{\mathrm{ref}}} \times 100, \tag{4}$$

where TFT_A is the TFT of the solution obtained by the tested algorithm A and $\mathrm{TFT}_{\mathrm{ref}}$ is the TFT of the reference solution.

The reference solutions are the best solutions in all of these computational experiments for all algorithms, and they are shown in the Appendix for all tested instances. Clearly, the lower the RPD value is, the better results the algorithm yields.

4.1. Algorithm Calibration. In this section, we carry out an experiment to calibrate the proposed DABC algorithm (denoted by DABC). Since the computational efforts of the CLS are usually more than that of the local search employing a single neighborhood structure and the CLS is performed for NP times in each generation of the DABC algorithm, we suggest that the parameter NP is not too large, especially when the allowed computational time of the algorithm is relatively less. For all computations of the DABC algorithm in this paper, we set NP to 10 and the stopping criterion is elapsed CPU time not less than $3n^2m$ milliseconds. Setting this CPU time related to the instances size allows the algorithm more time to solve the larger size instances that are probably "harder." In the calibration experiment, we perform a large Design of Experiments [42], and the following factors are tested: (1) the type of local search (LS), tested at three levels: the local search by Ruiz and Stützle [36] (denoted by LS_RS), ILS, and CLS; (2) the parameter d, tested at eight levels: 2–9; (3) the parameter ds, tested at eight levels: 2–9. Nine instances, Ta01, Ta11, ..., Ta81, are selected from each problem group to avoid bias of the results, and the algorithm is run for 10 replications with each parameter configuration for each selected instance. For simplicity, they are treated

Table 1: ANOVA results for the experiment on the calibration of DABC.

Source	Sum of squares	Df	Mean square	F value	p value
Main effects					
A: LS	5.37	2	2.69	62.23	<0.0001
B: d	5.81	7	0.83	19.23	<0.0001
C: ds	0.37	7	0.053	1.23	0.2836
Interactions					
AB	$6.192e-005$	14	$4.423e-006$	$1.025e-004$	1.0000
AC	$6.097e-005$	14	$4.435e-006$	$1.009e-004$	1.0000
BC	5.59	49	0.11	2.64	<0.0001

as the LBPFSP with all buffers equal to one. In all, the multifactor experimental design yields $3 \times 8 \times 8 \times 10 \times 9 = 17280$ results. With such a large data set, the Analysis of Variance (ANOVA) technique is introduced to draw a convincing conclusion of parameter calibration. The ANOVA results are shown in Table 1.

It is concluded from Table 1 that factor LS and factor d are statistically significant for the algorithm performance due to its p value less than 0.0001, while factor ds is not statistically significant with a p value equal to 0.2836. Besides, we note that the interaction of parameters d and ds is also significant, which is understandable since the employed bee phase is related to the scout bee phase.

Furthermore, to illustrate the differences of algorithm performance with different parameter values, we reproduce the one-factor means plots with 95% Least Significant Difference (LSD) confidence intervals of the factors LS and d, shown in Figure 1. According to the statistical theory, it is seen from Figure 1 that, for the local search method, the proposed CLS is statistically better than ILS and ILS is statistically better than LS_RS. For the parameter d, the setting value 7 is statistically better than the setting values 2–6. As regards parameter ds, the differences are small and its means plot is omitted for simplicity. Finally, we calibrate the DABC algorithm, using combined local search, as $d = 7$ and $ds = 4$.

4.2. Computational Comparisons. In the comparisons with other algorithms from the literature, the proposed algorithm uses the calibrated parameter setting. To our knowledge, the LBPFSP with the TFT criterion has not been well studied, so we take four well-performed algorithms from the PFSP literature and adapt them for the considered problem in this paper. The algorithms selected for comparisons are the following: (1) the iterated greedy algorithm [36] (IG); (2) the hybrid discrete differential evolution [25] (HDDE) algorithm; (3) the discrete artificial bee colony algorithm [35] (DABC_T); and (4) the discrete artificial bee colony algorithm [16] (DABC_D). All the above compared algorithms are reimplemented for the considered problem and performed under the original algorithm's parameter settings. Wang et al. [20] reported that when the buffer size is equal to 4, the problem is very close to the case with the buffers of infinite capacity. Therefore, here, all the five algorithms treat the problem with unitary buffer size B equal to 1, 2, 3, and 4. For each instance in all the 90 Taillard benchmark instances, each algorithm is run 10 times. In total, we have $5 \times 4 \times 90 \times 10 = 18000$ data points. The average relative percentage deviation (ARPD) values grouped in subsets of different sizes are summarized in Tables 2–5 for each buffer size, respectively.

Since the five algorithms are all executed in the same computational environment with the same stopping criteria,

Table 2: ARPD of each algorithm on Taillard benchmark set ($B = 1$).

$n \times m$	DABC	DABC_D	DABC_T	HDDE	IG_RS
20×5	0.00	0.00	0.00	0.00	0.04
20×10	0.00	0.00	0.00	0.00	0.03
20×20	0.00	0.00	0.00	0.00	0.03
50×5	0.30	0.35	0.42	0.61	1.14
50×10	0.30	0.36	0.46	0.64	0.89
50×20	0.23	0.29	0.42	0.49	0.75
100×5	0.34	0.40	0.44	0.80	1.51
100×10	0.29	0.35	0.47	0.63	0.97
100×20	0.27	0.33	0.47	0.60	0.84
Overall mean	0.19	0.23	0.30	0.42	0.69

Table 3: ARPD of each algorithm on Taillard benchmark set ($B = 2$).

$n \times m$	DABC	DABC_D	DABC_T	HDDE	IG_RS
20×5	0.00	0.00	0.00	0.00	0.02
20×10	0.00	0.00	0.00	0.00	0.03
20×20	0.00	0.00	0.00	0.00	0.02
50×5	0.19	0.24	0.30	0.34	0.47
50×10	0.21	0.27	0.36	0.49	0.72
50×20	0.23	0.29	0.41	0.47	0.66
100×5	0.18	0.23	0.33	0.49	0.88
100×10	0.32	0.38	0.48	0.67	0.97
100×20	0.33	0.39	0.51	0.58	0.86
Overall mean	0.16	0.20	0.26	0.34	0.52

Table 4: ARPD of each algorithm on Taillard benchmark set ($B = 3$).

$n \times m$	DABC	DABC_D	DABC_T	HDDE	IG_RS
20×5	0.00	0.00	0.00	0.00	0.02
20×10	0.00	0.00	0.00	0.00	0.02
20×20	0.00	0.00	0.00	0.00	0.02
50×5	0.16	0.22	0.22	0.24	0.41
50×10	0.28	0.34	0.39	0.45	0.65
50×20	0.22	0.28	0.35	0.43	0.69
100×5	0.18	0.23	0.32	0.43	0.74
100×10	0.26	0.31	0.47	0.58	0.91
100×20	0.27	0.33	0.42	0.49	0.72
Overall mean	0.15	0.19	0.24	0.29	0.46

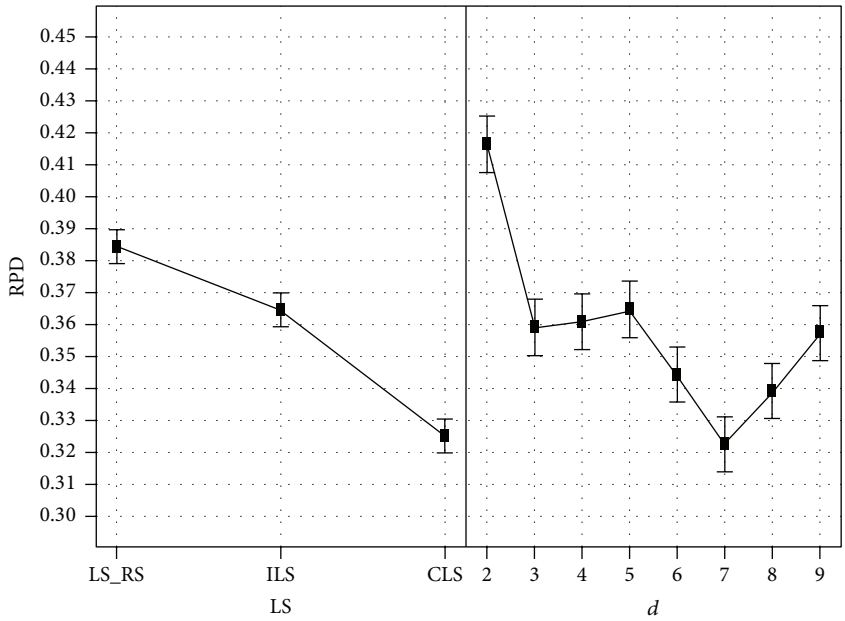

Figure 1: Means plot with 95% LSD intervals for the type of local search and the parameter d of the DABC algorithm.

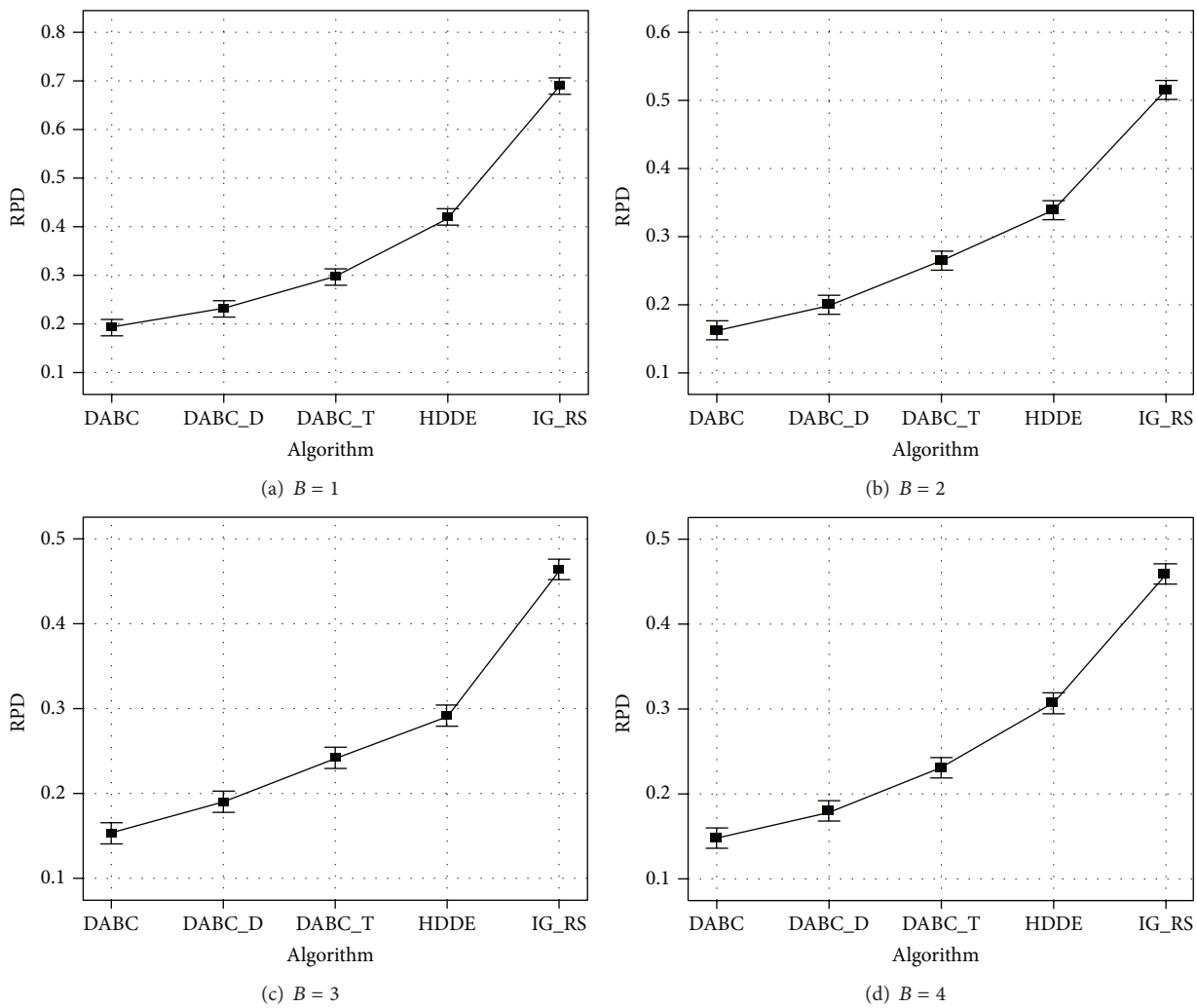

Figure 2: Means plot with 95% LSD intervals for different algorithms.

Table 5: ARPD of each algorithm on Taillard benchmark set ($B = 4$).

$n \times m$	DABC	DABC_D	DABC_T	HDDE	IG_RS
20×5	0.00	0.00	0.00	0.00	0.05
20×10	0.00	0.00	0.00	0.00	0.03
20×20	0.00	0.00	0.00	0.00	0.02
50×5	0.13	0.17	0.24	0.29	0.43
50×10	0.27	0.32	0.34	0.44	0.67
50×20	0.23	0.28	0.38	0.50	0.61
100×5	0.12	0.17	0.28	0.41	0.63
100×10	0.28	0.33	0.40	0.55	0.84
100×20	0.29	0.34	0.45	0.57	0.85
Overall mean	0.15	0.18	0.23	0.31	0.46

the results are fully and completely comparable. Tables 2–5 validate the superiority of the DABC algorithm over the other compared algorithms. The overall mean RPD values yielded by the DABC algorithm are 0.19, 0.16, 0.15, and 0.15 when buffer size is equal to 1, 2, 3, and 4, respectively, which are substantially lower than those (0.23, 0.20, 0.19, and 0.18) obtained by the DABC_D algorithm, those (0.30, 0.26, 0.24, and 0.23) obtained by the DABC_T algorithm, those (0.42, 0.34, 0.29, and 0.31) obtained by the HDDE algorithm, and those (0.69, 0.52, 0.46, and 0.46) obtained by the IG_RS algorithm. Furthermore, for each buffer size, the DABC algorithm has a lower ARPD value than all the other algorithms for each of the nine subsets except that, for the subsets with 20 jobs, the DABC, DABC_D, DABC_T, and HDDE algorithms generate the same ARPD value equal to zero.

While the differences of the DABC algorithm and the other algorithms are quite clear from these tables, it is still necessary to perform some statistical tests on the RPD results in order to observe whether the differences in the ARPD values are indeed statistically significant. Therefore, we employ the 4500 data points for each buffer size and conduct an ANOVA. The one-factor means plots with 95% Least Significant Difference (LSD) confidence intervals of the factor algorithm are shown in Figure 2.

From Figure 2, it can be seen that although there are slight differences in the means plots for different buffer sizes, the same dominance relation between any two algorithms can be obtained. Specifically, the LSD intervals of any two algorithms are not overlapping, so we can conclude that the differences between any two algorithms are statistically significant. The statistical results also show that the DABC_D algorithm is better than the DABC_T algorithm, the DABC_T algorithm is better than the HDDE algorithm, and the HDDE algorithm is better than the IG_RS algorithm.

Further, to illustrate the convergence characteristics of these algorithms, Figures 3–6 illustrate several typical convergence curves of the algorithms, for instance, Ta80. The convergence curves show how the best found total flow time values descended as the CPU time elapses for each algorithm, and they reveal that in general the proposed DABC algorithm obtained a better solution than the DABC_D, DABC_T,

Table 6: Best known solution values for Taillard benchmark set with different buffer sizes.

Instance	Best known solution			
	$B = 1$	$B = 2$	$B = 3$	$B = 4$
20×5				
Ta01	14056	14033	14033	14033
Ta02	15159	15151	15151	15151
Ta03	13407	13301	13301	13301
Ta04	15530	15447	15447	15447
Ta05	13529	13529	13529	13529
Ta06	13329	13123	13123	13123
Ta07	13606	13548	13548	13548
Ta08	13950	13948	13948	13948
Ta09	14325	14295	14295	14295
Ta10	13019	12943	12943	12943
20×10				
Ta11	21035	20955	20911	20911
Ta12	22532	22440	22440	22440
Ta13	19865	19833	19833	19833
Ta14	18758	18710	18710	18710
Ta15	18810	18641	18641	18641
Ta16	19245	19245	19245	19245
Ta17	18470	18363	18363	18363
Ta18	20241	20241	20241	20241
Ta19	20352	20330	20330	20330
Ta20	21335	21320	21320	21320
20×20				
Ta21	33623	33623	33623	33623
Ta22	31675	31588	31587	31587
Ta23	33920	33920	33920	33920
Ta24	31766	31684	31661	31661
Ta25	34557	34557	34557	34557
Ta26	32565	32564	32564	32564
Ta27	32922	32922	32922	32922
Ta28	32467	32412	32412	32412
Ta29	33621	33600	33600	33600
Ta30	32269	32262	32262	32262
50×5				
Ta31	65265	64838	64803	64803
Ta32	68791	68114	68094	68124
Ta33	64066	63365	63162	63242
Ta34	69012	68342	68360	68316
Ta35	70093	69434	69498	69414
Ta36	67815	66874	67009	66851
Ta37	66936	66271	66261	66294
Ta38	65250	64546	64420	64388
Ta39	63528	63018	62981	63047
Ta40	69603	68986	69000	69025
50×10				
Ta41	88433	87407	87345	87286
Ta42	84164	83140	83080	82960

Table 6: Continued.

Instance	Best known solution			
	$B = 1$	$B = 2$	$B = 3$	$B = 4$
Ta43	81658	80159	80147	79931
Ta44	87560	86664	86541	86678
Ta45	87650	86543	86448	86507
50×10				
Ta46	87511	86645	86704	86742
Ta47	89819	89080	88831	89011
Ta48	87774	87191	86822	86727
Ta49	86629	85853	85555	85649
Ta50	89013	88121	87998	87998
50×20				
Ta51	126856	125850	125844	125860
Ta52	120213	119284	119270	119333
Ta53	117415	116483	116653	116459
Ta54	121742	120969	121044	121033
Ta55	119708	118777	118379	118437
Ta56	121573	120638	120783	120870
Ta57	124122	123072	123018	123018
Ta58	123429	122677	122593	122576
Ta59	122997	122090	122130	121872
Ta60	125277	124436	123954	124101
100×5				
Ta61	258193	254676	253887	254083
Ta62	247898	243949	243357	243460
Ta63	243080	238802	238732	238589
Ta64	232350	228779	228394	228200
Ta65	244990	241513	240868	241247
Ta66	238391	233936	233432	233461
Ta67	245320	241630	241148	241078
Ta68	238380	233080	231720	232039
Ta69	253553	249418	248647	248701
Ta70	248872	244979	243684	243754
100×10				
Ta71	306450	300524	300289	299083
Ta72	286565	277464	276113	276321
Ta73	297709	290564	289191	288965
Ta74	312607	304039	303602	303495
Ta75	293860	287153	286124	286086
Ta76	280880	272115	271496	271055
Ta77	290803	282987	280778	281097
Ta78	299396	292674	292305	292774
Ta79	311321	304320	303358	303697
Ta80	300817	293468	292546	292351
100×20				
Ta81	375319	369698	368500	367140
Ta82	384108	375688	374152	374583
Ta83	379817	373071	371560	371677
Ta84	383871	376066	375079	375180
Ta85	377848	371292	370517	370279

Table 6: Continued.

Instance	Best known solution			
	$B = 1$	$B = 2$	$B = 3$	$B = 4$
Ta86	381872	374943	374127	373316
Ta87	386723	376675	375686	374992
Ta88	393530	386499	386371	386411
Ta89	383910	376687	377024	376929
Ta90	389725	382285	380606	380599

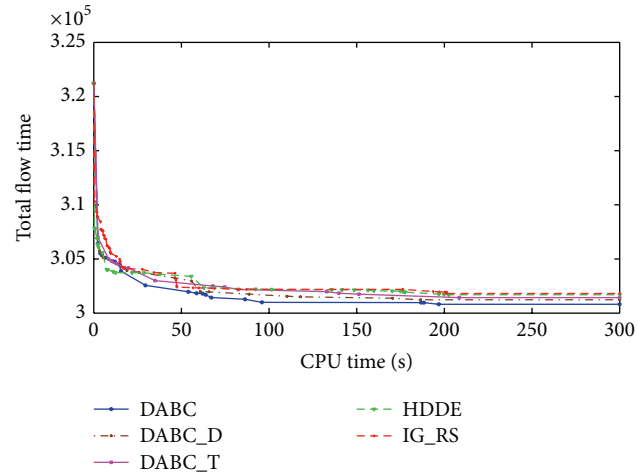

Figure 3: The convergence curves for instance Ta80 ($B = 1$).

HDDE, and IG_RS algorithms and its advantages become more and more impressive as the computational time elapses. After all, the convergence curves validate the superiority of the DABC algorithm over the DABC_D, DABC_T, HDDE, and IG_RS algorithms.

5. Conclusions

This paper proposes a discrete artificial bee colony (DABC) algorithm for solving the permutation flow shop scheduling problem with limited buffers with the total flow time minimization criterion. For solving this problem, the DABC algorithm uses discrete job permutation as food source and introduces the NEH heuristic and its variants to construct the initial population with consideration of both quality and diversity. Moreover, by presenting the best insertion procedure and the combined local search, we present the corresponding improved schemes for the employed bee, onlooker bee, and scout bee phases, respectively. The results of computational experiments and statistical analysis show that the proposed DABC algorithm not only is superior to the existing discrete differential evolution algorithm and iterated greedy algorithm but also performs better than two recently proposed discrete artificial bee colony algorithms. Besides, the DABC algorithm is technically feasible to apply in the practical production environment because of its structural simplicity as well as its high efficacy. In future, we will focus on adapting the DABC algorithm for multiobjective scheduling problems and stochastic scheduling models.

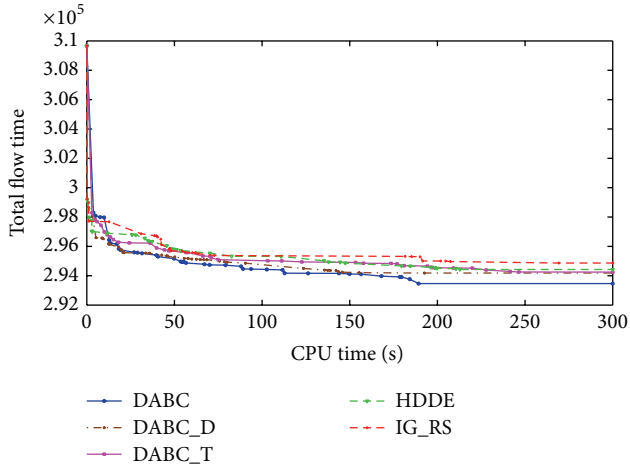

Figure 4: The convergence curves for instance Ta80 ($B = 2$).

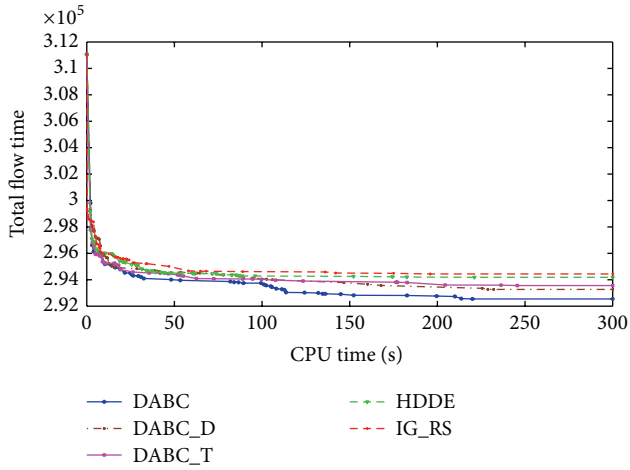

Figure 5: The convergence curves for instance Ta80 ($B = 3$).

Appendix

The best known solution values for all tested instances are given in terms of different buffer sizes in Table 6.

Competing Interests

The authors declare that they have no competing interests.

Acknowledgments

The research was partially supported by National Natural Science Foundation of China (Grant no. 61403180), the Project for Introducing Talents of Ludong University (Grant no. LY2013005), National Natural Science Foundation of China (Grant no. 61273152), the Promotive Research Fund for Excellent Young and Middle-Aged Scientists of Shandong Province (Grant no. BS2015DX018), National Natural Science Foundation of China (Grant no. 51407088), and the Project of Shandong Province Higher Educational Science and Technology Program (Grant no. J14LN20).

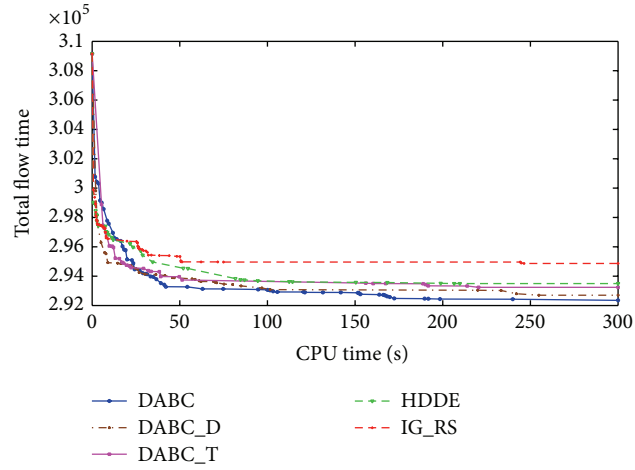

Figure 6: The convergence curves for instance Ta80 ($B = 4$).

References

[1] H. W. Thornton and J. L. Hunsucker, "A new heuristic for minimal makespan in flow shops with multiple processors and no intermediate storage," *European Journal of Operational Research*, vol. 152, no. 1, pp. 96–114, 2004.

[2] 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.

[3] Q.-K. Pan and L. Wang, "Effective heuristics for the blocking flowshop scheduling problem with makespan minimization," *Omega*, vol. 40, no. 2, pp. 218–229, 2012.

[4] N. G. Hall and C. Sriskandarajah, "A survey of machine scheduling problems with blocking and no-wait in process," *Operations Research*, vol. 44, no. 3, pp. 510–525, 1996.

[5] C. H. Papadimitriou and P. C. Kanellakis, "Flowshop scheduling with limited temporary storage," *Journal of the ACM*, vol. 27, no. 3, pp. 533–549, 1980.

[6] S. T. McCormick, M. L. Pinedo, S. Shenker, and B. Wolf, "Sequencing in an assembly line with blocking to minimize cycle time," *Operations Research*, vol. 37, no. 6, pp. 925–935, 1989.

[7] R. Leisten, "Flowshop sequencing problems with limited buffer storage," *International Journal of Production Research*, vol. 28, no. 11, pp. 2085–2100, 1990.

[8] 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.

[9] D. P. Ronconi, "A note on constructive heuristics for the flowshop problem with blocking," *International Journal of Production Economics*, vol. 87, no. 1, pp. 39–48, 2004.

[10] V. Caraffa, S. Ianes, T. P. Bagchi, and C. Sriskandarajah, "Minimizing makespan in a blocking flowshop using genetic algorithms," *International Journal of Production Economics*, vol. 70, no. 2, pp. 101–115, 2001.

[11] J. Grabowski and J. Pempera, "The permutation flow shop problem with blocking. A tabu search approach," *Omega*, vol. 35, no. 3, pp. 302–311, 2007.

[12] 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.

[13] I. Ribas, R. Companys, and X. Tort-Martorell, "A competitive variable neighbourhood search algorithm for the blocking flow shop problem," *European Journal of Industrial Engineering*, vol. 7, no. 6, pp. 729–754, 2013.

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

[15] L. Wang, Q.-K. Pan, and M. F. Tasgetiren, "Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms," *Expert Systems with Applications*, vol. 37, no. 12, pp. 7929–7936, 2010.

[16] G. Deng, Z. Xu, and X. Gu, "A discrete artificial bee colony algorithm for minimizing the total flow time in the blocking flow shop scheduling," *Chinese Journal of Chemical Engineering*, vol. 20, no. 6, pp. 1067–1073, 2012.

[17] C. Smutnicki, "A two-machine permutation flow shop scheduling problem with buffers," *Operations-Research-Spektrum*, vol. 20, no. 4, pp. 229–235, 1998.

[18] E. Nowicki, "The permutation flow shop with buffers: a tabu search approach," *European Journal of Operational Research*, vol. 116, no. 1, pp. 205–219, 1999.

[19] P. Brucker, S. Heitmann, and J. Hurink, "Flow-shop problems with intermediate buffers," *OR Spectrum*, vol. 25, no. 4, pp. 549–574, 2003.

[20] 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.

[21] B. Liu, L. Wang, and Y.-H. Jin, "An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers," *Computers & Operations Research*, vol. 35, no. 9, pp. 2791–2806, 2008.

[22] B. Qian, L. Wang, D. X. Huang, and X. Wang, "An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers," *International Journal of Production Research*, vol. 47, no. 1, pp. 1–24, 2009.

[23] Y.-C. Hsieh, P.-S. You, and C.-D. Liou, "A note of using effective immune based approach for the flow shop scheduling with buffers," *Applied Mathematics and Computation*, vol. 215, no. 5, pp. 1984–1989, 2009.

[24] Q.-K. Pan, L. Wang, and L. Gao, "A chaotic harmony search algorithm for the flow shop scheduling problem with limited buffers," *Applied Soft Computing Journal*, vol. 11, no. 8, pp. 5270–5280, 2011.

[25] Q.-K. Pan, L. Wang, L. Gao, and W. D. Li, "An effective hybrid discrete differential evolution algorithm for the flow shop scheduling with intermediate buffers," *Information Sciences*, vol. 181, no. 3, pp. 668–685, 2011.

[26] F. Q. Zhao, J. X. Tang, J. B. Wang, and J. Jonrinaldi, "An improved particle swarm optimisation with a linearly decreasing disturbance term for flow shop scheduling with limited buffers," *International Journal of Computer Integrated Manufacturing*, vol. 27, no. 5, pp. 488–499, 2014.

[27] G. Moslehi and D. Khorasanian, "A hybrid variable neighborhood search algorithm for solving the limited-buffer permutation flow shop scheduling problem with the makespan criterion," *Computers & Operations Research*, vol. 52, pp. 260–268, 2014.

[28] C. Y. Zhang, P. G. Li, Y. Q. Rao, and Z. L. Guan, "A very fast TS/SA algorithm for the job shop scheduling problem," *Computers & Operations Research*, vol. 35, no. 1, pp. 282–294, 2008.

[29] C. Y. Zhang, P. Li, Z. Guan, and Y. Rao, "A tabu search algorithm with a new neighborhood structure for the job shop scheduling problem," *Computers & Operations Research*, vol. 34, no. 11, pp. 3229–3242, 2007.

[30] 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.

[31] D. Karaboga and B. Basturk, "On the performance of artificial bee colony (ABC) algorithm," *Applied Soft Computing Journal*, vol. 8, no. 1, pp. 687–697, 2008.

[32] N. Karaboga, "A new design method based on artificial bee colony algorithm for digital IIR filters," *Journal of the Franklin Institute*, vol. 346, no. 4, pp. 328–348, 2009.

[33] D. Karaboga and B. Akay, "A comparative study of artificial Bee colony algorithm," *Applied Mathematics and Computation*, vol. 214, no. 1, pp. 108–132, 2009.

[34] Q.-K. Pan, M. F. Tasgetiren, P. N. Suganthan, and T. J. Chua, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," *Information Sciences*, vol. 181, no. 12, pp. 2455–2468, 2011.

[35] M. F. Tasgetiren, Q.-K. Pan, P. N. Suganthan, and A. H.-L. Chen, "A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops," *Information Sciences*, vol. 181, no. 16, pp. 3459–3475, 2011.

[36] R. Ruiz and T. Stützle, "A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 177, no. 3, pp. 2033–2049, 2007.

[37] J. M. Framinan and R. Leisten, "Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm," *International Journal of Production Research*, vol. 46, no. 22, pp. 6479–6498, 2008.

[38] E. Vallada and R. Ruiz, "Cooperative metaheuristics for the permutation flowshop scheduling problem," *European Journal of Operational Research*, vol. 193, no. 2, pp. 365–376, 2009.

[39] Q.-K. Pan, M. F. Tasgetiren, and Y.-C. Liang, "A discrete differential evolution algorithm for the permutation flowshop scheduling problem," *Computers and Industrial Engineering*, vol. 55, no. 4, pp. 795–816, 2008.

[40] G. Deng and X. Gu, "A hybrid discrete differential evolution algorithm for the no-idle permutation flow shop scheduling problem with makespan criterion," *Computers & Operations Research*, vol. 39, no. 9, pp. 2152–2160, 2012.

[41] Q.-K. Pan and R. Ruiz, "An effective iterated greedy algorithm for the mixed no-idle permutation flowshop scheduling problem," *Omega*, vol. 44, pp. 41–50, 2014.

[42] D. C. Montgomery, *Design and Analysis of Experiments*, Wiley, New York, NY, USA, 8th edition, 2012.