A Modified Cuckoo Search Algorithm for Flow Shop Scheduling Problem with Blocking

Hui Wang[1], Wenjun Wang[2], Hui Sun[1], Changhe Li[3]
[1]School of Information Engineering, Nanchang Institute of Technology, Nanchang 330099, China
huiwang@whu.edu.cn, sun_hui2006@163.com
[2]School of Business Administration,Nanchang Institute of Technology, Nanchang 330099, China
[3]School of Computer, China University of Geosciences, Wuhan 430072, China
changhe.lw@gmail.com

Shahryar Rahnamayan[4], Yong Liu[5]
[4]Department of Electrical, Computer, and Software Engineering, University of Ontario Institute of Technology (UOIT), 2000 Simcoe Street North, Oshawa, ON L1H 7K4, Canada
shahryar.rahnamayan@uoit.ca
[5]University of Aizu, Tsuruga, Ikki-machi, Aizu-Wakamatsu, Fukushima 965-8580, Japan
yliu@u-aizu.ac.jp

***Abstract*—This paper presents a Modified Cuckoo Search (MCS) algorithm for solving flow shop scheduling problem with blocking to minimize the makespan. To handle the discrete variables of the job scheduling problem, the smallest position value (SPV) rule is used to convert continuous solutions into discrete job permutations. The Nawaz-Enscore-Ham (NEH) heuristic method is utilized for generating high quality initial solutions. Moreover, two frequently used swap and insert operators are employed for enhancing the local search. To verify the performance of the proposed MCS algorithm, experiments are conducted on Taillard's benchmark set. Results show that MCS performs better than the standard CS and some previous algorithms proposed in the literature.**

Keywords—cuckoo search; flow shop scheduling; blocking; makespan; optimization

I. Introduction

Production scheduling plays an important role in any manufacturing system. With the rapid development of market, good scheduling technologies can greatly improve the production efficiency. In order to achieve a good position in the market competition, more effective scheduling methods are always needed. The flow shop scheduling problem with blocking is one of the most popular scheduling problems, which widely exists in the production system where processed jobs are sometimes kept in the machines because of lacking intermediate buffer storage [1], or storage is not allowed in some stages of the manufacturing process because of technological requirements [2–3]. Since there are no buffers between machines, intermediate queues of jobs waiting in the system for their subsequent operations are not allowed. A job completed on a machine blocks this machine until the next machine is available for processing. No job can surpass another, because there is no buffer. It has been proved that the flow shop scheduling problem with blocking is NP-hard, when there are more than two machines [4].

In the past decades, the research on flow shop scheduling problem with blocking has attracted much attention; and different approaches have been proposed. McCormich et al. [5] proposed a profile fitting method for the sequencing problems in an assembly line with blocking to minimize cycle time. Abadi et al. [6] used a heuristic approach to minimize the cycle time in a blocking flow shop. Ronconi [7] developed a branch-and-bound algorithm to minimize the makespan in a flow shop with blocking. Grabowski and Pempera [3] developed two tabu search methods, called TS and TS with multi-moves (TS+M), respectively. The TS+M uses the multi-moves that consist in performing several moves simultaneously in a single iteration and guide the search process to more promising areas of the solutions space. It allows the algorithm to achieve very good solutions in a much shorter time.

Recently, some bio-inspired algorithms have been proposed to solve this problem. Wang et al. [8] designed a novel hybrid discrete differential evolution (HDDE) algorithm for blocking flow shop scheduling problem. Individuals are represented as discrete job permutations, and new mutation and crossover operators are developed for this representation. So, HDDE can directly work in the discrete domain. A local search based on insert neighborhood structure is utilized to balance the exploration and exploitation. Experimental results show that the HDDE outperforms TS and TS+M. In [9], three hybrid harmony search (HS) algorithms are used to solve the flow shop scheduling with blocking to minimize the total flow time. Unlike HDDE, it employs a largest position value (LPV) rule to convert continuous harmony vectors into job permutations. A new population initialization method based on a variant of NEH heuristic is used. Liang et al. [2] presented a dynamic multi-swam particle swarm optimizer (DMS-PSO) for solving flow shop scheduling problem with blocking. To maintain good global search ability, small swarms and a regrouping scheduling were used. A specially designed local search phase was employed to improve the local search ability. Computational results show that DMS-PSO achieves a better performance than some other compared algorithms. Han et al. [10] proposed an improved artificial bee colony (IABC) algorithm, which utilized discrete job permutations to represent solutions and employed insert and swap operators to generate new candidate solutions for the employed and onlooker bees. The differential evolution algorithm is used to obtain solutions for the scout bees. Wang and Tang [11] designed a discrete particle swarm optimization (DPSO) algorithm for the blocking flow shop scheduling problem. To prevent the DPSO from premature convergence, a self-adaptive diversity control strategy is adopted.

Cuckoo search (CS) algorithm is a recently proposed optimization approach developed by Yang and Deb [12], which is inspired by the obligate brood parasitism of some cuckoo species by laying their eggs in the nests of other host birds. Preliminary studies show that CS outperforms some existing algorithms such as genetic algorithm (GA) and particle swarm optimization (PSO) [13]. Although the CS algorithm has been applied to solve flow shop scheduling problem, these research works focus on the permutation or hybrid flow shop scheduling problems [14–16]. In [14], Marichelvam proposed an improved hybrid cuckoo search (IHCS) algorithm for the permutation flow shop scheduling problem. The IHCS also used NEH heuristic to generate initial population. Results show that the IHCS performs better than an ant colony optimization meta-heuristic (MHD-ACS). In [15], Li and Yin presented a hybrid CS (HCS) to solve the permutation flow shop scheduling problem. To enhance the local exploitation ability of CS, a fast local search operator is used. Simulation results show the effectiveness of the HCS. In [16], Marichelvam et al. proposed an improved CS (ICS) algorithm based on the NEH heuristic to solve hybrid flow shop scheduling problem.

In this paper, we propose a modified CS (MCS) algorithm to solve the flow shop scheduling problem with blocking. In MCS, a smallest position value (SPV) rule is used to convert continuous solutions into discrete job permutations. To improve the quality of the population initialization, the NEH method is used. Moreover, two local search operators: swap and insertion, are employed to enhance the local search ability.

The rest of the paper is organized as follows. The problem description of the blocking flow shop scheduling is presented in Section 2. The standard CS algorithm is given in Section 3. The modified CS algorithm is proposed in Section 4. Experimental results are given in Section 5. Finally, the work is concluded and summarized in Section 6.

II. Flow Shop Scheduling Problem with Blocking

The flow shop scheduling problem with blocking can be described as follows. There are a set of n jobs, $J=\{1,2,\ldots,n\}$, and a set of m machines, $M=\{1,2,\ldots,m\}$. Each job j ($j\in J$) will be sequentially processed on machine 1, machine 2, and so on until machine m. The processing time of job j ($j\in J$) on machine k ($k\in M$) is denoted as $p(j, k)$. The processing of each job cannot be interrupted, and the setup time is included into the processing time. At any time, each machine can process at most one job, and each job can be processed on at most one machine. The sequence in which the jobs are to be processed is the same for each machine. There is no intermediate buffers between any two consecutive machines, a job j cannot leave a machine k until its next machine k+1 is available. That means the job j is blocked on machine k if the next machine k+1 is not free. The objective of the problem is to find a job sequence for processing all jobs on all machines so as to minimize the makespan [2, 8].

Let $\pi=[\pi_1,\pi_2,...,\pi_n]$ be a job permutation, where π_j is the jth job of π. $e(j, k)$ is the departure time of job π_j on machine k. The $e(j, k)$ is defined as follows [7].

$$e(1,0)=0 \tag{1}$$

$$e(1,k)=e(1,k-1)+p(1,k),\ k=1,2,...,m-1 \tag{2}$$

$$e(j,0)=e(j-1,1),\ j=1,2,...,n \tag{3}$$

$$e(j,k)=\max\{e(j,k-1)+p(j,k),e(j-1,k+1)\} \tag{4}$$

$$j=2,...,n,\ k=1,2,...,m-1$$

$$e(j,m)=e(j,m-1)+p(j,m),\ j=1,2,...,n \tag{5}$$

where $e(j,0)$, j=1,2,…,n, represents the starting time of job π_j on the first machine. In the above recursive equations, the departure times of the first job on each machine are calculated first, then the second job, until the last job [7]. The makespan $C_{\max}(\pi)$ of the job permutation π is given by [2]:

$$C_{\max}(\pi)=e(n,m) \tag{6}$$

Thus, the objective of this paper is to find a job permutation π to minimize the $C_{\max}(\pi)$:

$$\min\{C_{\max}(\pi)\} \tag{7}$$

III. Cuckoo Search Algorithm

Cuckoo search (CS) is a new population-based stochastic search algorithm. It is inspired by the obligate brood parasitic behavior of some cuckoo species by laying their eggs in the nests of other host birds. To simplify the CS algorithm, three idealized rules are used as follows [12].

- Each cuckoo lays one egg at a time, and dumps it in a randomly chosen nest;
- The best nests with high quality of eggs (solutions) will carry over to the next generations;
- The number of available host nests is fixed, and a host can discover an alien egg with a probability $p_a\in[0, 1]$. In this case, the host bird can either throw the egg away or abandon the nest so as to build a completely new nest in a new location.

Based on these three rules, the basic steps of the CS algorithm can be summarized in Algorithm 1, where N is the population size, f is the fitness function, $p_a\in[0,1]$ is the probability of discovering an alien egg, t is the generation index, and *MaxGen* is the maximum number of generations.

In the CS, new solutions X_i for the ith cuckoo are generated by the following Lévy flight [12].

$$X_i(t+1)=X_i(t)+\alpha\oplus L\acute{e}vy(\lambda) \tag{8}$$

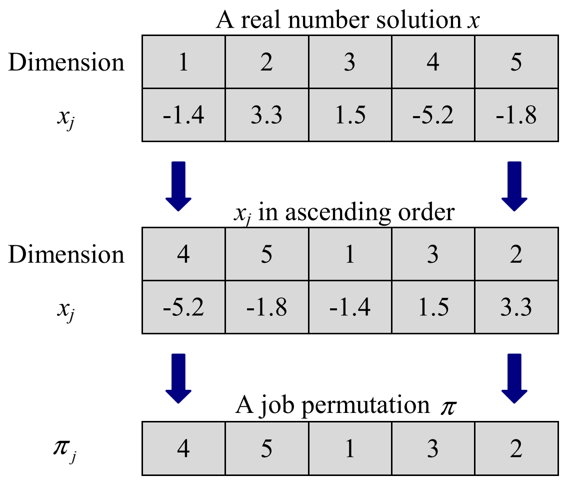
where $\alpha>0$ is the step size which should be related to the scales of the problem of interest. The product $\oplus$ means entrywise multiplications. The Lévy flight is a random walk, in which the step length is determined by Lévy distribution [12].

$$Le'vy \sim u = t^{-\lambda},\ (1 < \lambda \leq 3) \tag{9}$$

It is known that the Lévy distribution has an infinite variance with an infinite mean. Therefore, the consecutive jumps of a cuckoo form a random walk process which obeys a power-length distribution with a heavy tail [13].

Algorithm 1: The Standard CS

```
Initialize a population of $N$ host nests;
Calculate the fitness value of each initial solution;
while $t$<=$MaxGen$ do
    Get a cuckoo (say $i$) randomly by Lévy flights;
    Evaluate the fitness value of $f_i$;
    Randomly choose a nest among $N$ (say $j$);
    if $f_i$ is better than $f_j$ then
        Replace $j$ with new solution;
    end if
    Abandon a fraction ($p_a$) of worse nests, and build ones via Lévy flights;
    Keep the best solutions;
    Rank the solutions and find the current best;
    $t$++;
end while
```


A real number solution x

Dimension	1	2	3	4	5
x_j	-1.4	3.3	1.5	-5.2	-1.8

x_j in ascending order

Dimension	4	5	1	3	2
x_j	-5.2	-1.8	-1.4	1.5	3.3

A job permutation π

π_j	4	5	1	3	2

Fig. 1. An example for the SPV rule.

IV. The Proposed MCS Algorithm for Flow Shop Scheduling Problem with Blocking

In this section, we present a modified CS (MCS) algorithm for the flow shop scheduling problem with blocking. The detailed descriptions of the MCS are given as follows.

A. Solution Representation

The original CS algorithm was usually used to solve benchmark or real-world problems over continuous search space. However, the flow shop scheduling problem is a discrete problem. To handle the discrete variables, a smallest position value (SPV) rule is utilized [17]. The SPV is a simple method, which has been successfully applied to various production scheduling problems. Another popular solution representation method called largest position value (LPV) [2] is similar to the SPV rule.

Let each index of the dimensions of a real number solution represent a typical job from $J=\{1,2,\ldots,n\}$, and then n indexes denote n different jobs. Assume that $X=\{x_1,x_2,\ldots,x_n\}$ is a real number solution. By sorting the position values of X in ascending order, a job permutation π is obtained. Fig. 1 illustrates a simple example of the SPV rule.

B. Population Initialization

Population initialization is an important step for stochastic search algorithms. The Nawaz-Enscore-Ham (NEH) [18] heuristic is usually used for flow shop scheduling problems [2, 8–11, 14–16]. The main idea of the NEH heuristic is that the high processing time on all machines should be scheduled as early in the sequence as possible. The NEH heuristic can be described as follows [18].

- Step 1. Compute the total processing time of each job on all m machines. Sort the jobs according to the total processing time in non-increasing order. Then, we obtain the sequence $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$.
- Step 2. The first two jobs of π are taken and the two partial possible sequences of these two jobs are evaluated. Then, the better partial sequence is selected as the current sequence.
- Step 3. Take the job π_j, j=3,4,…,n, and find the best partial sequence by inserting it in all possible positions of the partial sequence of jobs that have been already scheduled. The best sequence would be selected for the next iteration.

Based on the above three steps, we can get a good job permutation. The CS algorithm works on continuous search space. So, the obtained job permutation should be converted to a real number solution. Let $\pi = [\pi_1, \pi_2, \ldots, \pi_n]$ be a job permutation, and $X=\{x_1,x_2,\ldots,x_n\}$ be a real number solution in the population. Then, we use a simple method to convert the π to X as follows.

$$x_{\pi_j} = \frac{(x_{\max} - x_{\min})}{n} \cdot j - x_{\max},\ j = 1,2,\ldots,n \tag{10}$$

where n is the number of jobs, $x_{\max}$ =1, and $x_{\min}$ =-1. It can be seen that x_1 obtains the smallest value, while x_n obtains the largest value. According to the SPV rule, the dimension index of the smallest value achieves the first job. Therefore, the job permutation can be converted to a real number vector, and a real number vector can also be converted to a job permutation.

The NEH heuristic can only generate one job permutation. Based on this permutation, we can only get one solution. Thus, we employ a modified NEH method to generate multiple initial solutions [2]. In the new NEH method, the initial job sequence in Step 1 is generated randomly, and then Step 2 and Step 3 are sequentially used to improve the seed sequence. Therefore, we can obtain multiple similar job permutations with high quality.

The main steps of the modified NEH method for population initialization can be described as follows [2].

- Step 1. Generate a job permutation π according to the NEH heuristic. The rest N–1 job permutation are generated by the modified NEH heuristic. Compute the makespan of each job permutation.
- Step 2. According to Eq. 10, convert the all N job permutation to N real number solutions. The fitness value of each solution is equal to its makespan.
- Step 3. Find the best solution in the initial population and set it as X_{best}.

Algorithm 2: The Swap Operator

```
Randomly select a position $\pi_j$ from a job $\pi$;
$l$=1;
while $l<=n$ do
  if $i \neq j$ then
    Set $\pi^* = \pi$;
    Swap $\pi^*_j$ with $\pi^*_l$;
    Select the better one between $\pi$ and $\pi^*$ as the new $\pi$;
  end if
  $l$++;
end while
```

Algorithm 3: The Insert Operator

```
$j$=1;
while $j<=n$ do
  Set $\pi^* = \pi$;
  Remove $\pi^*_j$ from $\pi$ and get a sub sequence $\pi^*$;
  Inert into all possible positions of the sub sequence, and
  calculate the makespan of all $n$ new job permutations;
  Compare the $n$ new job permutations with $\pi$, and the
  best one is selected as the new $\pi$;
  $j$++;
end while
```

C. Local Search

It has been proved that utilizing local search is helpful for solving flow shop scheduling problems [8]. In the past several years, different kinds of local search operators have been proposed, such as insert, interchange, inverse, and swap. Previous studies pointed out that the insert operator is more efficient than interchange [19]. In this paper, two local search operators, insert and swap, are employed to enhance the exploitation ability [20–21]. The main steps of the swap and insert operators are described in Algorithms 2 and 3, respectively.

Algorithm 4: The MCS Algorithm

```
Initialize population based on the modified NEH method;
while $t<=MaxGen$ do
  Get a cuckoo ($X_i$) randomly by Lévy flights;
  Apply the SPV rule to convert $X_i$ to a job permutation;
  Calculate the makespan of the new job permutation;
  Randomly choose a nest among $N$ ($X_j$);
  if $f_i$ is better than $f_j$ then
    Replace $X_j$ with $X_i$;
  end if
  Abandon a fraction ($p_a$) of worse nests, and build ones
  via Lévy flights;
  Apply the SPV rule to convert the new solution to job
  permutations, and calculate their makespan;
  Update the best solutions;
  for $i$=1; $i<=N$; $i$++ do
    if $rand(0,1)<=p_s$ then
      Execute the swap operator (Algorithm 2);
    end if
  end for
  for $i$=1; $i<=N$; $i$++ do
    if $rand(0,1)<=p_i$ then
      Execute the insert operator (Algorithm3);
    end if
  end for
  Update the best solutions;
  $t$++;
end while
```

D. Framework of MCS

The main steps of our MCS algorithm are described in Algorithm 4, where *rand*(0,1) is a random number between 0 and 1, p_s and p_i is the probability rate of conducting swap and insert operators, respectively.

V. Simulation Results

A. Test Problems

In order to verify the performance of our proposed MCS algorithm, experiments are conducted on Taillard's benchmark set [22]. In this paper, there are 110 test instances (Ta001-Ta110) with the size from 20 jobs and 5 machines (20×5) to 200 jobs and 20 machines (200×20).

The parameter settings are described as follows. For the standard CS and MCS, the population size N and *MaxGen* are set to 50 and 500, respectively. The parameter p_a is set to 0.25. For the probabilities of swap and insert, both p_s and p_i are set to 0.1. For each algorithm, each test instance is independently run ten times, and the percentage relative difference (PRD) is calculated as follows [2]:

$$\mathrm{PRD} = \frac{100 \times (C^{Ron} - C^{A})}{C^{Ron}} \qquad (11)$$

where C^{Ron} is the makespan provided by Ronconi [7], and C^{A} is the makespan achieved by our MCS algorithm or other compared algorithms. The PRD can measure the performance of an algorithm. A larger PRD means that the algorithm is better. For each kind of problem size, there are ten different test instances. The average percentage relative difference (APRD), maximum PRD (MaxPRD), and minimum PRD (MinPRD) are reported.

All algorithms are encoded in VC++ 6.0 and independently run on an Intel Core i7-4510U CPU 2.60 GHz with 8.0 GB Memory in the Windows 7 Operating System.

B. *Comparison between CS and MCS*

In this section, we compare the performance of standard CS and the proposed MCS on the test bed. Table I presents the computational results achieved by CS and MCS. As seen, MCS achieves much better results than the standard CS in terms of the solution accuracy. For all test instances, the PRD values obtained by the standard CS is negative. That is to say the performance of the standard CS is worse than Ronconi's branch-and-bound approach [7]. However, MCS needs a higher computational time than the standard CS. Especially for large size test instances, the running of MCS is very time consuming. Fig. 2 lists the convergence plots of CS and MCS on two test instances. It is obvious that MCS converges faster than CS during the search process.

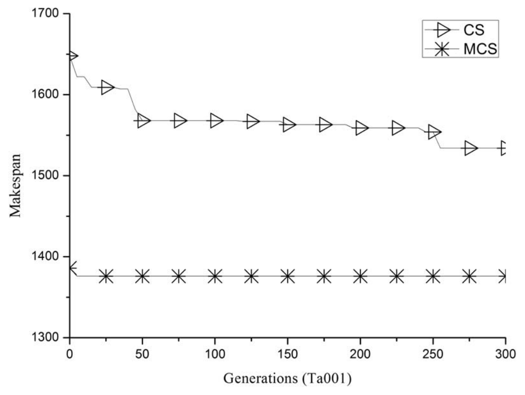
From the above analysis, though MCS achieves better solutions than the standard CS, CS cost less computational time. The main reason is that both of the two algorithms use the same *MaxGen* as the stopping criterion. To further compare the performance of CS and MCS, the running time is used as the stopping criterion. In the experiment, the standard CS is given more computational time than MCS. This is helpful to investigate the effectiveness of the employed local search and NEH based population initialization in MCS.

TABLE I. RESULTS ACHIEVED BY CS AND MCS

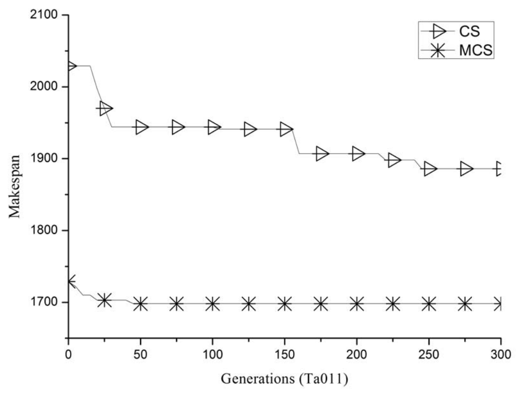
n×*m*	Standard CS				MCS			
	APRD	MaxPRD	MinPRD	Time (s)	APRD	MaxPRD	MinPRD	Time (s)
20×5	–12.61	–10.78	–13.72	0.28	0.34	0.37	0.078	1.61
20×10	–8.6	–7.45	–9.74	0.3	2.39	3.39	1.72	1.97
20×20	–4.53	–3.47	–5.48	0.32	3.39	3.93	2.98	2.78
50×5	–17.22	–16.34	–17.82	0.71	3.05	3.54	2.63	11.65
50×10	–11.02	–10.2	–11.73	0.75	4.52	5.04	4.12	17.4
50×20	–7.94	–7.36	–8.47	0.82	4.72	5.12	4.36	29.65
100×5	–22.29	–21.59	–22.79	1.58	1.02	1.33	0.61	64.58
100×10	–12.92	–12.39	–13.32	1.68	4.38	4.67	4.01	103.9
100×20	–9.01	–8.66	–9.31	1.82	3.98	4.07	3.54	213.7
200×10	–15.22	–14.89	–15.5	4.23	2.14	2.22	2.01	771.2
200×20	–9.71	–9.29	–10.02	4.51	2.89	2.95	2.82	1860.8
Average	–11.92	–11.13	–12.54	1.55	2.98	3.39	2.63	279.93

TABLE II. RESULTS ACHIEVED BY CS AND MCS WHEN CS COST MORE COMPUTATIONAL TIME

n×*m*	Standard CS		MCS	
	APRD	Time (s)	APRD	Time (s)
20×5	–9.49	2.0	0.34	1.61
20×10	–5.81	2.0	2.39	1.97
20×20	–2.08	3.0	3.39	2.78
50×5	–14.86	12.0	3.05	11.65
50×10	–8.89	18.0	4.52	17.4
50×20	–5.18	30.0	4.72	29.65
Average	–7.72	11.17	3.07	10.84



(a) Ta001



(b) Ta011

Fig. 2. The convergence characteristics between CS and MCS on Ta001 and Ta011.

TABLE III. RESULTS ACHIEVED BY TS, TS+M, HDE, DMS-PSO, AND MCS

n×*m*	TS	TS+M	HDE	DMS-PSO	MCS
	APRD	APRD	APRD	APRD	APRD
20×5	–1.64	–0.34	0.26	0.30	**0.34**
20×10	1.45	1.76	2.30	2.32	**2.39**
20×20	2.88	2.94	3.25	3.26	**3.39**
50×5	–0.55	0.55	**3.09**	2.78	3.05
50×10	1.98	3.52	**4.57**	4.09	4.52
50×20	3.68	4.26	**5.00**	4.91	4.72
100×5	–3.03	–2.62	–0.32	1.01	**1.02**
100×10	1.71	2.66	3.40	**4.43**	4.38
100×20	2.01	3.03	3.05	**4.72**	3.98
200×10	–0.60	0.58	0.33	1.93	**2.14**
200×20	1.24	2.31	1.36	**2.94**	2.89
Average	0.83	1.70	2.39	2.97	**2.98**

Table II presents the results achieved by standard CS and MCS when we assign the standard CS slightly more computational time than MCS. As shown, even though the standard CS runs for more time than MCS, it still achieves negative APRD values on all group instances. Compared to the results shown in Table 1, CS with more computational time can find better solutions. However, MCS is still much better than the standard CS. It also demonstrates that the employed local search and NEH based population strategies are helpful for MCS to solve flow shop scheduling problem with blocking.

C. Comparison of MCS, TS/TS+M, HDE, and DMS-PSO

This section presents a comparison of MCS with other existing meta-heuristics. The involved algorithms are listed as follows.

- TS and TS+M [3].
- Hybrid DE-based algorithm (HDE) [23].
- DMS-PSO [2].
- The proposed MCS.

Table III lists the results achieved by TS, TS+M, HDE, DMS-PSO, and MCS. The best results for each problem size are shown in bold. Results of TS, TS+M, HDE, and DMS-PSO are taken from Table I in the literature [2]. The parameter setting of these algorithms can be found in [2].

From Table III, it can be seen that the proposed MCS achieves best performance in terms of the overall solution quality, because it obtains the largest overall APRD (2.98%), which is slightly better than DMS-PSO (2.97%), and much better than TS (0.83%), TS+M (1.70%), and HDE (2.39%). MCS outperforms HDE on 8 out of 11 group instances. Compared to DMS-PSO, MCS performs better on 7 group instances, while DMS-PSO is better than MCS on the rest of 4 group instances. MCS achieves much larger APRD than both TS and TS+M on all 11 group instances.

Although MCS achieves better results than TS, TS+M, HDE, and DMS-PSO, it cost much more computational time. For HDE and DMS-PSO, they use the maximum computational time (T_{max}=5mn milliseconds) as the stopping criterion. In the current version of MCS, the swap and insert operators are conducted on all individuals with a probability 0.1. This is the main reason that MCS has a high computational cost for 300 iterations (*MaxGen*=300). To overcome this problem, the swap and insert operators can be conducted on the global best individual. This will save much computational time.

VI. Conclusions

In this paper, we propose a modified CS (MCS) algorithm to solve the flow shop scheduling problem with blocking. The MCS algorithm employs three strategies: the SPV rule, a variant of NEH-based population initialization, and two local search operators. To verify the performance of the proposed MCS, experiments are conducted on Taillard's benchmark set. Results show that MCS performs better than the standard CS, TS, TS+M, HDE, and DMS-PSO in terms of the overall solution quality.

Compared to the standard CS, the MCS achieves much better solutions, but its computational time is higher under the same iterations. When the standard CS runs more time than MCS, it cannot obtain better results than MCS. This confirms the effectiveness of the employed strategies in MCS.

As a main drawback, the proposed MCS is time consuming. The primary reason is that the swap and insert operators are conducted on all individuals with a probability rate. To overcome this problem, the swap and insert operators will be conducted on the global best individual to save computational time. Moreover, the opposition-based learning [24–26] may be helpful to accelerate the MCS. These will be investigated in the future work.

Acknowledgment

This work is supported by the Humanity and Social Science Foundation of Ministry of Education of China (No. 13YJCZH174), the National Natural Science Foundation of China (Nos. 61305150 and 61261039), the Science and Technology Plan Project of Jiangxi Provincial Education Department (Nos. GJJ14747 and GJJ13762), and the Natural Science Foundation of Jiangxi Province (No. 20142BAB217020).

References

[1] J. Grabowski and J. Pempera, "Sequencing of jobs in some production system," European Journal of Operational Research, vol. 125, pp. 535-550, 2000.

[2] J.J. Liang, Q.K. Pan, T.J. Chen, and L. Wang, "Solving the blocking flow shop scheduling problem by a dynamic multi-swarm particle swarm optimizer," The International Journal of Advanced Manufacturing Technology, vol. 55, pp. 755-762, 2011.

[3] J. Grabowski and J. Pempera, "The permutation flow shop problem with blocking. A tabu search approach," OMEGA, The international Journal of Management Science, vol. 35, pp. 302–311, 2007.

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

[5] S.T. McCormich, M.T. Pinedo, and S. Shenker, B. Wolf, "Sequencing in an assembly line with blocking to minimize cycle time," Operational Research, vol. 37, pp. 925–936, 1989.

[6] I.N.K. Abadi, N.G. Hall, and C. Sriskandarajh, "Minimizing cycle time in a blocking flowshop," Operational Research, vol. 48, pp. 77–180, 2000.

[7] D.P. Ronconi, "A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking," Annals of Operations Research, vol. 138, no. 1, pp. 53–65, 2005.

[8] 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 & OperationsResearch, vol. 37, pp. 509–520, 2010.

[9] 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.

[10] 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," The International Journal of Advanced Manufacturing Technology, vol. 60, pp. 1149–1159, 2012.

[11] X.P. Wang and L.X. Tang, "A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking," Applied Soft Computing, vol. 12, pp. 652–662, 2012.

[12] X.S. Yang and S. Deb, "Cuckoo Search Via Lévy Flights," Proceeding of World Congress on Nature & Biologically Inspired Computing (NaBIC 2009), December 2009, pp. 210–214.

[13] X.S. Yang and S. Deb, "Engineering Optimisation by Cuckoo Search," International Journal of Mathematical Modelling and Numerical Optimisation, vol. 1, no. 4, pp. 330–343, 2010.

[14] M.K. Marichelvam, "An improved hybrid Cuckoo Search (IHCS) metaheuristics algorithm for permutation flow shop scheduling problems," International Journal of Bio-Inspired Computation, vol. 4, no. 4, pp. 200–205, 2012.

[15] X.T Li and M.H Yin, "A hybrid cuckoo search via Lévy flights for the permutation flow shop scheduling problem," International Journal of Production Research, vol. 51, no. 16, pp. 4732–4754, 2013.

[16] M.K. Marichelvam, T. Prabaharan, and X.S. Yang, "Improved cuckoo search algorithm for hybrid flow shop scheduling problems to minimize makespan," Applied Soft Computing, vol. 19, pp.93–101, 2014.

[17] M.F. Tasgetiren, Y.C. Liang, M. Sevkli, and G. Gencyilmaz, "Particle swarm optimization and differential evolution for the single machine total weighted tardiness problem," International Journal of Production Research, vol. 44, no. 22, pp. 4737–4754, 2006.

[18] M. Nawaz, E.E.J. Enscore, and I. Ham, "A heuristic algorithm for the m-machine, n-job flow shop sequencing problem," OMEGA, The international Journal of Management Science, vol. 11, no. 1, pp. 91–95, 1983.

[19] E. Nowicki and C. Smutnicki, "A fast tabu search algorithm for the permutation flowshop problem," European Journal of Operational Research, vol. 91, pp. 160–75, 1996.

[20] R.Ruben and T. Stutzle, "An Iterated Greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives," European Journal of Operational Research, vol. 187, pp. 1143–1159, 2008.

[21] 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, pp. 795–816, 2008.

[22] E. Taillard, "Some efficient heuristic methods for the flow shop sequencing problem," European Journal of Operational Research, vol. 47, no. 1, pp. 65–74, 1990.

[23] B. Qian, L. Wang, D.X. Huang, W.L. Wang, and X. Wang, "An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers," Computers & Operations Research, vol. 36, no. 1, pp. 209–233, 2009.

[24] S. Rahnamayan, H.R. Tizhoosh, and M.A.Salama, "Opposition-based differential evolution', IEEE Transactions on Evolutionary Computation, vol. 12, no. 1, pp. 64–79, 2008.

[25] H. Wang, Z.J. Wu, S. Rahnamayan, Y. Liu, and M. Ventresca, "Enhancing particle swarm optimization using generalized opposition-based learning," Information Sciences, vol. 181, no. 20, pp. 4699–4714, 2011.

[26] H. Wang, Z.J. Wu, S. Rahnamayan, "Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems," Soft Computing, vol. 15, no. 11, pp. 2127–2140, 2011.