Migrating Birds Optimization for Lot-streaming flow shop scheduling problem

Yuyan Han
School of Computer Science
Liaocheng University
Liaocheng, China
hanyuyan@lcu-cs.com

Junqing Li
School of Computer Science
Liaocheng University
Liaocheng, China
lijunqing@lcu-cs.com

Yiping Liu
Department of Computer Science and Intelligent Systems, Graduate School of Engineering, Osaka Prefecture University
yiping0liu@gmail.com

Zhixin Zheng
School of Computer Science
Liaocheng University
Liaocheng, China
zhengzhixin@lcu-cs.com

Yuxia Pan
School of Information Intelligence Engineeringe
University of Sanya
panyuxia2008@163.com

Hongyan Sang
School of Computer Science
Liaocheng University
Liaocheng, China
sanghongyan@lcu-cs.com

Lili Liu
School of Computer Science
Liaocheng University
Liaocheng, China
liulili@lcu-cs.com

***Abstract*—This paper presents a novel migrating birds optimization (NMBO) algorithm for solving the lot-streaming flowshop scheduling problem with minimizing makespan. The proposed NMBO algorithm utilizes discrete job permutations to represent solutions, and applies multiple neighborhoods based on insert and swap operators to improve the leading solution. Two new crossover operators, i.e., similar job order with artificial chromosome crossover, and similar block order crossover are employed to obtain solutions for the rest migrating birds. An initialization scheme based on the problem-specific heuristics is presented to generate an initial population with a certain level of quality and diversity. A local search based on the insert neighborhood is embedded to improve the algorithm's local exploitation ability. NMBO is compared with the existing discrete invasive weed optimization, estimation of distribution algorithm and modified MBO algorithms based on the well-known lot-streaming flow shop benchmark. The computational results and comparison demonstrate the superiority of the proposed NMBO algorithm for the lot-streaming flow shop scheduling problems with makespan criterion.**

Keywords*—*migrating birds optimization, lot-streaming, scheduling, multiple neighborhoods

I. INTRODUCTION

Lot-steaming is a production layout in which every job can be split into a number of smaller sub-lots. When a sub-lot is completed, it can be immediately transferred to the downstream machine. By this splitting technique, the idle time on successive machines can be reduced, and thereby reducing productive cycle, accelerating the manufacturing process and enhancing the production efficiency [1].

With respect to the lot-streaming technique, the concept of lot-steaming was introduced into flow shop and job shop scheduling problems and efforts of some significant research of meta-heuristics have been made to optimize the lot-streaming scheduling problems. For the lot-streaming problem in flow shop environment, Pan et al. proposed a discrete artificial bee colony (DABC) algorithm to solve a lot-streaming flow shop scheduling problem (LSFSP) with the objective of minimizing the maximum completion time,i.e., makespan [2]. In their work, an adaptive strategy based on insert and swap operators and a path re-linking strategy were utilized to generate neighboring food sources. Sang et al. employed an effective discrete invasive weed optimization (DIWO) algorithm to solve LSFSP with sequence-dependent setup times, in which a local search procedure based on the insertion neighborhood is employed to perform local exploitation [3]. Pan et al. presented a novel estimation of distribution algorithm (EDA) to minimize the maximum completion time, in which an estimation of a probabilistic model is constructed to guide the algorithm search towards good solutions by taking into account both job permutation and similar blocks of jobs [4].

Very recently, a new metaheuristic intelligence approach named the Migrating Birds Optimization (MBO) algorithm was presented by Duman [5], which simulates the V flight formation of migrating birds. For the scheduling problems, Tongur and Ülker first applied the basic MBO algorithm to optimize the discrete flow shop sequencing problem [6]. Following that Pan and Dong designed an improved MBO (IMBO) algorithm to minimize the total flow time of the hybrid flow shop scheduling [7]. In this work, the authors presented a diversified method to initialize population with high quality, and constructed a mixed neighbourhood based on insertion and pairwise exchange operators to generate promising neighboring solutions for the leader and the following birds. Similarly, Niroomand et al. also proposed modified MBO (MMBO) algorithm to optimize closed loop layout with exact distances in flexible manufacturing systems, which are different from IMBO considered by Pan and Dong [8]. In MMBO, the authors employed crossover and mutation operators to yield the neighbor regeneration.

As mentioned above, the MBO algorithm has two main characteristics in addition to an easy implementing, a simple structure, and few mathematical requirements. Thus, many heuristics, meta-heuristics, problem-dependent local search methods and operators can be embedded in the above search

978-1-7281-2153-6/19/$31.00 ©2019 IEEE

framework to further enhance exploration and exploitation. Moreover, for the above literature, the simulation experimental results have verified that the MBO algorithm is appropriate and competitive for solving continuous and discrete optimization problems. To the best of our knowledge, the MBO algorithm has not been applied to the LSFS with blocking scheduling problem.

The rest of this paper is organized as follows. After this brief introduction, in Section I, the description of LSFS scheduling problem is stated. Next, the basic MBO algorithm is stated in Section III. Section IV presents the proposed algorithm. Section V provides the experimental results. Finally, the paper ends with some conclusions in Section VI

II. LSFS SCHEDULING PROBLEM

LSFS can be defined as follows [1]: There are n ($\pi = \{\pi(1), \pi(2), ..., \pi(j), ..., \pi(n)\}$) jobs and m machines. Each job has to be processed on each of m machines in the same series. Each job can be split into several sub-lots and each sub-lot being of different processing time sequence is the same on each machine. The constraints of the lot-streaming flow shop scheduling problem are presented below:

For each job, it can be processed at the i-th machine after its front job completed at the i-th machine, in which all the sub-lots of the same job should be processed continuously. At any time, for each machine, it can process at most a sub-lot, and a sub-lot can be processed on at most one machine at one time. The idling case of any two adjacent sub-lots of job j at the same stage allows idle time. Both setup times and sub-lot transportation times are included in processing times. In order to clearly explain the mathematical model, the following notations are given.

notation	meaning
j	job index, j=1, 2,…,n
t	machine index, t=1, 2,…,m
π	a job sequence
Π	the set of all sequences, $\Pi = \{\pi_1, \pi_2, ..., \pi_{PS}\}$
PS	the population size
$\pi(j)$	the j-th job of sequence π
$w_{\pi(j)}$	the total number of sub-lots of job $\pi(j)$
e	the e-th sub-lot with each being indexed by $e = 1, 2, ..., w_{\pi(j)}$
$\pi(j), e$	the e-th sub-lot of job $\pi(j)$
$p_{\pi(j),t}$	the processing time of each sub-lot of job $\pi(j)$ on machine t
$C_{\pi(j),t,e}$	the completion time of the e-th sub-lot of job $\pi(j)$ on machines t
$S_{\pi(j),t,e}$	the start time of the e-th sub-lot of job $\pi(j)$ on machines t

Let $\pi = \{\pi(1), \pi(2), ..., \pi(n)\}$ represent the sequence of jobs to be processed, and the start time of the first sub-lot of the first job on the first machine is zero as $S_{\pi(1),1,1} = 0$. The completion time of each job on each machine can be calculated using the following equations:

$$\begin{cases} S_{\pi(1),1,1} = 0 \\ C_{\pi(1),1,1} = S_{\pi(1),1,1} + p_{\pi(1),1} \end{cases} \tag{1}$$

$$\begin{cases} S_{\pi(1),t,1} = C_{\pi(1),t-1,1} \\ C_{\pi(1),t,1} = S_{\pi(1),t,1} + p_{\pi(1),t} \end{cases} \quad t = 2,3,...m \tag{2}$$

$$\begin{cases} S_{\pi(1),1,e} = C_{\pi(1),1,e-1} \\ C_{\pi(1),1,e} = S_{\pi(1),1,e} + p_{\pi(1),1} \end{cases} \quad e = 2,3,..., w_{\pi(1)} \tag{3}$$

$$\begin{cases} S_{\pi(1),t,e} = \max\{C_{\pi(1),t-1,e}, C_{\pi(1),t,e-1}\} \\ C_{\pi(1),t,e} = S_{\pi(1),t,e} + p_{\pi(1),t} \end{cases} \quad e = 2,3,..., w_{\pi(1)}; t = 2,3,...m \tag{4}$$

$$\begin{cases} S_{\pi(j),1,1} = C_{\pi(j-1),1,w_{\pi(j-1)}} \\ C_{\pi(j),1,1} = S_{\pi(j),1,1} + p_{\pi(1),1} \end{cases} \quad j = 2,3,...,n \tag{5}$$

$$\begin{cases} S_{\pi(j),t,1} = \max\{C_{\pi(j),t-1,1}, C_{\pi(j-1),t,w_{\pi(j-1)}}\} \\ C_{\pi(j),t,1} = S_{\pi(j),t,1} + p_{\pi(j),t} \end{cases} \quad j = 2,3,...,n; t = 2,3,...m \tag{6}$$

$$\begin{cases} S_{\pi(j),1,e} = C_{\pi(j),1,e-1} \\ C_{\pi(j),1,e} = S_{\pi(j),1,e} + p_{\pi(j),1} \end{cases} \quad j = 2,3,...,n; e = 2,3,...w_{\pi(j)} \tag{7}$$

$$\begin{cases} S_{\pi(j),t,e} = \max\{C_{\pi(j),t-1,e}, C_{\pi(j),t,e-1}\} \\ C_{\pi(j),t,e} = S_{\pi(j),t,e} + p_{\pi(j),t} \end{cases} \quad j = 2,3,...,n; e = 2,3,..., w_{\pi(j)}; t = 2,3,...m \tag{8}$$

The makespan, is equal to the time when the last job in the processing sequence is finished at machine m. Its value can be represented according to Eq. (9).

$$C_{\max}(\pi) = C_{\pi(n),m,w_{\pi(n)}} \tag{9}$$

III. INTRODUCTION TO THE BASIC MBO ALGORITHM

The MBO algorithm [5], a neighborhood search technique, is inspired by the V flight formation. The MBO algorithm is an iterative process, like the other swarm intelligence based algorithm, and consists of four main parts. One is the initialization, in which a number of initial solutions corresponding to birds are randomly generated, and place them on a hypothetical V formation arbitrarily. Then, the remaining three parts, i.e., improving the leading solution, improving the other solutions in the population (except leading solution) and replacing the leading solution, are repeated until the termination criterion is satisfied. In the improving the leading solution, some neighbor solutions are

generated by pairwise exchange of any two locations of the current leading solution, and the best solution is selected to update the leading one. In the improving the other solutions phase, each solution is tried to be improved by its neighbor one and the best neighbor solution of the previous one. If the best neighbor one is better than the current one, then the best one replaces the current one. The above two phases are repeated a lifetime of the leading solution. Following that a replacing process of the leading solution is implemented, in which the leading solution is moved to the end and the rest solutions are moved forward a position in turn.

MBO distinguishes from other metaheuristic methods, its properties are that (1) it is a parallel processing that can somehow be regarded as being inherent to GAs and scatter search; (2) there exists benefit mechanism for the solutions (birds) from the solutions in front of them, in which the best unused neighbors are shared with the solutions that follow (here 'unused' refers to a neighbor solution which is not used to replace the existing solution). Although MBO appears to have some similarities to swarm intelligence algorithms and to an artificial bee colony algorithm in particular in which better solutions are explored more, the benefit mechanism is totally unique to MBO.

IV. THE PROPOSED NMBO

This section presents a novel MBO (NMBO) algorithm for the LSFS scheduling problem with makespan criterion. NMBO includes the parameter setting, the initialization population, improving the leading solution, improving the other solutions in the population (except leading solution) and replacing the leading solution. We detail them as follows. Let *nTour* be the lifetime of the leader bird, *nBor* be the number of neighbor solutions, *nShare* be the number of neighbor solutions to be shared with the next solution, and PS be the solution in the population.

A. Initialization population

To generate an initial population with a certain level of quality and diversity, many heuristics, i.e., NEH [9], and MME[10], have been successfully adapted to initialize the seeds of the population. But, they can only generate a single solution. If some good seeds in the initial population can be generated, the efficiency convergence of the whole algorithm will be enhanced. Therefore, the above idea is employed in this study. That is, a multiple-based NEH (MNEH) initial strategy is proposed to yield β solutions with high quality, and a random method is adopted to generate the rest solutions so as to maintain the diversity of the population. The detailed process of generating initialization individuals is shown in Algorithm 1.

Algorithm 1 population initialization

01: **Input:** number of jobs, *n*, the population size *PS*.
02: **Output:** a population with *PS* individuals
03: **Begin**
04: Let $\pi = \phi$, $\pi' = \phi$;
05: **for** *i*=1 **to** *n*;
06: $\pi'(i) = i$;
07: **end for**
08: **for** *i*=1 **to** β *//beginning of MNEH*
09: $\pi_i \leftarrow random\_shuffle(\pi'.begin(), \pi'.end())$;
10: $\pi_i' \leftarrow NEH(\pi_i)$;
11: Calculate makespan, $C_{\max}(\pi_i')$, using Eqs. (1-8);
12: $population \leftarrow \pi_i'$;
13: **end for** *//ending of MNEH*
14: **for** *i*= $\beta+1$ **to** *PS*
15: $\pi_i = random\_shuffle(\pi'.begin(), \pi'.end())$;
16: Calculate makespan, $C_{\max}(\pi_i)$, using Eqs. (1-8);
17: $population \leftarrow \pi_i$;
18: **end for**
19: **End**

In Algorithm 1, *random_shuffle*() is a function of randomized element of sequence. The function has two parameters which are used to determine a range. Then the function generates random element set from the specified range by upsetting the order of elements.

Algorithm 2 *NEH*(π)

01: **Input:** a seed sequence $\pi, \pi = \{\pi(1), \pi(2), ..., \pi(n)\}$, and the number of jobs, *n*.
02: **Output:** a good sequence, π^*
03: **Begin**
04: $\pi \leftarrow$ a seed sequence obtained by the line 9 of Algorithm 1;
05: Pick the first two jobs of π to form two sub-sequences, i.e. $\pi_1 = \{\pi(1), \pi(2)\}$ and $\pi_1 = \{\pi(2), \pi(1)\}$;
06: Calculate makespan of the two sub-sequences with respect to $C_{\max}$, using Eqs. (1-10);
07: Set $\pi^* = \pi^* \bigcup \arg\min_{q\in\{1,2\}} C_{\max}(\pi_q)$;
08: Set $k = 3$;
09: **while** $k \le n$ **do**
10: Insert $\pi(k)$ into k possible positions in π^* to form a temporary set η^k;
11: $\pi^* = \pi^* \bigcup \arg\min_{\pi\in\eta^k} C_{\max}(\pi)$;
12: $k = k+1$;
13: **end while**
14: **Output** π^*
15: **End**

B. Improving the leading solution

The operator has two ways, named forward insertion, *insert_f* , and backward insertion *insert_b* [11].The former randomly selects two different positions, *p*1 and *p*2. If *p*1 < *p*2, all jobs between position *p*1+1 and *p*2 move forward a position in turn. At last, put the original job in position *p*1 into position p2. As for *insert_b* , randomly select two different positions, *p*1 and *p*2. When *p*1 < *p*2, all jobs between position *p*1+1 and *p*2 move backward a position in turn. With respect to the swap operator, randomly select two different positions from the sequence, and interchange their corresponding jobs.

In this section, four strategies based on insert and swap operators are proposed on this account: (1) perform *insert_f* once; (2) apply *insert_b* one time; (3) conduct the swap operator once. Generally speaking, more strategies generate different solutions with a larger probability than a single strategy, and avoid the population trapping in local optima.

We randomly choose one of the above four strategies to generate *nBor* solutions, in which we choose the best neighbor solution to update the leading solution, and the remaining *nBor*-1 solutions are put into two shared neighbor sets, respectively.

C. Improve the other solutions in the population

Various crossover operators have been developed for scheduling problems, such as Similar Job Order Crossover (SJOX) and Similar Block Order Crossover (SBOX) [12]. Zhang et al. proposed a new crossover operator based on SJOX, called Artificial Chromosome SJOX (ACJOX), which can produce better solutions than SJOX [13].

For two parents, when performing SBOX, the identical block of these parents at the same position is selected and copied to their offspring. With respect to the ACJOX operator, a temporary individual is first generated as a reference and compared with each of the two parents, respectively. Only the identical gene between the reference and one parent at the same position is selected and copied to its offspring.

The common drawback is that the two crossover operators utilize information provided by only two parents, not taking full advantage of valuable information provided by other individuals. In addition, SBOX does not concern with the identical block of parents at different positions. In view of this, we propose two improved crossover operators based on SBOX and ACJOX, respectively, named ISBOX and ISJOX, for the LSFS scheduling problem.

The process of ISBOX can be divided into the following steps. First, generate a temporary set consisted of block genes based on all solutions in the current population. Followed that, randomly select two parents from the population, for the first parent, seek for the block genes consisted of adjacent two genes existed in the temporary set. Put the identical block genes into its offspring at the same positions as this parent. Finally, for the rest genes of the offspring, obtain their values based on the One-Point order crossover between the two parents. Similarly, we can get another offspring.

The process of ISOJX can be described according to the following steps. First, generate a temporary individual based on all solutions in the current population. Followed that, for two parents randomly selected from the population, compare the temporary individual with each of them. Put the identical gene at the same position between the temporary individual and the first parent into its offspring at the same positions as this parent. Finally, for the rest genes of the offspring, obtain their values based on the One-Point order crossover between the two parents. Similarly, we can get another offspring.

It is worth noting that ISBOX considers identical genes between the temporary individual and parents at all positions, avoiding excellent genes of these parents lost.

For each solution except for the leading one in the current population, we implement the below steps. First, generate *nBor-nShare* solutions by ISBOX and ISJOX. Second, select the best solutions among *nBor-nShare* solutions and the *nShare* unused best neighbors of the previous solutions to update the current one. In this paper, *nBor*, *nShare* and *nTour* are equal to 3, 1 and 10, respectively.

D. Local search

In this work, the purpose of the local search is to generate a better solution from the neighborhood of a given solution. We adopt an insert-neighborhood-based local search [14], which has been regarded as superior to the swap or exchange neighborhood. Furthermore, we try to present a simple algorithm with few parameters, so some relative algorithms such as taboo search and simulated annealing algorithm are not applied. In this paper, we apply the local search to the solutions generated in subsection 4.3 with a small probability of *pls* (parameter *pls* controls whether the solution undergoes the local search or not). That is, a uniform random number r is generated from 0 and 1, if $r<pls$, the solution will employ the local search. Otherwise, the solution does not perform the local search.

VI. EXPERIMENTS

We compared NMBO with DIWO [3], EDA [4], MMBO [8] to evaluate the performance of the proposed algorithm in this section. The test instance set is composed of 150 different instances, which are divided into 15 subsets and each subset consists of 10 instances with the same size. These subsets range from 10 jobs and 5 machines to 500 jobs and 20 machines. Each instance is independently executed five replications, records the minimal makespan, and obtains the average relative percentage difference of 5 times. For all instances in a group, we obtain the above average relative percentage differences, and denote their average as ARPD. Denote the makespan of the *j*th instance provided by the *i*th algorithm in the *t*th run as $C^i_{j,t}$, C^R_j is the best known solution provided so far by existing algorithms for the specified problem or by our proposed algorithms. From the following Eq. (11), we can see that the smaller the average relative percentage difference APRD is, the better result the algorithm produces. Denote APRD obtained by the *i*th algorithm as $ARPD_i$, then $ARPD_i$ can be stated as follows.

$$APRD_i = \frac{1}{50}\sum_{j=1}^{10}\sum_{t=1}^{5}\frac{C^i_{j,t}-C^R_j}{C^R_j}\times 100 \quad (10)$$

All these algorithms were implemented with C++ in a PC environment with Pentium(R) Dual 2.8GHZ and 2GB memory. Following Yoon, Ventura [15], the related data for each LSFS scheduling problem are given according to the discrete uniform distributions as below. The values of the rest parameters are set in Table I.

TABLE I. PARAMETER SETTINGS

Parameter	Notation	Value
Number of jobs	n	10,50,70,110,200,500
Number of machines	m	5, 10, 20
Number of sub-lots	w	Uniform(1,6)
Processing time	$p_{\pi(j),t}$	Uniform(1,31)
Population size	PS	20
Number of initializing solutions	β	3
local search rate	pls	0.6
stopping time	$Time_{max}$	$T\times n\times m$ milliseconds

Tables II-IV report APRD over each subset for computation time T=5, 15, 30, respectively. It can be seen from Table II that the overall mean APRD value yielded by the NMBO algorithm is equal to 0.49, which is much smaller than 0.96,0.65,0.61 generated by the EDA, DIWO and MMBO algorithm. As the problem size increases, the superiority of the NMBO algorithm over EDA, DIWO and MMBO algorithms increases. On the other side, The results reported in Tables III-IV further justify the superiority of the NMBO algorithm over the EDA, DIWO and MMBO algorithms for computation time T=15, 30, respectively. Thus, we can conclude that the presented NMBO algorithm outperforms EDA, DIWO and MMBO algorithms for LSFS scheduling problems with makespan.

TABLE II. PERFORMANCE COMPARISON OF EDA, DIWO, MMBO AND NMBO ALGORITHMS (T=5)

Instances	EDA	DIWO	MMBO	NMBO	time
10×5	0.49	0.68	0.47	**0.40**	0.25
10×10	0.77	0.63	**0.32**	0.74	0.50
10×20	**0.02**	0.32	0.33	0.90	1.00
50×5	1.04	0.64	0.35	**0.16**	1.25
50×10	0.56	0.64	0.56	**0.38**	2.50
50×20	1.23	0.89	0.99	**0.51**	5.00
70×5	0.92	0.78	0.73	**0.13**	1.75
70×10	0.93	**0.46**	0.62	0.89	3.50
70×20	0.96	0.45	**0.29**	0.45	7.00
110×5	1.54	0.90	0.66	**0.52**	2.75
110×10	0.96	0.82	0.72	**0.36**	5.50
110×20	1.56	0.66	0.62	**0.38**	11.0
200×10	0.67	0.32	**0.25**	0.32	10.0
200×20	1.01	**0.17**	0.92	0.47	20.0
500×20	1.67	1.44	1.36	**0.71**	50.0
average	0.96	0.65	0.61	**0.49**	8.14

TABLE III. PERFORMANCE COMPARISON OF EDA, DIWO, MMBO AND NMBO ALGORITHMS (T=15)

Instances	EDA	DIWO	MMBO	NMBO	time
10×5	**0.01**	0.02	0.05	0.18	0.50
10×10	0.72	0.52	0.43	**0.34**	1.00
10×20	**0.03**	0.76	0.30	0.56	2.00
50×5	0.93	0.56	0.38	**0.11**	2.50
50×10	0.48	0.18	0.19	**0.14**	5.00
50×20	0.59	0.24	**0.12**	0.16	10.00
70×5	0.34	0.18	0.15	**0.09**	3.50
70×10	0.95	0.23	0.19	**0.18**	7.00
70×20	0.67	0.29	**0.17**	0.18	14.00
110×5	0.74	**0.14**	0.55	0.68	5.50
110×10	0.86	0.45	1.06	**0.30**	11.00
110×20	**0.18**	0.23	0.55	0.75	22.00
200×10	0.94	0.20	0.16	**0.08**	20.00
200×20	0.74	0.86	0.52	**0.17**	40.00
500×20	0.50	0.79	0.50	**0.30**	100.0
average	0.59	0.38	0.35	**0.28**	16.27

TABLE IV. PERFORMANCE COMPARISON OF EDA, DIWO, MMBO AND NMBO ALGORITHMS (T=30)

Instances	EDA	DIWO	MMBO	NMBO	time
10×5	0.01	0.02	**0.00**	**0.00**	1.50
10×10	0.22	0.31	0.37	**0.12**	3.00
10×20	**0.02**	0.23	0.10	0.25	6.00
50×5	0.46	0.45	0.32	**0.12**	7.50
50×10	0.30	0.07	**0.04**	0.07	15.00
50×20	0.29	0.01	**0.00**	0.01	30.00
70×5	0.23	**0.07**	0.15	**0.07**	10.50
70×10	0.67	0.30	**0.21**	0.30	21.00
70×20	0.26	**0.00**	**0.00**	0.01	42.00
110×5	0.12	**0.00**	**0.00**	**0.00**	16.50
110×10	0.62	0.42	0.51	**0.20**	33.00
110×20	0.81	0.28	**0.18**	0.27	66.00
200×10	0.67	0.32	0.25	**0.10**	60.00
200×20	0.18	0.28	**0.04**	0.07	120.0
500×20	0.38	**0.15**	0.30	**0.15**	300.0
average	0.35	0.19	0.16	**0.12**	48.80

Table V reports the two-side Wilcoxon rank sum tests of NMBO, EDA, DIWO and MMBO algorithms with significant level equal to 5%. In the tables 45, there are two values, i.e, p value and h value. P is the probability of observing the given result by chance if the null hypothesis is true. When h equals 1, it indicates that the results obtained by the two compared algorithms are obviously different. When h equals 0, it denotes that the difference between the two algorithms is not significant at 5% significant level. From the Table V, we can see that NMOB proposed in this paper is significantly different from the other compared algorithms.

TABLE V. THE WILCOXON TWO-SIDED RANK SUM TEST RESULTSOF NMBO, DIWO, MMBO AND EDA

(NMBO, EDA)		(NMBO, DIWO)		(NMBO, MMBO)	
p	h	p	h	p	h
4.14211e-053	**1**	**5.68921e-056**	**1**	**8.36981e-023**	**1**

VI. CONCLUSIONS

In this paper, we have presented NMBO to minimize makespan for LSFS scheduling problem. In order to perform exploration for promising solutions within the entire solution space, the NMBO-based searching mechanism with an effective population initialization approach is developed. A simple but effective local search algorithm was employed. To further enhance the NMBO algorithm, we adopt the two new crossover operators to obtain solutions for the rest migrating birds. Computational experiments are given and compared with the results yielded by EDA, DIWO, and MMBO algorithms. The simulation results demonstrated the superiority of the proposed NMBO algorithm in terms of solution quality and effectiveness. The future work is to apply the NMBO algorithm to other optimization problems and encourage us to extend the ideas proposed to the different objective functions or multi-objective in scheduling problems.

ACKNOWLEDGMENT

This work is supported by National Nature Science Foundation under Grant 61803192, 61773192, 61503170, 61503220, 61603169, 61773246, and 71533001. Natural Science Foundation of Shandong Province under Grant ZR2017BF039. Natural Science Foundation of Hainan Province under Grant 617182.

REFERENCES

[1] Y. Y. Han, D. W. Gong, Y. C. Jin, Q. K. Pan, "Evolutionary multiobjective blocking lot-Streaming flow shop scheduling with machine breakdowns," IEEE Transactions on Cybernetics, Vol. 49, No. 1, pp. 184-198, 2019.

[2] Q. K. Pan, M. F. Tasgetiren, P. N. Suganthan, et al, "A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem," Computer Engineering & Applications, Vol. 181, No. 12, pp. 2455-2468, 2011.

[3] H. Y. Sang, Q. K. Pan, P. Y. Duan, et al, "An effective discrete invasive weed optimization algorithm for lot-streaming flowshop scheduling problems," Journal of Intelligent Manufacturing, No. 2, pp. 1-13, 2015.

[4] Q. K. Pan, R. Ruiz, "An estimation of distribution algorithm for lot-streaming flow shop problems with setup times," Omega, Vol.40, No.2, pp. 166-180, 2012.

[5] E. Duman, M. Uysal, A. F. Alkaya, "Migrating Birds Optimization: A New Meta-heuristic Approach and Its Application to the Quadratic Assignment Problem," Information Sciences, Vol. 217, pp. 254-263, 2011.

[6] V. Tongur, E.Ülker, "Migrating Birds Optimization for Flow Shop Sequencing Problem," Journal of Computer & Communications, Vol. 02, No. 4, pp. 142-147, 2014.

[7] Q. K. Pan, Y. Dong, "An improved migrating birds optimisation for a hybrid flowshop scheduling with total flowtime minimization," Information Sciences, Vol. 277, No. 2, pp. 643-655, 2014.

[8] S. Niroomand, A.Hadi-Vencheh, "Modified migrating birds optimization algorithm for closed loop layout with exact distances in flexible manufacturing systems," Expert Systems with Applications, Vol. 42, No. 19, pp. 6586-6597, 2015.

[9] M. Nawaz, E. Emory Enscore, I.Ham, "A heuristic algorithm for the m -machine, n -job flow-shop sequencing problem," Omega, Vol. 11, No. 1, pp. 91-95, 1983.

[10] 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.

[11] D. W. Gong, Y. Y. Han, J. Y. Sun. "A Hybrid Discrete Artificial Bee Colony Algorithm for Multi-objective Blocking Lot-Streaming Flow Shop Scheduling Problem" Knowledge-Based Systems, Vol. 148, pp. 115-130, 2018.

[12] V. P. S. Rawat, M. Cusan, A. Deshpande, et al, "Two new robust genetic algorithms for the flowshop scheduling problem," Omega, Vol. 34, No. 5, 461-476, 2006.

[13] Y. Zhang, "Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization," European Journal of Operational Research, Vol. 196, No. 3, pp. 869-876, 2009.

[14] Y. Y. Han, D. W. Gong, X. Y Sun, "A discrete artificial bee colony algorithm incorporating differential evolution for flow shop scheduling problem with blocking," Engineering Optimization, Vol. 47, No. 7, pp. 927-946, 2015.

[15] S.H. Yoon, J. Ventura, "An application of genetic algorithms to lot-streaming flow shop scheduling," IIE Transactions, Vol. 34, No. 9, pp. 779-787, 2002.