Information Sciences 181 (2011) 2455–2468

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem

Quan-Ke Pan [a], M. Fatih Tasgetiren [b], P.N. Suganthan [c,*], T.J. Chua [d]

[a] College of Computer Science, Liaocheng University, Liaocheng 252059, PR China
[b] Department of Industrial Engineering, Yasar University, Bornova, Izmir, Turkey
[c] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore 639798, Singapore
[d] Singapore Institute of Manufacturing Technology, Nanyang Drive 638075, Singapore

ARTICLE INFO

Article history:
Received 4 August 2009
Received in revised form 9 November 2009
Accepted 22 December 2009
Available online 4 January 2010

Keywords:
Flow shop scheduling
Lot-streaming
Artificial bee colony algorithm
Weighted earliness and tardiness criterion

ABSTRACT

In this paper, a discrete artificial bee colony (DABC) algorithm is proposed to solve the lot-streaming flow shop scheduling problem with the criterion of total weighted earliness and tardiness penalties under both the idling and no-idling cases. Unlike the original ABC algorithm, the proposed DABC algorithm represents a food source as a discrete job permutation and applies discrete operators to generate new neighboring food sources for the employed bees, onlookers and scouts. An efficient initialization scheme, which is based on the earliest due date (EDD), the smallest slack time on the last machine (LSL) and the smallest overall slack time (OSL) rules, is presented to construct the initial population with certain quality and diversity. In addition, a self adaptive strategy for generating neighboring food sources based on insert and swap operators is developed to enable the DABC algorithm to work on discrete/combinatorial spaces. Furthermore, a simple but effective local search approach is embedded in the proposed DABC algorithm to enhance the local intensification capability. Through the analysis of experimental results, the highly effective performance of the proposed DABC algorithm is shown against the best performing algorithms from the literature.

1. Introduction

Flow shop scheduling problem is one of the most popular machine scheduling problems with extensive engineering relevance, representing nearly a quarter of manufacturing systems, assembly lines, and information service facilities in use nowadays [13,19,26,27]. In a traditional flow shop scheduling, a job is indivisible and it cannot be transferred to the next machine before its processing is finished [17,18]. However, this may not be the case in many practical situations where a job consists of many identical items. In order to reduce machine waiting time, a job is allowed to be overlapping its operations between successive machines by splitting it into a number of smaller sublots [29]. The job splitting into sublots process is usually called lot-streaming, which is one of the effective techniques used to implement the time-based strategy in today's era of global competition [3]. Through the extensive use of just-in-time (JIT) system in manufacturing, the performance measure related to both earliness and tardiness penalties has raised significant attention [23]. For this reason, this paper aims at solving an n-job, m-machine lot-streaming flow shop scheduling problem with equal size sublots to minimize the total weighted earliness and tardiness penalties. We consider two different cases, namely, the idling case and no-idling case. The no-idling case is the one where there is no-idling production interruption time (i.e., the idle time) between any two

* Corresponding author. Tel.: +65 670 5404; fax: +65 6793 3318.
E-mail addresses: qkpan@gmail.com (Q.-K. Pan), mf_tasgetiren@hotmail.com (M. Fatih Tasgetiren), epnsugan@ntu.edu.sg (P.N. Suganthan), tjchua@simtech.a-star.edu.sg (T.J. Chua).

0020-0255/$ - see front matter © 2010 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2009.12.025

adjacent sublots at the same stage, whereas the idling case allows idle time on machines [3]. A comprehensive review of scheduling problems involving lot-streaming can be found in [3].

The lot-streaming flow shop scheduling problems can be classified into two categories, i.e., single-job and multi-job problems. For the first category, the main goal is to find the optimal sublot sizes. A flow shop with makespan criterion is considered in [17] indicating that it is sufficient in a single-job model to use the same sublot sizes for all machines. Regarding total flowtime minimization, optimal sublot size policies and two heuristic methods are given for a single job in a flow shop by Kropp and Smunt [11]. Bukchin et al. [1] proposed the optimal solution properties and developed a solution procedure for solving a two-machine flow shop scheduling problem with the total flow time criterion where sublot detached setups and batch availability were considered.

For the second category, the primary purpose is to simultaneously obtain the best sublot allocation, i.e., sublot starting and completion times, and the best job sequence. For this propose, Vickson and Alfredsson [24] studied the effects of transfer batches in the two-machine and special three-machine problems with unit-size sublots. Cetinkaya [2] proposed an optimal transfer batch and scheduling algorithm for a two-stage flow shop scheduling problem with setup, processing and removal times separated. Vickson [25] examined a two-machine lot-streaming flow shop problem involving setup and sublot transfer times with respect to both continuous and integer valued sublot sizes. Sriskandarajah and Wagneur [22] devised an efficient heuristic for solving the problem of simultaneous lot-streaming and scheduling of multiple products in a two-machine no-wait flow shop.

For the m-machine flow shop scheduling problems, Kumar et al. [12] extended the approach in Sriskandarajah and Wagneur [22] to an m-machine case. Kalir and Sarin [4] presented a bottleneck minimal idleness (BMI) heuristic to sequence a set of batches to be processed in equal sublots for minimizing makespan. Recently, Marimuthu et al. [15] proposed a genetic algorithm (GA) and hybrid genetic algorithm (HGA) for an m-machine flow shop with makespan and total flow time criteria. More recently, ant colony optimization (ACO) and threshold accepting (TA) algorithms were presented for solving the same problem by Marimuthu et al. [16]. As mentioned before, with the advent and development of JIT manufacturing systems, efforts have been focused on minimizing the total weighted earliness and tardiness penalties for the m-machine lot-streaming flow shop scheduling problems. Yoon and Ventura [29] presented a procedure where a linear programming (LP) formulation was designed to obtain optimal sublot completion times for a given sequence and sixteen pairwise interchange methods were utilized to search for the best sequence. Later on, Yoon and Ventura [28] proposed a hybrid genetic algorithm (HGA) by incorporating the LP and a pairwise interchange method into the traditional genetic algorithm. More recently, Tseng and Liao [23] developed a discrete particle swarm optimization (DPSO), where a net benefit of movement (NBM) algorithm was utilized to obtain the optimal allocation of the sublots for a given sequence, and the objective function values obtained by the NBM heuristic are used to guide the search towards the best sequence. In these three recent literatures, only the idling case is addressed. To the best of our knowledge, there is no published paper for dealing with the lot-streaming flow shop with total earliness and tardiness criterion with respect to the no-idling production interruption time between any two adjacent sublots.

In general, swarm intelligence is based on collective behavior of self-organized systems. As a typical example of swarm intelligence, the bee swarming around her hive has received significant interest from researchers. Recently, by modeling the specific intelligent behaviors of honey bee swarms, an artificial bee colony (ABC) algorithm is developed by Karaboga [9] to optimize multi-variable and multi-modal continuous functions. Numerical comparisons demonstrated that the performance of the ABC algorithm is competitive to other population-based algorithm with an advantage of employing fewer control parameters [6–10]. Due to its simplicity and ease of implementation, the ABC algorithm has captured much attention and has been applied to solve many practical optimization problems [5,6,21]. Since there is no published work to deal with the lot-streaming flow shop problem by using the ABC algorithm, we present a novel discrete ABC (DABC) algorithm for solving the lot-streaming flow shop problem with a criterion of total weighted earliness and tardiness penalties. Furthermore, both no-idling case and idling case between any two adjacent sublots of a job at the same stage are considered, respectively. As in Yoon and Ventura [28] and Tseng and Liao [23] we assume equal size sublots and infinite capacity buffers, and divide the problem solving process into two closely dependent phases. In the first phase, the NBM heuristic developed by Tseng and Liao [23] is used to determine the optimal sublot allocations for a given sequence. In the second phase, the proposed DABC algorithm is applied to find the best sequence in the entire region. Computational experiments and comparisons show that the proposed DABC algorithm outperforms the two recent best performing algorithms for solving the lot-streaming flow shop scheduling problem with the total weighted earliness and tardiness criterion.

The rest of the paper is organized as follows. In Section 2, the lot-streaming flow shop scheduling problem with equal size sublots is formulated. In Section 3, the basic ABC algorithm is introduced. Section 4 describes the basic components of DABC algorithm whereas the computational results and comparisons are described in Section 5. Finally, Section 6 presents the concluding remarks.

2. Lot-streaming flow shop scheduling problem

The problem considered in this research is the minimization of the total weighted earliness and tardiness penalties for n-job, m-machine lot-streaming flow shop scheduling with infinite buffer capacities between successive machines and equal size sublots for a job on all the machines. Each job $j \in J = \{1, 2, \ldots, n\}$ will be sequentially processed on m machines and the job

sequence is the same on each machine $i \in M = \{1,2,\ldots,m\}$. In order to reduce the lead time and accelerate the production, each job j can be split into s_j number of smaller sublots with equal size such that s_j is the same for all the machines. Once the processing of a sublot on a machine is completed, it can be transferred to the downstream machine immediately. However, all the s_j sublots of job $j$should be processed continuously. Under the no-idling constraints, there is no-idling production interruption time between any two adjacent sublots of job j at the same stage, whereas the idling case allows idle times. Furthermore, at any time, each machine can process at most one sublot and each sublot can be processed on at most one machine. Let t_{ij} denote an uninterrupted processing time of a sublot of job j on machine $i \in M$; c_{ijk} be the completion time of the kth sublot of job j on machine i; d_j be the due date of job j; and α_j and β_j be the earliness and tardiness penalties of job j, respectively. Given that the release time of all jobs is zero, and both setup times and sublot transportation times are included in processing times, then the objective is to find a sequence with the optimal sublot starting and completion times of this sequence so that the total weighted earliness and tardiness penalties are minimized as follows:

$$\min\ f(\pi) = \sum_{j=1}^{n} (\alpha_j E_j + \beta_j T_j), \tag{1}$$

where π is a processing sequence of the jobs; $E_j = \max\left\{0, c_{mjs_j} - d_j\right\}$ is the earliness of job j; and $T_j = \max\left\{0, d_j - c_{mjs_j}\right\}$ is the tardiness of job j.

An example of a single job, three equal size sublots, three-machine flow shop with the job processing times of 6, 3 and 6 time units is shown in Fig. 1. If the job is not split into sublots, the job completion time is 15 time units (see in Fig. 1a). When the job is split into three sublots and no-idling production interruption time is allowed between any two adjacent sublots, the job completion time is reduced to 11 time units (see in Fig. 1b), whereas for the idling case, the job completion time is further reduced to nine time units. Obviously, the completion time of the job under the idling case is shorter than the one under the no-idling case with the same sublot type. However, there are also many practical applications for the lot-streaming flow shop scheduling under no-idling case [20]. Therefore, in this paper, both the idling and no-idling cases are considered.

According to [23,28], the problem can be divided into two closely dependent sub-problems. The first one is to find the optimal sublot allocation for a given sequence whereas the second one is to search for the best sequence within the entire solution space. In order to determine the optimal starting and completion times on each machine for a given sequence, Yoon and Ventura [28] presented an LP model. However, the LP model can only be applied to small-sized problems due to its large computational requirement. To overcome this drawback, recently, Tseng and Liao [23] proposed a net benefit of movement (NBM) heuristic which can serve the same propose but requires much less computation time. At the beginning of the NBM algorithm, each sublot is started as soon as possible and only a forward movement is considered to minimize the objective function. Then the algorithm constructs the schedule in a backward manner by moving forward the last sublot of a job on the

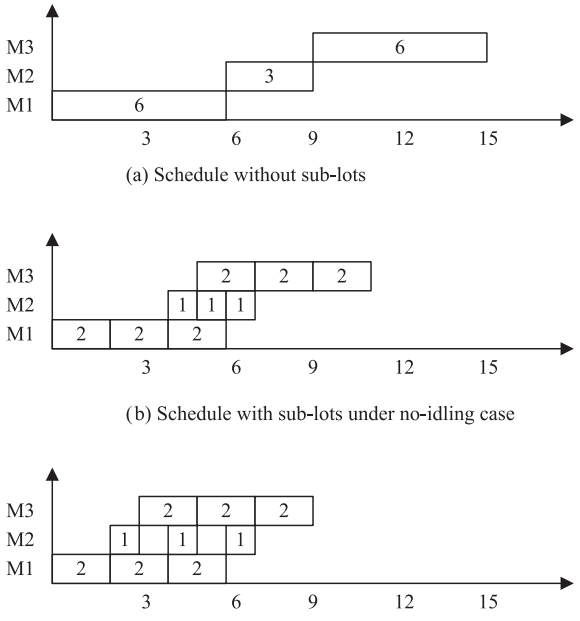


Fig. 1. An example of the lot-streaming flow shop scheduling.

last machine if this movement can yield a net benefit. In this paper, the NBM heuristic is employed to find the optimal sublot allocation for a given sequence. Note that although the original NBM heuristic was presented for the idling case, it can be easily applied to the no-idling case by only considering the idle time between two successive jobs.

3. The basic artificial bee colony algorithm

The artificial bee colony (ABC) algorithm is a new swarm intelligence based optimizer proposed by Karaboga [9] for multivariable and multi-modal continuous function optimization. Inspired by the intelligent foraging behavior of honeybee swarm, the ABC algorithm classifies the foraging artificial bees into three groups, namely, employed bees, onlookers and scouts. A bee that is currently exploiting a food source is called an *employed bee*. A bee waiting in the hive for making decision to choose a food source is named as an *onlooker*. A bee carrying out a random search for a new food source is called a *scout*. In the ABC algorithm, each solution to the problem under consideration is called a *food source* and represented by an n-dimension real-valued vector, whereas the fitness of the solution is corresponded to the *nectar amount* of the associated food resource. Similar to the other swarm intelligence based approaches, the ABC algorithm is an iterative process. It starts with a population of randomly generated solutions or food sources. Then the following three steps are repeated until a termination criterion is met [8]:

1. Send the employed bees onto the food sources and then measure their nectar amounts.
2. Select the food sources by the onlookers after share the information of employed bees and determine the nectar amount of the food sources.
3. Determine the scout bees and send them onto possible food sources.

The main components of the basic ABC algorithm are described in detail below.

3.1. Initialization of the parameters

The parameters of the basic ABC algorithm are the number of food sources (SN) which is equal to the number of the employed bees or on looker bees, the number of trials after which a food source is assumed to be abandoned (*limit*), and a termination criterion. In the basic ABC algorithm, the number of employed bees or the onlookers is set equal to the number of food sources in the population. In other words, for every food source, there is only one employed bee.

3.2. Initialization of the population

The initial population of solutions is filled with SN number of randomly generated n-dimensional real-valued vectors (i.e., food sources). Let $X_i = \{x_{i1}, x_{i2}, \ldots, x_{in}\}$ represent the ith food source in the population, and then each food source is generated as follows:

$$x_{ij} = LB_j + (UB_j - LB_j) \times r \quad \text{for } j = 1, 2, \ldots, n \text{ and } i = 1, 2, \ldots, SN, \tag{2}$$

where r is a uniform random number in the range $[0, 1]$; and LB_j and UB_j are the lower and upper bounds for the dimension j, respectively. These food sources are randomly assigned to PS number of employed bees and their finesses are evaluated.

3.3. Initialization of the bee phase

At this stage, each employed bee x_i generates a new food source x_{new} in the neighborhood of its present position as follows.

$$x_{new(j)} = x_{ij} + (x_{ij} - x_{kj}) \times r, \tag{3}$$

where $k \in \{1, 2, \ldots, PS\} \wedge k \neq i$ and $j \in \{1, 2, \ldots, n\}$ are randomly chosen indexes. r is a uniformly distributed real number in $[-1, 1]$.

Once x_{new} is obtained, it will be evaluated and compared to x_i. If the fitness of x_{new} is equal to or better than that of x_i, x_{new} will replace x_i and become a new member of the population; otherwise x_i is retained. In other words, a greedy selection mechanism is employed between the old and candidate solutions.

3.4. Onlooker bee phase

An onlooker bee evaluates the nectar information taken from all the employed bees and selects a food source x_i depending on its probability value p_i calculated by the following expression.

$$p_i = \frac{f_i}{\sum_{i=1}^{SN} f_i}, \tag{4}$$

where f_i is the nectar amount (i.e., the fitness value) of the ith food source x_i. Obviously, the higher the f_i is, the more probability that the ith food source is selected.

Once the onlooker has selected her food source x_i, she produces a modification on x_i by using Eq. (3). As in the case of the employed bees, if the modified food source has a better or equal nectar amount than x_i, the modified food source will replace x_i and become a new member in the population.

3.5. Scout bee phase

If a food source x_i cannot be further improved through a predetermined number of trials *limit*, the food source is assumed to be abandoned, and the corresponding employed bee becomes a scout. The scout produces a food source randomly as follows:

$$x_{ij} = LB_j + (UB_j - LB_j) \times r \quad \text{for } j = 1, 2, \ldots, n. \tag{5}$$

where r is a uniform random number in the range $[0,1]$.

In the basic ABC algorithm, at each cycle at most one scout goes outside for searching a new food source.

3.6. Main steps of the ABC algorithm

Based on the above explanation of initializing the algorithm parameters and population, the employed bee phase, the onlooker bee phase, and the scout bee phase, the main steps of the basic ABC algorithm can be summarized as follows [8]:

Step 1: Initialization.
Step 2: Place the employed bees on their food sources.
Step 3: Place the onlooker bees on the food sources depending on their nectar amounts.
Step 4: Send the scouts to the search area for discovering new food sources.
Step 5: Memorize the best food source found so far.
Step 6: If a termination is not satisfied, go to step 2; otherwise stop the procedure and output the best food source found so far.

4. The proposed ABC algorithm for lot-streaming flow shop problem

The basic ABC algorithm was originally designed for continuous function optimization. In order to make it applicable for solving the lot-streaming flow shop problem with total weighted earliness and tardiness criterion, a novel discrete variant of the ABC algorithm, called DABC, is proposed in this section.

4.1. Solution representation

The permutation based representation is easy to decode a schedule, which has been widely used in literature for a verity of permutation flow shop scheduling problems [26]. For this reason, it is also used in the DABC algorithm.

4.2. Population initialization

To guarantee an initial population with certain quality and diversity, a portion of food sources are generated by using some priority rules whereas the rest are produced randomly. For the scheduling problems with total weighted earliness and tardiness criterion, the earliest due date (EDD), the smallest slack time on the last machine (LSL), and the smallest overall slack time (OSL) rules are commonly used to produce initial sequences . Thus, this paper applies these rules to produce three initial solutions, respectively. The EDD rule is defined by sorting the jobs according to their ascending due dates. In other words, a permutation of n jobs $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ is sorted according to $d_{\pi_{(j)}} \leqslant d_{\pi_{(j+1)}}$. The LSL rule sorts the jobs according to their ascending slack times on the last machines such that $d_{\pi(j)} - s_{\pi(j)} \times t_{m,\pi(j)} \leqslant d_{\pi(j+1)} - s_{\pi(j+1)} \times t_{m,\pi(j+1)}$. The OSL rule is to sort the jobs according to their ascending overall slack times such that $d_{\pi(j)} - s_{\pi(j)} \times \sum_{r=1}^{m} t_{r,\pi(j)} \leqslant d_{\pi(j+1)} - s_{\pi(j+1)} \times \sum_{r=1}^{m} t_{r,\pi(j+1)}$.

4.3. Employed bee phase

According to the basic ABC algorithm, the employed bees generate food sources in the neighborhood of their current positions. As to the permutation based neighborhood structure, insert and swap operators are commonly used to produce neighboring solutions in the literature [26]. The insert operator of a permutation π is defined by removing a job from π from its original position j and inserts it into another position k such that $(k \in \{j, j-1\})$ whereas the swap operator produces a neighbor of π by interchanging two jobs of π in the different positions. To enrich the neighborhood structure and diversify the

population, four neighboring approaches based on the insert or swap operator are separately utilized to generate neighboring food sources for the employed bees as follows:

1. Performing one insert operator to a sequence π.
2. Performing one swap operator to a sequence π.
3. Performing two insert operators to a sequence π.
4. Performing two swap operators to a sequence π.

Each method for the generation of neighboring food sources may have different performance during the evolution process. Therefore, an adaptive mechanism is presented in Section 4.5 for determining the best approach by the learning mechanism in order to balance the global exploration and local exploitation. As for the selection, new good source is always accepted if it is better than the current food source the same as in the basic ABC algorithm which carries out a greedy selection procedure.

4.4. Onlooker bee phase

In the basic ABC algorithm, an onlooker bee selects a food source x_i depending on its winning probability value p_i, which is similar to the wheel selection in GAs. However, the tournament selection is widely used in GA applications due to its simplicity and ability to escape from local optima [26]. For this reason, we propose a tournament selection with the size of 2 in the DABC algorithm. In the tournament selection, an onlooker bee selects a food source x_i in such a way that two food sources are picked up randomly from the population, and compared to each other, then the better one is chosen. In addition, the onlooker utilizes the same method as used by the employed bee to produce a new neighboring solution. If the new food source obtained is better than or equal to the current one, the new food source will replace the current one and become a new member in the population.

4.5. A self-adaptive strategy to produce neighboring solutions

Both employed bees and onlookers apply a self-adaptive strategy to find neighboring food sources. The self-adaptive strategy is presented as follows. At the beginning, an initial neighbor list (NL) with a specified length is generated by filling the list one by one randomly from four neighboring approaches explained before. Then the DABC algorithm is started. During the evolution process, one approach from the NL is taken out and used to generate a new food source for an employed bee or onlooker. If the new food source successfully replaces the current one, this approach will enter into a winning neighboring list (WNL). Once the NL is empty, it is refilled as follows: 75% of the NL is refilled from the WNL list, and then the rest of 25% is refilled by a random selection from four different approaches. If the WNL is empty (this may happen when the search is performed near an optimum with negligible population diversity), the latest NL is used again. The above process is repeated until a termination criterion is reached. As a result, the proper neighboring approach can be gradually learned by the algorithm itself to suit to the particular problem and the particular phase of search process. Note that the WNL is reset to zero once the NL is refilled to avoid any inappropriate long-term accumulation effects. In our experiments, the length of NL is set as 200, and the influence on the performance of the DABC algorithm due to small variations of the NL length is found to be insignificant.

An example is given as follows. Suppose that an NL with length equal to 8 is randomly generated (see in Fig. 2a). During the evolution process, the first approach '*insert*' is taken out (see in Fig. 2b) and used to generate a neighbor π_i^* for the solution π_i. Given that π_i^* is better than π_i. It will replace π_i and become a new member of the population. Accordingly, the approach '*insert*' will enter into WNL (see in Fig. 2c). Then the second approach '*swap*' is taken out (see in Fig. 2d) and used to yield π_i^* for the solution π_i. Suppose that π_i^* is worse than π_i, the associated approach '*swap*' will be discarded. Repeat the above process until all approaches are taken out and the NL is empty (see in Fig. 2f and g).

The refilling process for the NL then starts. A uniform random number between 0 and 1 is generated firstly. Suppose that the random number is less than 0.75, then randomly select an approach from the WNL (e.g. insert) and put it in the first position of NL (see in Fig. 3a). Secondly, another uniform random number is generated. Suppose that it is greater than 0.75, then the approach is randomly chosen among the four approaches, and put them into second position of the NL (see in Fig. 3b). Repeat the above process until the NL is filled, and the WNL is reset empty (see in Fig. 3c and d).

4.6. Scout bee phase

In the basic ABC algorithm, a scout produces a food source randomly in the predefined search scope. This will decrease the search efficacy, since the best food source in the population often carries better information than others during the evolution process, and the search space around it could be the most promising region. Therefore, in the DABC algorithm, the scout generates a food source by performing several 'insert' operators to the best food source in the population. And to avoid the algorithm trap into a local optimum, at least three '*insert*' are performed. In addition, as in the case of the basic ABC algorithm, at most one food source is generated by a scout bee at each iteration.

NL	
Index	Approach
1	One INSERT
2	One SWAP
3	Two INSERTs
4	Two SWAPs
5	One SWAP
6	Two SWAPs
7	Two INSERTs
8	One INSERT

(a) NL with length equal 8

NL	
Index	Approach
1	
2	One SWAP
3	Two INSERTs
4	Two SWAPs
5	One SWAP
6	Two SWAPs
7	Two INSERTs
8	One INSERT

(b) NL after first element is taken out

WNL	
Index	Approach
1	One INSERT

(c) WNL after first neighbor is produced

NL	
Index	Approach
1	
2	
3	Two INSERTs
4	Two SWAPs
5	One SWAP
6	Two SWAPs
7	Two INSERTs
8	One INSERT

(d) NL after second element is taken out

WNL	
Index	Approach
1	One INSERT

(e) WNL after second neighbor is produced

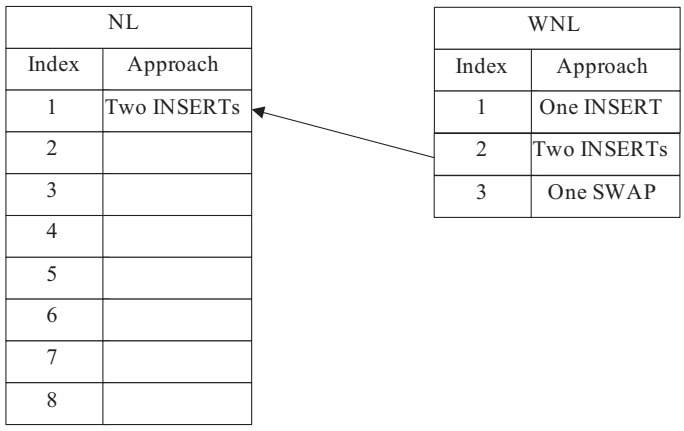
NL	
Index	Approach
1	
2	
3	
4	
5	
6	
7	
8	

(f) NL after eighth element is taken out

WNL	
Index	Approach
1	One INSERT
2	Two INSERTs
3	One SWAP

(g) WNL after eighth neighbor is produced

Fig. 2. The process for taking out operators from NL.



NL	
Index	Approach
1	Two INSERTs
2	
3	
4	
5	
6	
7	
8	

WNL	
Index	Approach
1	One INSERT
2	Two INSERTs
3	One SWAP

(a) randomly select an operator from WNL and put it into NL

NL	
Index	Approach
1	Two INSERTs
2	One SWAP
3	
4	
5	
6	
7	
8	

One SWAP

(b) randomly select one of the four approaches and put it into NL

NL	
Index	Approach
1	Two INSERTs
2	One SWAP
3	One INSERT
4	One SWAP
5	Two INSERTs
6	One SWAP
7	One INSERT
8	Two SWAPs

(c) The refilled NL

WNL	
Index	Approach
1	
2	
3	

(d) The empty WNL

Fig. 3. The process for refilling NL.

4.7. A local search algorithm

In order to enhance the exploitation capability of the algorithm, a simple local search is embedded in the proposed DABC algorithm. The local search is based on the insert and swap operator. Let $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ be an initial job permutation and *l*max be the maximum number of iterations for the local search algorithm. The local search algorithm is described as follows.

Step 1: Set $l = 1$.
Step 2: Perform one insert or swap operator to π. Denote the obtained permutation π^*.
Step 3: If π^* is better than or equal to π, let $\pi = \pi^*$.
Step 4: Set $l = l + 1$. If $l \leqslant l_{\max}$, go to Step 2; otherwise stop the procedure and return π.

In the DABC algorithm, the above local search is only applied to the neighboring food source found by an employed bee with a probability P_L. That is, a uniform random number between 0 and 1 is firstly generated, and the food source will undergo the local search if the random number is less than P_L. In addition, to avoid both cycling search and getting trapped into local optima, the insert operator is used in the local search if the neighboring food source is produced by the swap-based operators; otherwise, the swap operator is applied in the local search.

4.8. Computational procedure of the proposed DABC algorithm

Having discussed all the components of the DABC algorithm, the complete computational procedure is outlined as follows:

Step 1: Set the parameters *SN*, *limit*, P_L and $loop_{\max}$.
Note that in the DABC algorithm, the number of employed bees or onlookers is equal to the number of solutions in the population.
Step 2: Initialize the population: $P = \{\pi_1, \pi_2, \ldots, \pi_{PS}\}$ and evaluate each solution in the population.
Step 3: Employed bee phase.
For $i = 1, 2, \ldots, PS$, repeat the following sub-steps:

1. Produce a new solution π_i^* for the ith employed bee who is associated with solution π_i by using the self-adaptive strategy presented in Section 4.5 and evaluate the new solution.
2. If $r < P_L$, perform local search to π_i^*.
3. If π_i^* is better than or equal to π_i, let $\pi_i = \pi_i^*$.

Step 4: Onlooker phase.
For each $i = 1, 2, \ldots, PS$, repeat the following sub-steps:

1. Select a food source in the population for the onlooker bee i by using the tournament selection presented in Section 4.3.
2. Generate a new solution for the onlooker by using the self-adaptive strategy presented in Section 4.5 and evaluate it.
3. If the generated solution is better than or equal to the selected one, update the population.

Step 5: Scout phase.
If a solution in the population has not been improved during the last *limit* number of trials , abandon it and put a new solution generated by performing several *insert* operators to the best solution into the population. If there are several such solutions, randomly select one.
Step 6: Memorize the best solution achieved so far.
Step 7: If the termination criterion is reached, return the best solution found so far; otherwise go to Step 3.

5. Computational results and comparisons

In this section, we test the performance of the DABC algorithm for solving the lot-streaming flow shop problem with total weighted earliness and tardiness criterion under both the no-idling and idling cases. For this propose, 20 problem sizes each with the following sizes ($n = 10, 20, 30, 40$, $m = 3, 5, 7, 10, 15$) are randomly generated, and the related data is given by the discrete uniform distributions as follows: $t_{ij} \in U(1, 31)$, $s_j \in U(1, 6)$, $\alpha_j \in U(1, 6)$, $\beta_j \in U(1, 6)$, $d_i \in U(15 \times n, 15 \times (m + n))$. We coded all the algorithms in Visual C++ and conducted experiments on a Pentium PIV 3.0 GHz PC with 1 G MB memory. The parameters for the proposed DABC algorithm were set as follow: $SN = 10$, $limit = 20$, $P_L = 0.1$, and $l_{\max} = n^2$. On the other hand, for the best performing algorithms from the literature, their parameters were fixed at the same ones as given in their papers. To make a fair comparison, all the algorithms adopted the same maximum CPU time limit of $CT = 5n^2m$ ms as a termination criterion. For each instance, we carried out 30 independent replications and for each replication we calculate the relative percentage increase (RPI) as follows:

$$\mathrm{RPI}(O_i) = \frac{O_i - C^*}{C^*} \times 100, \tag{6}$$

where O_i is the objective function value generated in the ith replication, and C^* is the best objective function value found by any of the algorithms compared. Obviously, the smaller the RPI value, the better result the algorithm produces. In addition,

we also calculate the average relative percentage increase (ARPI) and standard deviation (SD) over the 30 replications as statistics for the solution quality.

5.1. Comparison under no-idling case

There are several existing meta-heuristics for solving the lot-streaming flow shop scheduling problem with total weighted earliness and tardiness criterion. Typically, Yoon and Ventura [28] presented a hybrid genetic algorithm (HGA), where the non-adjacent pairwise interchange based local search and the LP formulation were used to overcome the premature convergence and maintain the search power of the traditional genetic algorithm. And the computational experiments showed that the proposed HGA works well for the problems considered. Recently, Tseng and Liao [23] developed a new DPSO algorithm by introducing an inheritance scheme into particle's construction of the earlier version of their DPSO algorithm

Table 1
Comparison of different algorithms without local search under no-idling case.

$n \times m$	$DPSO_{NL}$		$DABC_{NL}$		GA_{NL}		*t*-Test	
	ARPI	SD	ARPI	SD	ARPI	SD	(ABC_{NL}, $DPSO_{NL}$)	(ABC_{NL},GA_{NL})
10×3	0.4703	0.2804	0.1615	0.2012	12.8991	6.4631	1	1
10×5	0.6380	1.2703	0	0	14.4079	5.5651	1	1
10×7	0.0102	0.0386	0	0	14.5362	5.0486	0	1
10×10	0.1840	0.2926	0	0	8.1256	3.8550	1	1
10×15	0	0	0	0	12.4473	4.3689	0	1
20×3	1.5055	1.1169	0.0005	0.0025	48.7548	6.5272	1	1
20×5	1.3381	0.6843	0.1670	0.0849	50.1042	10.0133	1	1
20×7	1.6020	1.0847	0	0	46.2141	7.3323	1	1
20×10	0.7038	0.5862	0.0906	0.1462	39.1604	5.7001	1	1
30×15	1.6106	0.7665	0	0	44.1871	8.7462	1	1
30×3	1.5581	0.8820	0.0400	0.0457	78.1761	8.3882	1	1
30×5	1.6766	1.1397	0.0661	0.1830	78.3263	9.9322	1	1
30×7	3.0014	0.7620	0.5716	0.4471	77.9213	8.9553	1	1
30×10	1.1704	0.5644	0.0621	0.0392	64.7227	7.4870	1	1
30×15	1.6087	0.8580	0.0182	0.0572	71.4290	6.1484	1	1
40×3	3.8339	1.0828	0.9908	0.8841	91.6082	8.6771	1	1
40×5	3.2921	1.0130	0.9097	0.7322	85.2229	6.8213	1	1
40×7	3.3039	1.6200	0.3461	0.4046	103.3269	9.3987	1	1
40×10	2.2771	1.4229	0.2033	0.2126	95.1839	9.3481	1	1
40×15	2.9756	1.1394	0.4911	0.5373	89.1686	6.8407	1	1
Mean	1.6380	0.8302	0.2059	0.1989	56.2961	7.2808		

Table 2
Comparison of different algorithms with local search under no-idling case.

$n \times m$	HDPSO		DABC		HGA		*t*-Test	
	ARPI	SD	ARPI	SD	ARPI	SD	(DABC, HDPSO)	(DABC, HGA)
10×3	0.2961	0.1816	0	0	14.5169	7.5796	1	1
10×5	0	0	0	0	14.0640	5.7110	0	1
10×7	0	0	0	0	15.1098	5.1717	0	1
10×10	0	0	0	0	8.7008	4.1961	0	1
10×15	0.1327	0.4049	0	0	12.5958	4.3309	0	1
20×3	0.0005	0.0025	0	0	50.8744	7.1132	0	1
20×5	0.1600	0.0898	0.1113	0.1059	54.2589	9.9344	0	1
20×7	0	0	0	0	49.1546	8.2449	0	1
20×10	0.1996	0.1598	0	0	41.9112	5.2184	1	1
30×15	0	0	0	0	49.1853	7.7843	0	1
30×3	1.6194	1.5413	0.0407	0.0405	81.6779	10.2031	1	1
30×5	0.6367	1.0386	0	0	86.1659	9.6917	1	1
30×7	0.7715	0.5729	0.4756	0.3403	80.8398	8.0610	1	1
30×10	0.0460	0.0264	0.0278	0.0283	68.6216	7.4924	1	1
30×15	0.2077	1.1375	0	0	77.8597	8.6564	0	1
40×3	13.5884	3.3445	0.6719	0.7351	95.8190	9.8535	1	1
40×5	7.8491	2.5696	0.6303	0.5015	90.2602	7.9221	1	1
40×7	4.0808	1.7770	0.1126	0.0861	110.5808	9.4115	1	1
40×10	1.2204	1.7994	0.0890	0.0957	99.6326	9.7449	1	1
40×15	1.1251	0.5034	0.1351	0.1021	98.3821	9.2488	1	1
Mean	1.5967	0.7575	0.1147	0.1018	60.0106	7.7785		

[14]. To further stress the exploitation ability, a hybrid DPSO (HDPSO) algorithm was developed by incorporating an interchange-and-insertion based local search scheme into their DPSO algorithm. It was concluded that the DPSO and HDPSO algorithms performed much better than the GA (i.e. HGA without the non-adjacent pairwise interchange based local search) and HGA, respectively. However, the above algorithms were originally designed for the lot-streaming flow shop scheduling problem under the idling case, that is, idling production interruption times between any two adjacent sublots of a job are allowed. To apply them to the lot-streaming flow shop scheduling problem under the no-idling case, we modify their objective function evaluation method by considering no-idling production interruption time between any two adjacent sublots of a job.

Following the work of Tseng and Liao [23], we first compare the proposed DABC algorithm without local search, denoted as $DABC_{NL}$, with the GA and DPSO algorithms to show the effectiveness of the DABC-based search. It should be noted that the LP formulation in GA is replaced by the NBM heuristic to save computational time. Table 1 summarizes the computational results for each problem size. Note that in the last two columns of Table 1, the h values of the two-sided t-tests on the compared algorithms (**A**,**B**) are reported. An h value equal to 1 or -1 indicates that results obtained by the algorithm **A** is significantly better or worse than those by the algorithm **B**, while h value equal to zero implies that the results by the two compared algorithms are not statistically different.

It can be easily seen from Table 1 that the $DABC_{NL}$ algorithm is the winner, since it produces the smallest overall mean *APRI* value (0.2059) with the smallest overall mean SD value (0.1989), which is much better than those by the $DPSO_{NL}$ (1.6380/0.8302) and GA (56.2961/7.2808) algorithms. More excitingly, the $DABC_{NL}$ algorithm achieves the best *APRI* value for all the problem sizes. When comparing to HGA, the $DABC_{NL}$ algorithm produces significantly better results for all the problem sizes. At the same time, when compared to the DPSO algorithm, the $DABC_{NL}$ algorithm finds significantly better results for 18 out of 20 problem sizes. Hence, it is concluded that the $DABC_{NL}$ algorithm without local search is clearly superior to the GA and DPSO algorithms for solving the lot-streaming flow shop scheduling problem with the total weighted earliness and tardiness criterion under the no-idling case under the same computational time.

Then we compare the three algorithms with local search. The computational results are reported in Table 2.

It is clear from Table 2 that the DABC algorithm is the best one in terms of the average percentage increase and standard deviation. In comparison to HGA, the DABC algorithm yields significantly better results for all the problem sizes. When comparing to HDPSO, the DABC algorithm produces significantly better results for 11 out of 20 problem sizes. However, for the remaining nine problem sizes, the results generated by the two algorithms are not significantly different at the 5% significant level. Therefore, it is concluded that the DABC algorithm outperforms both the HGA and HDPSO algorithms to minimize the total weighted earliness and tardiness penalties for the lot-streaming flow shop scheduling problem under the no-idling case with the same computational effort.

Finally, we compare the DABC algorithm with and without local search. The computational results are summarized in Table 3. It can be observed that the results for 7 out of 20 problems sizes which are generated by the $DABC_{NL}$ algorithm are significantly improved by the DABC algorithm, which demonstrates the effectiveness of incorporating a local search into the DABC variant. In other words, the superiority in terms of search ability and efficiency of the DABC algorithm should be attributed to the combination of global search and local search with an appropriate balance between exploration and exploitation.

Table 3
Comparison of DABC with and without local search under no-idling case.

$n \times m$	$DABC_{NL}$		DABC		t-Test
	ARPI	SD	ARPI	SD	(DABC,$DABC_{NL}$)
10×3	0.1615	0.2012	0	0	1
10×5	0	0	0	0	0
10×7	0	0	0	0	0
10×10	0	0	0	0	0
10×15	0	0	0	0	0
20×3	0.0005	0.0025	0	0	0
20×5	0.1670	0.0849	0.1113	0.1059	1
20×7	0	0	0	0	0
20×10	0.0906	0.1462	0	0	1
30×15	0	0	0	0	0
30×3	0.0400	0.0457	0.0407	0.0405	0
30×5	0.0661	0.1830	0	0	0
30×7	0.5716	0.4471	0.4756	0.3403	0
30×10	0.0621	0.0392	0.0278	0.0283	1
30×15	0.0182	0.0572	0	0	0
40×3	0.9908	0.8841	0.6719	0.7351	0
40×5	0.9097	0.7322	0.6303	0.5015	0
40×7	0.3461	0.4046	0.1160	0.0861	1
40×10	0.2033	0.2126	0.0890	0.0957	1
40×15	0.5045	0.5374	0.1351	0.1021	1
Mean	0.2066	0.1989	0.1149	0.1018	

Table 4
Comparison of different algorithms without local search under idling case.

$n \times m$	$DPSO_{NL}$		$DABC_{NL}$		GA_{NL}		*t*-Test	
	ARPI	SD	ARPI	SD	ARPI	SD	(ABC_{NL}, $DPSO_{NL}$)	(ABC_{NL}, GA_{NL})
10×3	2.1945	1.5083	0.3852	0.9990	19.0126	6.0073	1	1
10×5	0.0048	0.0181	0	0	25.1557	6.7992	0	1
10×7	0.3328	0.7589	0	0	25.4427	7.6377	1	1
10×10	0.4661	1.0384	0	0	15.7762	4.6013	1	1
10×15	0.1154	0.4560	0	0	20.9582	6.8665	0	1
20×3	4.0140	1.7759	0.5176	1.0089	57.8052	8.5210	1	1
20×5	2.1695	1.2090	0.3019	0.3904	60.7073	10.0629	1	1
20×7	3.2951	2.1097	0.1425	0.2059	62.8374	10.4876	1	1
20×10	1.9502	0.8106	0.3857	0.4261	45.2674	7.4102	1	1
30×15	1.7480	0.8538	0.2084	0.2170	48.7352	7.8537	1	1
30×3	5.5208	2.0907	3.1399	1.8159	87.6195	12.8889	1	1
30×5	3.2295	1.3556	1.5310	0.6831	96.9963	12.4416	1	1
30×7	5.1501	1.4732	2.4172	1.4890	92.2751	9.0664	1	1
30×10	1.8174	0.8046	1.1873	0.4273	77.5153	6.7797	1	1
30×15	3.5953	1.2444	1.5312	0.9469	76.7238	8.3083	1	1
40×3	4.3384	1.5880	2.0899	0.9011	107.6651	11.5961	1	1
40×5	4.7957	1.5124	2.6632	1.1091	103.5353	8.9237	1	1
40×7	4.5746	2.4297	1.8686	1.6223	128.3174	9.8195	1	1
40×10	3.4409	0.8714	2.5488	1.0735	107.6709	8.4311	1	1
40×15	3.3360	1.5053	1.2258	0.9060	98.4175	7.8749	1	1
Mean	2.8045	1.2707	1.1072	0.7111	67.9217	8.6189		

5.2. Comparison under idling case

In this section, we compare the proposed $DABC_{NL}$/DABC algorithm with the GA/HGA and DPSO/HDPSO algorithms under the idling case. We first compare the $DABC_{NL}$ algorithm with the GA and DPSO algorithms, and summarize the computational results and the two-sided *t*-test results in Table 4. It can be easily observed from the Table 4 that the $DABC_{NL}$ algorithm performs much better than the GA and DPSO algorithms at the same computational time, since it produces significantly better results than the GA for all the problem sizes, and generates significantly better results than the DPSO algorithm for 18 out of 20 problem sizes.

We further compare the DABC algorithm with the GA and HDPSO algorithms, and report the computational results and the two-sided *t*-test results in Table 5. It is clear from the Table 5 that the DABC algorithm yields solutions of superior quality against the HGA and HDPSO algorithms.

Table 5
Comparison of different algorithms with local search under idling case.

$n \times m$	HDPSO		DABC		HGA		*t*-Test	
	ARPI	SD	ARPI	SD	ARPI	SD	(DABC, HDPSO)	(DABC, HGA)
10×3	0.5779	1.1755	0	0	19.0126	6.0073	1	1
10×5	0	0	0	0	25.1557	6.7992	0	1
10×7	0	0	0	0	25.4427	7.6377	0	1
10×10	0	0	0	0	15.7762	4.6013	0	1
10×15	0	0	0	0	20.9582	6.8665	0	1
20×3	5.2329	2.9422	0.1359	0.1498	57.8052	8.5210	1	1
20×5	0.4907	0.6257	0.1078	0.2544	60.7073	10.0629	1	1
20×7	0.2923	0.6114	0.0322	0.0501	62.8374	10.4876	1	1
20×10	0.1838	0.2157	0.0897	0.1691	45.2674	7.4102	0	1
30×15	0.0476	0.0967	0.0238	0.0725	48.7352	7.8537	0	1
30×3	30.6035	8.3293	1.5607	1.3087	87.7082	12.8950	1	1
30×5	13.4604	5.4178	1.0424	0.5627	97.1849	12.4535	1	1
30×7	10.5217	3.3206	1.5302	0.6888	92.5296	9.0784	1	1
30×10	1.6608	1.3011	0.4568	0.3622	77.5957	6.7828	1	1
30×15	2.1904	1.5252	0.1811	0.3475	76.8980	8.3165	1	1
40×3	47.4938	9.9819	1.1538	0.6496	107.4263	11.5828	1	1
40×5	32.9344	7.9091	1.1957	0.7134	102.6311	8.8841	1	1
40×7	25.1983	5.1084	0.8184	0.6788	128.6459	9.8336	1	1
40×10	15.2250	4.3901	0.8698	0.8302	109.0368	8.4865	1	1
40×15	6.7767	2.9463	0.5227	0.2677	99.6959	7.9256	1	1
Mean	9.6445	2.7948	0.4860	0.3553	68.0525	8.6243		

Table 6
Comparison of DABC with and without local search under idling case.

$n \times m$	$DABC_{NL}$		DABC		t-Test
	ARPI	SD	ARPI	SD	$(DABC, DABC_{NL})$
10×3	0.3852	0.9990	0	0	1
10×5	0	0	0	0	0
10×7	0	0	0	0	0
10×10	0	0	0	0	0
10×15	0	0	0	0	0
20×3	0.5176	1.0089	0.1359	0.1498	1
20×5	0.3019	0.3904	0.1078	0.2544	1
20×7	0.1425	0.2059	0.0322	0.0501	1
20×10	0.3857	0.4261	0.0897	0.1691	1
30×15	0.2084	0.2170	0.0238	0.0725	1
30×3	3.1887	1.8167	1.5607	1.3087	1
30×5	1.6282	0.6837	1.0424	0.5627	1
30×7	2.5528	1.4910	1.5302	0.6888	1
30×10	1.2331	0.4275	0.4568	0.3622	1
30×15	1.6313	0.9478	0.1811	0.3475	1
40×3	2.0899	0.9011	1.2702	0.6504	1
40×5	2.6632	1.1091	1.6473	0.7166	1
40×7	2.0152	1.6246	0.8184	0.6788	1
40×10	3.2233	1.0805	0.8698	0.8302	1
40×15	1.8779	0.9118	0.5227	0.2677	1
Mean	1.2022	0.7121	0.5144	0.3555	

Finally, the computational results produced by the DABC algorithm with and without local search in Table 6 further suggest that the superiority in terms of search ability and efficiency of the DABC algorithm should be attributed to the combination of global search and local search with an appropriate balance between exploration and exploitation.

In a nutshell, it can be concluded that, at the same computational time, the proposed DABC algorithm performs much better than the existing GA and HDPSO algorithms to minimize total earliness and tardiness penalties for the lot-streaming problem with equal sized sublots under both the idling and no-idling cases.

6. Conclusions

This paper aimed at minimizing total weighted earliness and tardiness penalties for the lot-streaming flow shop scheduling problems with equal sized sublots. We examined the problem under both the idling and no-idling cases and proposed a novel discrete artificial bee colony (DABC) algorithm. To the best of our knowledge, this was the first reported application of the ABC algorithm for solving the problem under consideration. In the proposed algorithm, the food sources were represented as discrete job permutations. The ABC-based searching mechanism with an effective population initialization approach and a self-adaptive neighboring food source generation strategy were developed to perform exploration for promising solutions within the entire solution space. Furthermore, a simple but effective local search algorithm was employed to further exploit the best solution. Due to the reasonable hybridization of the ABC search and local search, the proposed DABC algorithm had the ability to obtain good results for the lot-streaming flow shop scheduling problems with total weighted earliness and tardiness criterion. Computational simulations and comparisons demonstrated the effectiveness and efficiency of the proposed DABC algorithm. Our future work is to investigate the other meta-heuristics for the lot-streaming flow shop problem and generalize the application of the ABC algorithms to solve other combinatorial optimization problems.

Acknowledgements

This research is partially supported by National Science Foundation of China under Grants 60874075, 70871065, and Open Research Foundation from State Key Laboratory of Digital Manufacturing Equipment and Technology (Huazhong University of Science and Technology). Authors also acknowledge the financial support offered by the A*Star (Agency for Science, Technology and Research, Singapore) under the Grant #052 101 0020.

References

[1] J. Bukchin, M. Tzur, M. Jaffer, Lot splitting to minimize average flow-time in a two-machine flow-shop, IIE Transactions 34 (2002) 953–970.
[2] F.C. Cetinkaya, Lot streaming in a two-stage flow shop with set-up, processing and removal times separated, Journal of Operational Research Society 45 (1994) 1445–1455.
[3] J.H. Chang, H.N. Chiu, A comprehensive review of lot streaming, International Journal of Production Research 43 (2005) 1515–1536.
[4] A.A. Kalir, S.C. Sarin, A near-optimal heuristic for the sequencing problem in multiple-batch flow-shops with small equal sublots, Omega 29 (2001) 577–584.
[5] F. Kang, J. Li, Q. Xu, Structural inverse analysis by hybrid simplex artificial bee colony algorithms, Computers and Structures 87 (2009) 861–870.

[6] N. Karaboga, A new design method based on artificial bee colony algorithm for digital IIR filters, Journal of the Franklin Institute 346 (4) (2009) 328–348.
[7] N. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing 8 (2008) 687–697.
[8] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Applied Mathematics and Computation (2009), doi:10.1016/j.amc.2009.03.90.
[9] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005.
[10] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm, Journal of Global Optimization 39 (2007) 459–471.
[11] D.H. Kropp, T.L. Smunt, Optimal and heuristic models for lot splitting in a flow shop, Decision Sciences 21 (1990) 691–709.
[12] S. Kumar, T.P. Bagchi, C. Sriskandarajah, Lot streaming and scheduling heuristics for *m*-machine no-wait flow shops, Computers and Industrial Engineering 38 (2000) 149–172.
[13] W.C. Lee, C.C. Wu, Some single-machine and *m*-machine flow shop scheduling problems with learning considerations, Information Sciences 179 (2009) 3885–3892.
[14] C.J. Liao, C.T. Tseng, P. Luarn, A discrete version of particle swarm optimization for flow shop scheduling problem, Computers and Operations Research 34 (2007) 3099–3111.
[15] S. Marimulthu, S.G. Ponnambalam, N. Jawahar, Evolutionary algorithms for scheduling *m*-machine flow shop with lot streaming, Robotics and Computer-Integrated Manufacturing 24 (2008) 125–139.
[16] S. Marimuthu, S.G. Ponnambalam, N. Jawahar, Threshold accepting and Ant-colony optimization algorithm for scheduling *m*-machine flow shop with lot streaming, Journal of Material Processing Technology 209 (2009) 1026–1041.
[17] C.N. Potts, K.K. Baker, Flow shop scheduling with lot streaming, Operation Research Letters 8 (1989) 297–303.
[18] C.N. Potts, W.L.N. Van, Integrating scheduling with batching and lot-sizing: a review of algorithms and complexity, Journal of the Operational Research Society 43 (1992) 395–406.
[19] T.M. Reza, R.V. Alireza, H.M. Ali, A hybrid multi-objective immune algorithm for a flow shop scheduling problem with bi-objectives: weighted mean completion time and weighted mean tardiness, Information Sciences 177 (22) (2007) 5072–5090.
[20] S.C. Sarin, A.A. Kalir, M. Chen, A single-lot, unified cost-based flow shop lot-streaming problem, International Journal of Production Economics (2008) 113;413–424.
[21] A. Singh, An artificial bee colony algorithm for the leaf-constrained minimum spanning tree problem, Applied Soft Computing 9 (2009) 625–631.
[22] C. Sriskandarajah, E. Wagneur, Lot streaming and scheduling multiple products in two-machine no-wait flow shops, IIE Transactions 31 (1999) 695–707.
[23] C.T. Tseng, C.J. Liao, A discrete particle swarm optimization for lot-streaming flow shop scheduling problem, European Journal of Operational Research 191 (2008) 360–373.
[24] R.G. Vickson, B.E. Alfredsson, Two- and three-machine flow shop scheduling problems with equal size transfer batches, International Journal of Production Research 30 (1992) 1551–1574.
[25] R.G. Vickson, Optimal lot streaming for multiple products in a two-machine flow shop, European Journal of Operational Research 85 (1995) 556–575.
[26] L. Wang, Shop Scheduling with Genetic Algorithms, Tsinghua Univ Press, Beijing, China, 2003.
[27] Y.Q. Yin, D.H. Xu, K.B. Sun, H.X. Li, Some scheduling problems with general position-dependent and time-dependent learning effects, Information Sciences 179 (2009) 2416–2425.
[28] S.H. Yoon, J.A. Ventura, An application of genetic algorithms to lot-streaming flow shop scheduling, IIE Transactions 34 (2002) 779–787.
[29] S.H. Yoon, J.A. Ventura, Minimizing the mean weighted absolute deviation from due dates in lot-streaming flow shop scheduling, Computers and Operations Research 29 (2002) 1301–1315.