Applied Mathematical Modelling 37 (2013) 6758–6779

Contents lists available at SciVerse ScienceDirect

Applied Mathematical Modelling

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

A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion

M. Fatih Tasgetiren [a,*], Quan-Ke Pan [b], P.N. Suganthan [c], Adalet Oner [a]

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

ARTICLE INFO

Article history:
Received 11 October 2010
Received in revised form 7 January 2013
Accepted 18 February 2013
Available online 27 February 2013

Keywords:
Artificial bee colony algorithm
No-idle permutation flowshop scheduling problem
Metaheuristics
Evolutionary algorithms
Genetic algorithm

ABSTRACT

In this paper, we present a discrete artificial bee colony algorithm to solve the no-idle permutation flowshop scheduling problem with the total tardiness criterion. The no-idle permutation flowshop problem is a variant of the well-known permutation flowshop scheduling problem where idle time is not allowed on machines. In other words, the start time of processing the first job on a given machine must be delayed in order to satisfy the no-idle constraint. The paper presents the following contributions: First of all, a discrete artificial bee colony algorithm is presented to solve the problem on hand first time in the literature. Secondly, some novel methods of calculating the total tardiness from makespan are introduced for the no-idle permutation flowshop scheduling problem. Finally, the main contribution of the paper is due to the fact that a novel speed-up method for the insertion neighborhood is developed for the total tardiness criterion. The performance of the discrete artificial bee colony algorithm is evaluated against a traditional genetic algorithm. The computational results show its highly competitive performance when compared to the genetic algorithm. Ultimately, we provide the best known solutions for the total tardiness criterion with different due date tightness levels for the first time in the literature for the Taillard's benchmark suit.

1. Introduction

In a flowshop, the processing order for all jobs is the same. Furthermore, jobs are assumed to a permutation and therefore, once a permutation is fixed for all jobs on the first machine, this permutation is maintained for all machines, which is so called a permutation flowshop scheduling problem (PFSP). There exist several performance measures when dealing with scheduling in a flowshop. The makespan criterion is the most commonly studied performance measure in the literature [1,2]. On the other hand, due date based other objectives have not been attracted enough interest from the researchers until recently. Among due-date based performance measures, tardiness minimization focuses on finding schedules in order to satisfy the external due dates promised to customers. Therefore, it is of significance and importance to real life considerations [3,4].

This paper is basically concerned with solving a variant of the PFSP where no-idle times are allowed on machines. The no-idle constraint refers to an important practical situation in the production environment, where expensive machinery

* Corresponding author.
E-mail addresses: fatih.tasgetiren@yasar.edu.tr (M.F. Tasgetiren), panquanke@gmail.com (Q.-K. Pan), epnsugan@ntu.edu.sg (P.N. Suganthan), adalet.oner@yasar.edu.tr (A. Oner).

0307-904X/$ - see front matter © 2013 Elsevier Inc. All rights reserved.
http://dx.doi.org/10.1016/j.apm.2013.02.011

is employed [5]. Idling on such expensive machinery is often not desirable. For instance, the steppers employed in the production of integrated circuits through photolithography are clear examples. Some other examples arise in industries where less expensive machinery is employed. However, they cannot be stopped and restarted. For example, ceramic roller kilns consume large quantities of natural gas when it is in operation. Idling cannot be an option in this case since it takes several days to stop and to restart the kiln because of a very large thermal inertia. In such cases, idling should be avoided. Another practical example is due to the furnace in the fiberglass processing, where glass batches are reduced to molten. Since it takes three days to heat the furnace back to the required temperature of 2800 $^{\circ}F$, the furnace should stay on during the entire production season. In addition to the above, a three-machine flowshop production of engine blocks in a foundry is presented in [6]. It includes the casting of sand molds and sand cores. The molds are filled up with metal in fusion and the cores prevent the metal to fill some species in the mold. The casting machines work without idle times due to both economic and technological constraints.

In a no-idle permutation flowshop scheduling (NIPFS) problem, each machine has to process jobs without any interruption from the start of processing the first job to the completion of the last job. Therefore, when needed, the start of processing the first job on a given machine must be delayed in order to meet the no-idle requirement. Here we denote it with the well-known three fold notation of $F_m|prmu, no-idle|C_{\max}$. The computational complexity of the $F_m|prmu, no-idle|C_{\max}$ problem is briefly commented in [7]. The NP-Hardness of the $F3|prmu, no-idle|C_{\max}$ problem was proved by [6,8]. Therefore, it has a significant importance both in theory and engineering applications to develop effective and efficient approaches for the problem discussed in this paper.

In spite of its practical importance, the $F_m|prmu, no-idle|C_{\max}$ problem has not attracted much attention in the literature [9]. To the best of our knowledge, a polynomial time algorithm for solving the $F2|prmu, no-idle|\sum C_j$ problem optimally was presented in [10]. The makespan criterion was studied for the first time in [11] whereas heuristic approaches for the general m – machine no-idle PFSP with the makespan criterion were examined in [12]. A branch and bound (B&B) method is also presented by [8] for the general m-machine no-idle PFSP with the makespan criterion.

Some mistakes in the paper by [10] were reported in [13]. The $F3|prmu, no-idle|C_{\max}$ problem was studied in [14]. The same problem was also studied in [6] where a lower bound and an efficient heuristic are presented. This new heuristic favors the earlier method of the authors [15]. They published this work later on in [16]. Kamburowski in [17] further enhanced the idea in [6] by proposing a network representation. A heuristic for the the $F3|prmu, no-idle|C_{\max}$ problem based on the traveling salesman problem (TSP) was proposed in [16]. The $F2|prmu, no-idle|C_{\max}$ and the $F_m|prmu, no-idle|C_{\max}$ problems were studied in two similar papers, respectively [18,19]. Kalczynski and Kamburowski in [20] developed a constructive heuristic, named KK heuristic, for the $F_m|prmu, no-idle|C_{\max}$ problem with a time complexity of $O(n^2m)$. The authors also presented an adaptation of the NEH heuristic [21] for the NIPFS problem. In addition, Kalczynski and Kamburowski studied the interactions between the no-idle and no-wait flowshops in [20], too. Recently, Baraz and Mosheiov introduced an improved two-stage greedy algorithm consisting of a simple greedy heuristic and an improvement step based on the API method in [9].

In recent years, meta-heuristics have attracted increasing attention to solve scheduling problems owing to the fact that they are able to provide high quality solutions with reasonable computational effort [22]. In addition to the above literature, in two similar papers, Pan and Wang proposed a discrete differential evolution (DDE) and a discrete particle swarm optimization (DPSO) algorithms for the same problem in [23,24]. In both papers, a speed-up scheme for the insertion neighborhood is proposed, which reduces the computational complexity of a single insertion neighborhood scan from $O(n^3m)$ to $O(n^2m)$ when the insertion is carried out in order. The speed-up they proposed is based on the very well-known accelerations presented by [25] for the insertion neighborhood for the PFSP. In fact, both in DDE and DPSO, an advanced local search form, which is an iterated greedy (IG) algorithm proposed in [26] is employed as a local search. Both DDE and DPSO used the well-known benchmark suite of [27] by treating them as the NIPFS instances in order to test the results. In both papers, the authors tested the proposed methods against the heuristics in [9,20]. More recently, an IG algorithm for the NIPFS problem with the makespan criterion was presented in [5]. They employed their own benchmark suite and examined the performance of IG in detail against the existing heuristics and meta-heuristics from the literature. To the best of our knowledge, this paper is the first to presents a discrete artificial bee colony algorithm to study the NIPFS problem with the total tardiness criterion in the literature.

In general, swarm intelligence is based on collective behavior of self-organized systems [28]. 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 in [28–32] 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 algorithms with an advantage of employing fewer control parameters [28–32]. Recently, a discrete version of ABC algorithm is applied to the lot-streaming flowshop scheduling problem in [33,34]. Since there is no published work to deal with the NIPFS problem with the total tardiness criterion by using the ABC algorithm, we present a novel discrete ABC (DABC) algorithm as well as a genetic algorithm to be compared for solving the NIPFS problem with the total tardiness criterion in this paper.

The remaining paper is organized as follows. Section 2 introduces the NIPFS problem. Section 3 presents the DABC algorithm in detail while Section 4 briefly explains the traditional GA. Section 5 discusses the computational results over benchmark problems. Finally, Section 6 summarizes the concluding remarks.

2. No-idle permutation flowshop scheduling problem algorithm

The no-idle permutation flow shop scheduling problem with n jobs and m machines can be defined as follows. Each of n ($j = 1, 2, \ldots, n$) jobs will be sequenced through m machines ($k = 1, 2, \ldots, m$). Operation $o(j, k)$ corresponds to the processing of job j on machine k during an uninterrupted processing time $p(j, k)$, where its setup time is included in it, whereas $d(j)$ denotes the due date of job j. 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. To follow the no-idle restriction, each machine must process jobs without any interruption from the start of processing the first job to the completion of processing the last job. In other words, there must be no idle time between the processing of any consecutive operations on each machine. The aim is then to find a schedule such that the processing order of jobs is the same on each machine and its total tardiness is minimized. We follow Pan and Wang in [23,24] for the formulation of the NIPFS problem with the makespan criterion and extend it to the total tardiness criterion. The formulation of total tardiness variant consists of forward and backward passes as well as their combined method to facilitate the speed-up method, which are explained and well illustrated with examples below.

2.1. Forward pass calculation

Let a job permutation $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ represent the schedule of jobs to be processed, and $\pi_j^E = \{\pi_1, \pi_2, \ldots, \pi_j\}$ be a partial schedule of π such that $1 < j < n$. In addition, $F(\pi_j^E, k, k+1)$ refers to the lower bound for the minimum difference between the completion of processing the last job of π_j^E on machines $k+1$ and k, which is restricted by the no-idle constraint. Then, $F(\pi_j^E, k, k+1)$ can be computed as follows:

$$F(\pi_1^E, k, k+1) = p(\pi_1, k+1) \quad j = 1, 2, \ldots, m-1 \tag{1}$$

$$F\left(\pi_j^E, k, k+1\right) = \max\left\{F(\pi_{j-1}^E, k, k+1) - p(\pi_j, k), 0\right\} + p(\pi_j, k+1) \quad j = 2, 3, \ldots, n \text{ and } k = 1, 2, \ldots, m-1 \tag{2}$$

Then, the makespan of job π_n on machine m can be given by

$$C(\pi_n, m) = C_{\max}(\pi_n^E) = \sum_{k=1}^{m-1} F(\pi_n^E, k, k+1) + \sum_{j=1}^{n} p(\pi_j, 1) \tag{3}$$

In fact, the Eq. (3) is a completion time of job π_n on the last machine m. Hence, the completion time of job π_i on the last machine m can be computed by subtracting the processing time of job $i+1$ from the completion time of job $i+1$ on the last machine m as follows:

$$C(\pi_j, m) = C(\pi_{j+1}, m) - p(\pi_{j+1}, m), \quad i = n-1, n-2, \ldots, 1 \tag{4}$$

Then, the total tardiness is given by:

$$T = \sum_{j=1}^{n} (\max(C(\pi_j, m) - d(\pi_j), 0)) \tag{5}$$

An example instance for 3-job 3-machine problem is given in Table 1. Through Figs. 1a-1d, the forward calculation is illustrated in detail with a permutation, $\pi = \{1, 2, 3\}$, as well as with the due date tightness factor of $\tau = 1$.

$$F(\pi_1^E, k, k+1) = p(\pi_1, k+1), \quad k = 1, 2, \ldots, m-1$$

$$F(\pi_1^E, 1, 2) = p(1, 2) = 1$$

$$F(\pi_1^E, 2, 3) = p(1, 3) = 3$$

$$F\left(\pi_j^E, k, k+1\right) = \max\left\{F\left(\pi_{j-1}^E, k, k+1\right) - p(\pi_j, k), 0\right\} + p(\pi_j, k+1), \quad j = 2 \text{ and } k = 1, 2$$

Table 1
An example instance.

Job(j)	*Machine*(k)			Due Date
	1	2	3	$d_j = \tau \times \sum_{k=1}^{m} p_{jk}$
p_{1j}	4	1	3	8
p_{2j}	2	3	3	8
p_{3j}	2	2	3	7

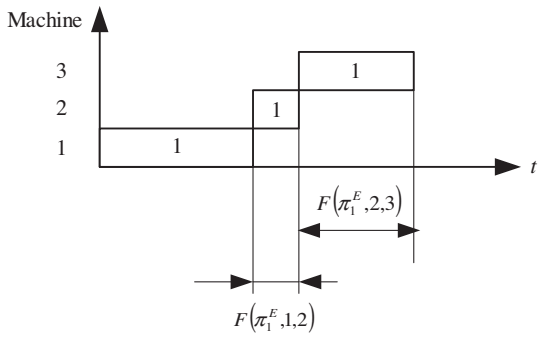

Fig. 1a. Computation of $F(\pi_1^E, k, k+1)$.

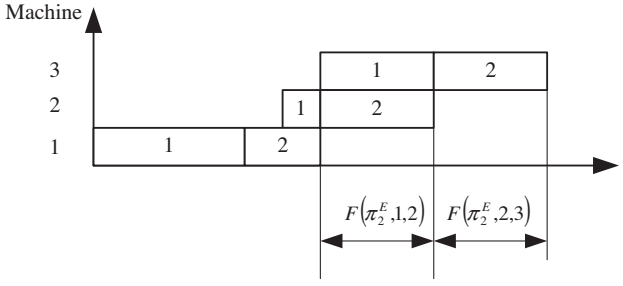

Fig. 1b. Computation of $F(\pi_2^E, k, k+1)$.

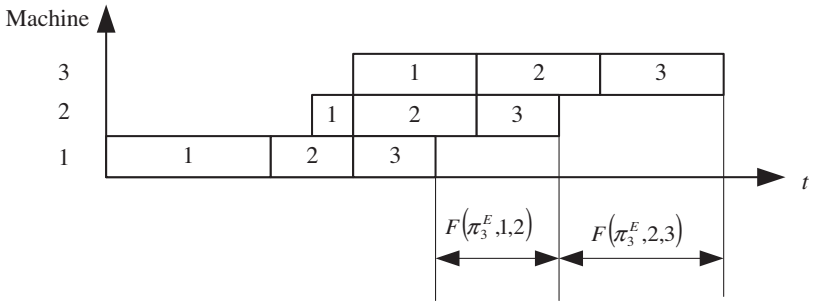

Fig. 1c. Computation of $F(\pi_3^E, k, k+1)$.

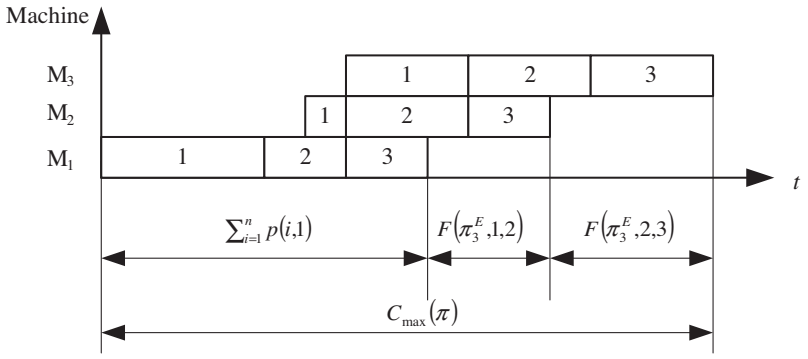

Fig. 1d. Computation of $C_{\max}(\pi)$.

$$F(\pi_2^E,1,2)=\max\{F(\pi_1^E,1,2)-p(\pi_2,1),0\}+p(\pi_2,2)$$

$$F(\pi_2^E,1,2)=\max\{F(\pi_1^E,1,2)-p(2,1),0\}+p(2,2)$$

$$F(\pi_2^E,1,2)=\max\{(1-2),0\}+3=3$$

$$F(\pi_2^E,2,3)=\max\{F(\pi_1^E,2,3)-p(\pi_2,2),0\}+p(\pi_2,3)$$

$$F(\pi_2^E,2,3)=\max\{F(\pi_1^E,2,3)-p(2,2),0\}+p(2,3)$$

$$F(\pi_2^E,2,3)=\max\{(3-3),0\}+3=3$$

$$F\left(\pi_j^E,k,k+1\right)=\max\left\{F\left(\pi_{j-1}^E,k,k+1\right)-p(\pi_j,k),0\right\}+p(\pi_j,k+1)\quad j=3\ and\ k=1,2$$

$$F(\pi_3^E,1,2)=\max\{F(\pi_2^E,1,2)-p(\pi_3,1),0\}+p(\pi_3,2)$$

$$F(\pi_3^E,1,2)=\max\{F(\pi_2^E,1,2)-p(3,1),0\}+p(3,2)$$

$$F(\pi_3^E,1,2)=\max\{(3-2),0\}+2=3$$

$$F(\pi_3^E,2,3)=\max\{F(\pi_2^E,2,3)-p(\pi_3,2),0\}+p(\pi_3,3)$$

$$F(\pi_3^E,2,3)=\max\{F(\pi_2^E,2,3)-p(3,2),0\}+p(3,3)$$

$$F(\pi_3^E,2,3)=\max\{(3-2),0\}+3=4$$

$$C(\pi_3,3)=\sum_{k=1}^{3-1}F(\pi_3^E,k,k+1)+\sum_{j=1}^{3}p(\pi_j,1)$$

$$C(\pi_3,3)=F(\pi_3^E,1,2)+(\pi_3^E,2,3)+p(1,1)+p(2,1)+p(3,1)$$

$$C(\pi_3,3)=3+4+4+2+2=15$$

$$C(\pi_2,3)=C(\pi_3,3)-p(\pi_3,3)$$

$$C(\pi_2,3)=15-3=12$$

$$C(\pi_1,3)=C(\pi_2,3)-p(\pi_2,3)$$

$$C(\pi_1,3)=12-3=9$$

$$T=\sum_{j=1}^{3}(\max(C(\pi_j,3)-d(\pi_j),0))$$

$$T=\max(C(\pi_1,3)-d(\pi_1),0)+\max(C(\pi_2,3)-d(\pi_2),0)+\max(C(\pi_3,3)-d(\pi_3),0)$$

$$T=\max(9-8,0)+\max(12-8,0)+\max(15-7,0)=1+4+8=13$$

2.2. Backward pass calculation

Let $\pi_j^F=\{\pi_j,\pi_{j+1},\ldots,\pi_n\}$ denote an another partial schedule of π such that $1<j<n$. And let $E(\pi_j^F,k,k+1)$ be the lower bound for the minimum delay between the start of processing the first job of π^F on machines $k+1$ and k, which is restricted by the no-idle constraint. Then, $E(\pi_j^F,k,k+1)$ can be computed as follows:

$$E(\pi_n^F,k,k+1)=p(\pi_n,k),\quad k=1,2,\ldots,m-1 \tag{6}$$

$$E\left(\pi_j^F,k,k+1\right)=\max\left\{E(\pi_{j+1}^F,k,k+1)-p(\pi_j,k+1),0\right\}+p(\pi_j,k),\quad j=n-1,n-2,\ldots,1\ and\ k=1,2,\ldots,m-1 \tag{7}$$

The completion time $C(\pi_1, m)$ of job π_1 on the last machine m can be given as follows:

$$C(\pi_1, m) = \sum_{k=1}^{m-1} E(\pi_1^F, k, k+1) + p(\pi_1, m) \tag{8}$$

Then, the completion time $C(\pi_{j+1}, m)$ of job π_{j+1} on the last machine m can be obtained as follows:

$$C(\pi_{j+1}, m) = C(\pi_j, m) + p(\pi_{j+1}, m), \quad j = 1, 2, \cdots, n-1 \tag{9}$$

Figs. 2a-2d illustrate the calculation of makespan for a 3-job 3-machine problem.
Ultimately, the total tardiness is given by:

$$T = \sum_{j=1}^{n} (\max(C(\pi_j, m) - d(\pi_j), 0)) \tag{10}$$

Using the same example instance for 3-job 3-machine problem given in Table 1 the forward calculation is illustrated in detail below with a permutation $\pi = \{1, 2, 3\}$ as well as with the due date tightness factor of $\tau = 1$.

$$E(\pi_n^F, k, k+1) = p(\pi_n, k) \quad k = 1, 2$$

$$E(\pi_3^F, 1, 2) = p(\pi_3, 1) = p(3, 1) = 2$$

$$E(\pi_3^F, 2, 3) = p(\pi_3, 2) = p(3, 2) = 2$$

$$E\left(\pi_j^F, k, k+1\right) = \max\left\{E(\pi_{j+1}^F, k, k+1) - p(\pi_j, k+1), 0\right\} + p(\pi_j, k), \quad j = 2 \text{ and } k = 1, 2$$

$$E(\pi_2^F, 1, 2) = \max\left\{E(\pi_3^F, 1, 2) - p(\pi_2, 2), 0\right\} + p(\pi_2, 1)$$

$$E(\pi_2^F, 1, 2) = \max\left\{E(\pi_3^F, 1, 2) - p(2, 2), 0\right\} + p(2, 1)$$

$$E(\pi_2^F, 1, 2) = \max\{(2-3), 0\} + 2 = 2$$

$$E(\pi_2^F, 2, 3) = \max\left\{E(\pi_3^F, 2, 3) - p(\pi_2, 3), 0\right\} + p(\pi_2, 2)$$

$$E(\pi_2^F, 2, 3) = \max\left\{E(\pi_3^F, 2, 3) - p(2, 3), 0\right\} + p(2, 2)$$

$$E(\pi_2^F, 2, 3) = \max\{(2-3), 0\} + 3 = 3$$

$$E\left(\pi_j^F, k, k+1\right) = \max\left\{E(\pi_{j+1}^F, k, k+1) - p(\pi_j, k+1), 0\right\} + p(\pi_j, k), \quad j = 1 \text{ and } k = 1, 2$$

$$E(\pi_1^F, 1, 2) = \max\left\{E(\pi_2^F, 1, 2) - p(\pi_1, 2), 0\right\} + p(\pi_1, 1)$$

$$E(\pi_1^F, 1, 2) = \max\left\{E(\pi_2^F, 1, 2) - p(1, 2), 0\right\} + p(1, 1)$$

$$E(\pi_1^F, 1, 2) = \max\{(2-1), 0\} + 4 = 5$$

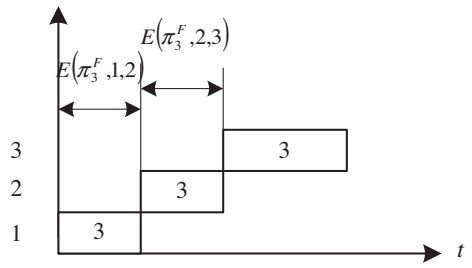

Fig. 2a. Computation of $E(\pi_3^F, k, k+1)$.

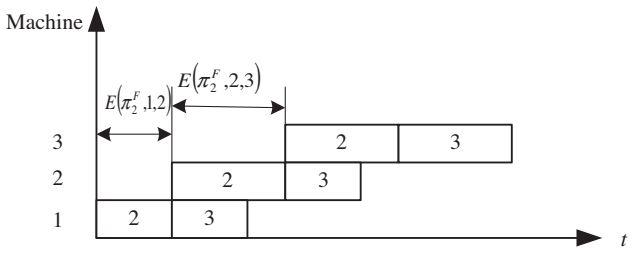

Fig. 2b. Computation of $E(\pi_2^F, k, k+1)$.

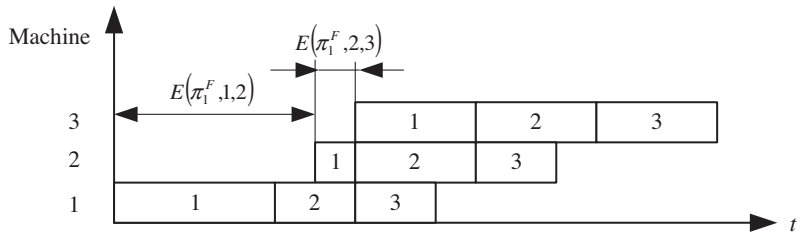

Fig. 2c. Computation of $E(\pi_1^F, k, k+1)$.

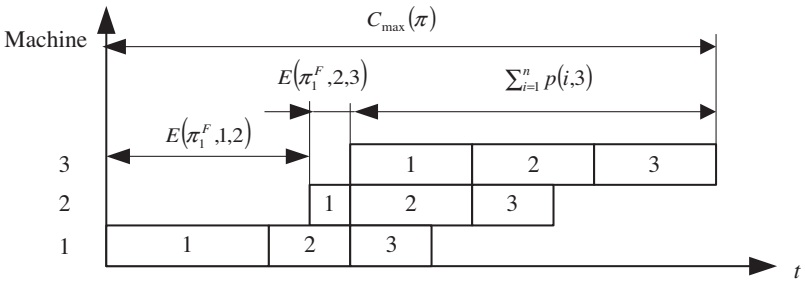

Fig. 2d. Computation of $C_{\max}(\pi)$.

$$E(\pi_1^F,2,3) = \max\{E(\pi_2^F,2,3) - p(\pi_1,3), 0\} + p(\pi_1,2)$$

$$E(\pi_1^F,2,3) = \max\{E(\pi_2^F,2,3) - p(1,3), 0\} + p(1,2)$$

$$E(\pi_1^F,2,3) = \max\{(3-3), 0\} + 1 = 1$$

$$C(\pi_1,3) = \sum_{k=1}^{2} E(\pi_1^F,k,k+1) + p(\pi_1,3)$$

$$C(\pi_1,3) = E(\pi_1^F,1,2) + E(\pi_1^F,2,3) + p(\pi_1,3)$$

$$C(\pi_1,3) = 5 + 1 + 3 = 9$$

$$C(\pi_2,3) = C(\pi_1,3) + p(2,3)$$

$$C(\pi_2,3) = 9 + 3 = 12$$

$$C(\pi_3,3) = C(\pi_2,3) + p(3,3)$$

$$C(\pi_3,3) = 12 + 3 = 15$$

$$T=\sum_{j=1}^{3}(\max(C(\pi_j,3)-d(\pi_j),0))$$

$$T=\max(C(\pi_1,3)-d(\pi_1),0)+\max(C(\pi_2,3)-d(\pi_2),0)+\max(C(\pi_3,3)-d(\pi_3),0)$$

$$T=\max(9-8,0)+\max(12-8,0)+\max(15-7,0)=1+4+8=13$$

Therefore, the objective of the NIPFS problem with the total tardiness criterion is to find a permutation π^* in the set of all permutations Π such that

$$T(\pi^*)\leqslant T(\pi_n^E)\ or\ T(\pi^*)\leqslant T(\pi_1^F),\quad \forall\pi\in\Pi. \tag{11}$$

Further to be used in the speed-up method, let $F(\pi_j^F,k,k+1)$ represents the lower bound for the minimum difference between the completion of processing the last job of π^F on machine $k+1$ and k which is restricted by the no-idle constraint. Then, $F(\pi_j^F,k,k+1)$ can be calculated as follows:

$$F(\pi_n^F,k,k+1)=p(\pi_n,k+1),\quad k=1,2,\ldots,m-1 \tag{12}$$

$$F(\pi_{n-1}^F,k,k+1)=\max\{p(\pi_{n-1},k+1)-E(\pi_n^F,k,k+1),0\}+F(\pi_n^F,k,k+1)\quad k=1,2,\ldots,m-1 \tag{13}$$

$$F\left(\pi_j^F,k,k+1\right)=\max\{p(\pi_j,k+1)-E\left(\pi_{j+1}^F,k,k+1\right),0\}+F\left(\pi_{j+1}^F,k,k+1\right),\quad j=n-1,n-2,\ldots,1,\ k=1,2,\ldots,m-1 \tag{14}$$

Figs. 3a-3c and d illustrate the calculation of makespan for a 3-job 3-machine problem.

Using the same example instance for 3-job 3-machine problem, the above calculations are illustrated in detail below:

$$F(\pi_n^F,k,k+1)=p(\pi_n,k+1)$$

$$F(\pi_3^F,1,2)=p(\pi_3,2)=p(3,2)=2\quad j=3\ and\ k=1$$

$$F(\pi_3^F,2,3)=p(\pi_3,3)=p(3,3)=3\quad j=3\ and\ k=2$$

$$F\left(\pi_j^F,k,k+1\right)=\max\{p(\pi_j,j+1)-E\left(\pi_{j+1}^F,k,k+1\right),0\}+F\left(\pi_{j+1}^F,k,k+1\right)\quad j=2\ and\ k=1$$

$$F(\pi_2^F,1,2)=\max\{p(\pi_2,2)-E(\pi_3^F,1,2),0\}+F(\pi_3^F,1,2)$$

$$F(\pi_2^F,1,2)=\max\{p(2,2)-E(\pi_3^F,1,2),0\}+F(\pi_3^F,1,2)$$

$$F(\pi_2^F,1,2)=\max\{(3-2),0\}+2=3$$

$$F(\pi_2^F,2,3)=\max\{p(\pi_2,3)-E(\pi_3^F,2,3),0\}+F(\pi_3^F,2,3),\quad j=2\ and\ k=2$$

$$F(\pi_2^F,1,2)=\max\{p(2,3)-E(\pi_3^F,2,3),0\}+F(\pi_3^F,2,3)$$

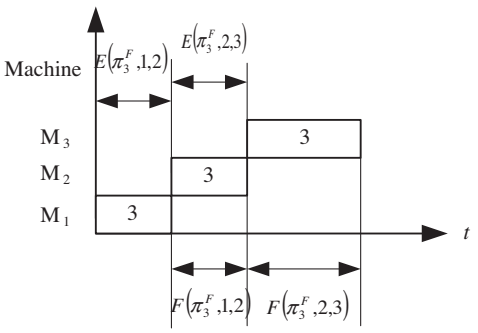

Fig. 3a. Computation of $F(\pi_3^F,k,k+1)$.

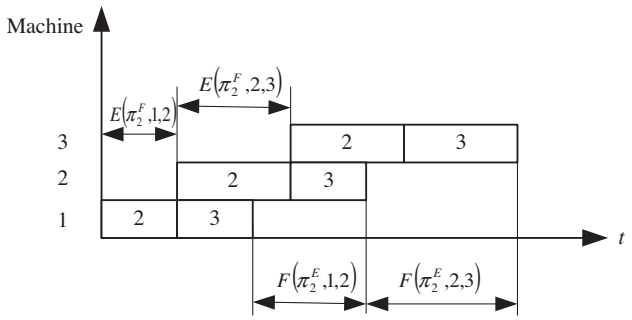

Fig. 3b. Computation of $F(\pi_2^E, k, k+1)$.

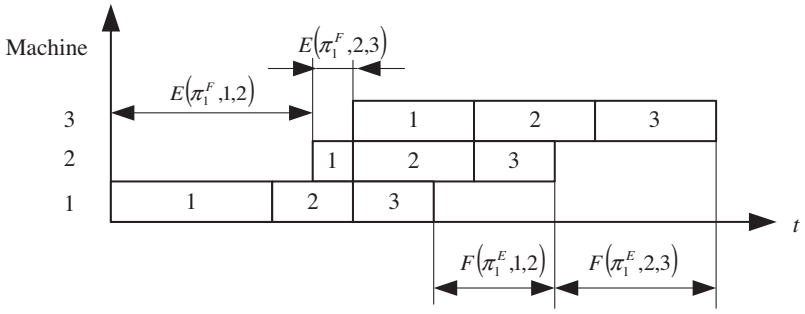

Fig. 3c. Computation of $F(\pi_1^E, k, k+1)$.

$$F(\pi_2^F,1,2) = \max\{(3-2),0\} + 3 = 4$$

$$F\left(\pi_j^F,k,k+1\right) = \max\left\{p(\pi_j,k+1) - E\left(\pi_{j+1}^F,k,k+1\right),0\right\} + F\left(\pi_{j+1}^F,k,k+1\right), \quad j=1 \text{ and } k=1$$

$$F(\pi_1^F,1,2) = \max\left\{p(\pi_1,2) - E(\pi_2^F,1,2),0\right\} + F(\pi_2^F,1,2)$$

$$F(\pi_1^F,1,2) = \max\{p(1,2) - E(\pi_2^F,1,2),0\} + F(\pi_2^F,1,2)$$

$$F(\pi_1^F,1,2) = \max\{(1-2),0\} + 3 = 3$$

$$F(\pi_1^F,2,3) = \max\left\{p(\pi_1,3) - E(\pi_2^F,2,3),0\right\} + F(\pi_2^F,2,3), \quad j=1 \text{ and } k=2$$

$$F(\pi_1^F,2,3) = \max\left\{p(1,3) - E(\pi_2^F,2,3),0\right\} + F(\pi_2^F,2,3)$$

$$F(\pi_1^F,2,3) = \max\{(3-3),0\} + 4 = 4$$

2.3. Speed-up method for insertion neighborhood

Insertion neighborhood of a job permutation π is widely used for flow shop scheduling problems in the literature [35] which is defined by considering all possible insertion moves $m(u, v)$, $u, v \in \{1, 2, \ldots, n\}$. The insert move $m(u, v)$ generates a permutation $d\pi$ from π by removing a job of π from its original position u and inserting it into position v ($v \notin (u, u-1)$)

$$d\pi = \{\pi_1,\ldots,\pi_{u-1},\pi_{u+1},\ldots,\pi_v,\pi_u,\pi_{v+1},\ldots,\pi_n\} \quad \text{if } u < v \tag{15}$$

$$d\pi = \{\pi_1,\ldots,\pi_{v-1},\pi_u,\pi_{v+1},\ldots,\pi_{u-1},\pi_{u+1},,\ldots,\pi_n\} \quad \text{if } u > v \tag{16}$$

Based on the similarity of $d\pi$ and π, a short cut to evaluate the insert neighborhood can be developed by following the reduction of computational complexity presented by [25] for the PFSP with makespan criterion. Then it is trivial to extend it to the total tardiness criterion as explained in the forward and backward pass calculations. The following speed-up method can be used to evaluate the insert neighborhood for the total tardiness criterion:

1. Let $t = 1$.
2. Let $\Delta\pi = \{\pi_1, \pi_2, \ldots, \pi_{n-1}\}$ be a partial permutation generated by removing job π_t from permutation π.
   a. Compute $F\left(\Delta\pi_j^E, k, k+1\right)$, $E\left(\Delta\pi_j^F, k, k+1\right)$, and $F\left(\Delta\pi_j^F, k, k+1\right)$, respectively.
   b. Let $F(\Delta\pi_n^F, k, k+1) = E(\Delta\pi_n^F, k, k+1) = 0$ for $j \in \{1, 2, \ldots, n-1\}$ and $k \in (1, 2, \ldots, m-1)$.
3. Repeat the following steps until all possible positions h of $\Delta\pi = \{\pi_1, \pi_2, \ldots, \pi_{n-1}\}$ are considered such that $h \in \{1, 2, \ldots, n\} \wedge h \notin \{t, t-1\}$.
   a. Let $\Delta\pi_h^E = \Delta\pi_{h-1}^E \cup \pi_t$. Note that $\Delta\pi_0^E = \phi$. Calculate $F(\Delta\pi_h^E, k, k+1)$ for $k = 1, 2, \ldots, m-1$.
   b. Let $c\pi = \Delta\pi_h^E \cup \Delta\pi_h^F = \{c\pi_1, c\pi_2, \cdots, c\pi_n\}$ (Note that $\Delta\pi_n^F = \phi$), then $F(d\pi, k, k+1)$ can be given by $F(c\pi, k, k+1) = \max\{F(\Delta\pi_h^E, k, k+1) - E(\Delta\pi_h^F, k, k+1), 0\} + F(\Delta\pi_h^F, k, k+1)$, for $k = 1, 2, \ldots, m-1$.
   c. Then the completion time of job $c\pi_n$ on machine m is

$$C(c\pi_n, m) = \sum_{k=1}^{m-1} F(c\pi, k, k+1) + \sum_{j=1}^{n} p(c\pi_j, 1)$$

   d. The completion time of job $c\pi_j$ on the last machine m is

$$C(c\pi_j, m) = C(c\pi_{j+1}, m) - p(c\pi_{j+1}, m),\ j = n-1, n-2, \ldots, 1$$

   e. Then, the total tardiness is

$$T = \sum_{j=1}^{n} (\max(C(c\pi_j, m) - d(c\pi_j), 0))$$

3. Let $t = t + 1$. If $t > n$ then stop; otherwise go back to step 2.

There are n iterations for Step 2 and Step 3, and both Step2 and Step 3 can be executed in time $O(mn)$. So, the computational complexity of this speed-up method is $O(mn^2)$ to evaluate the whole insert neighborhood of a permutation. The sped-up method is illustrated with the following example with an example instance for 4-job 3-machine problem is given in Table 2:

Step 1. Suppose that the permutation is given by $\pi = \{1, 2, 3, 4\}$, Let $t = 1$.

Step 2. Let $\Delta\pi = \{2, 3, 4\}$ be a partial permutation generated by removing job 1 from π. Then

a. Calculate $F\left(\Delta\pi_j^E, k, k+1\right)$, $E\left(\Delta\pi_j^F, k, k+1\right)$, and $F\left(\Delta\pi_j^F, k, k+1\right)$, respectively.

$$F(\Delta\pi_1^E, 1, 2) = 1F(\Delta\pi_1^E, 2, 3) = 3F(\Delta\pi_2^E, 1, 2) = 3F(\Delta\pi_2^E, 2, 3) = 3F(\Delta\pi_3^E, 1, 2) = 3F(\Delta\pi_3^E, 2, 3) = 4;$$

$$E(\Delta\pi_3^F, 1, 2) = 2E(\Delta\pi_3^F, 2, 3) = 2E(\Delta\pi_2^F, 1, 2) = 2E(\Delta\pi_2^F, 2, 3) = 3E(\Delta\pi_1^F, 1, 2) = 5E(\Delta\pi_1^F, 2, 3) = 1;$$

$$F(\Delta\pi_3^F, 1, 2) = 2F(\Delta\pi_3^F, 2, 3) = 3F(\Delta\pi_2^F, 1, 2) = 3F(\Delta\pi_2^F, 2, 3) = 4E(\Delta\pi_1^F, 1, 2) = 3E(\Delta\pi_1^F, 2, 3) = 4$$

a. Let $F(\Delta\pi_4^F, k, k+1) = E(\Delta\pi_4^F, k, k+1) = 0$, $k = 1, 2$.

Step 3. Repeat the following steps until all possible positions h of $\Delta\pi = \{\pi_1, \pi_2, \ldots, \pi_{n-1}\}$ are considered such that $h \in \{1, 2, \ldots, n\} \wedge h \notin \{t, t-1\}$.

Case 1: Insert job 1 into position 2 of $\Delta\pi$:

a. Let $\Delta\pi_2^E = \Delta\pi_1^E \cup \pi_t = \{2, 1\}$, then $F(\Delta\pi_2^E, 1, 2) = 3F(\Delta\pi_2^E, 2, 3) = 2$
b. Let $c\pi = \Delta\pi_2^E \cup \Delta\pi_2^F = \{2, 1, 3, 4\}$ then

$$F(c\pi, 1, 2) = \max\{F(\Delta\pi_2^E, 1, 2) - E(\Delta\pi_2^F, 1, 2), 0\} + F(\Delta\pi_2^F, 1, 2) = \max\{3-2, 0\} + 3 = 4$$

$$F(c\pi, 2, 3) = \max\{F(\Delta\pi_2^E, 2, 3) - E(\Delta\pi_2^F, 2, 3), 0\} + F(\Delta\pi_2^F, 2, 3) = \max\{2-3, 0\} + 4 = 4$$

Table 2
An Example Instance.

Job(j)	*Machine*(k)			Due Date
	1	2	3	$d_j = \tau \times \sum_{k=1}^{m} p_{jk}$
p_{1j}	3	3	2	8
p_{2j}	4	1	3	8
p_{3j}	2	3	3	8
p_{4j}	2	2	3	7

c. The completion time of job $c\pi_4$ on machine 3 is
$C(c\pi_4,3) = F(c\pi,1,2) + F(c\pi,2,3) + 11 = 19$

d. The completion time of job $c\pi_3$ on machine 3 is
$C(c\pi_3, 3) = C(c\pi_4, 3) - p(c\pi_4, 3) = 19 - 3 = 16,$

e. The completion time of job $c\pi_2$ on machine 3 is

$$C(c\pi_2,3) = C(c\pi_3,3) - p(c\pi_3,3) = 16 - 3 = 13$$

f. The completion time of job $c\pi_1$ on machine 3 is
$C(c\pi_1,3) = C(c\pi_2,3) - p(c\pi_2,3) = 13 - 2 = 11$

g. Finally, the total tardiness is

$$T = \sum_{j=1}^{4} (\max(C(c\pi_j,3) - d(c\pi_j),0))$$

$$T = \max(11-8,0) + \max(13-8,0) + \max(16-8,0) + \max(19-7,0) = 3+5+8+12 = 28$$

Case 2: Insert job 1 into position 3 of $\Delta\pi$:

a. let $\Delta\pi_3^E = \Delta\pi_2^E \cup \pi_t = \{2,3,1\}$, then $F(\Delta\pi_3^E,1,2) = 3F(\Delta\pi_3^E,2,3) = 2$

b. Let $c\pi = \Delta\pi_3^E \cup \Delta\pi_3^F = \{2,3,1,4\}$ then

$$F(c\pi,1,2) = \max\{F(\Delta\pi_3^E,1,2) - E(\Delta\pi_3^F,1,2),0\} + F(\Delta\pi_3^F,1,2) = \max\{3-2,0\} + 2 = 3$$

$$F(c\pi,2,3) = \max\{F(\Delta\pi_3^E,2,3) - E(\Delta\pi_3^F,2,3),0\} + F(\Delta\pi_3^F,2,3) = \max\{2-2,0\} + 3 = 3$$

c. The completion time of job $c\pi_4$ on machine 3 is

$$C(c\pi_4,3) = F(c\pi,1,2) + F(c\pi,2,3) + 11 = 17$$

d. The completion time of job $c\pi_3$ on machine 3 is $C(c\pi_3, 3) =$
$C(c\pi_4, 3) - p(c\pi_4, 3) = 17 - 3 = 14$

e. The completion time of job $c\pi_2$ on machine 3 is

$$C(c\pi_2,3) = C(c\pi_3,3) - p(c\pi_3,3) = 14 - 2 = 12$$

f. The completion time of job $c\pi_1$ on machine 3 is

$$C(c\pi_1,3) = C(c\pi_2,3) - p(c\pi_2,3) = 12 - 3 = 9$$

g. Finally, the total tardiness is

$$T = \sum_{j=1}^{4} (\max(C(c\pi_i,3) - d(c\pi_i),0))$$

$$T = \max(9-8,0) + \max(12-8,0) + \max(14-8,0) + \max(17-7,0) = 1+4+6+10 = 21$$

Case 3: Insert job 1 into position 4 of $\Delta\pi$:

a. Let $\Delta\pi_4^E = \Delta\pi_3^E \cup \pi_t = \{2,3,4,1\}$, then $F(\Delta\pi_4^E,1,2) = 3F(\Delta\pi_4^E,2,3) = 3$
b. Let $c\pi = \Delta\pi_4^E \cup \Delta\pi_4^F = \{2,3,4,1\}$ then

$$F(d\pi,1,2) = \max\left\{F(\Delta\pi_4^E,1,2) - E(\Delta\pi_4^F,1,2),0\right\} + F(\Delta\pi_4^F,1,2) = \max\{3-0,0\} + 0 = 3$$

$$F(d\pi,2,3) = \max\left\{F(\Delta\pi_4^E,2,3) - E(\Delta\pi_4^F,2,3),0\right\} + F(\Delta\pi_4^F,2,3) = \max\{3-0,0\} + 0 = 3$$

c. The completion time of job $c\pi_4$ on machine 3 is
$C(c\pi_4,3) = F(c\pi,1,2) + F(c\pi,2,3) + 11 = 17$

d. The completion time of job $c\pi_3$ on machine 3 is
$C(c\pi_3, 3) = C(c\pi_4, 3) - p(c\pi_4, 3) = 17 - 2 = 15$,

e. The completion time of job $c\pi_2$ on machine 3 is

$$C(c\pi_2, 3) = C(c\pi_3, 3) - p(c\pi_3, 3) = 15 - 3 = 12$$

f. The completion time of job $c\pi_1$ on machine 3 is
$C(c\pi_1, 3) = C(c\pi_2, 3) - p(c\pi_2, 3) = 12 - 3 = 9$

g. Finally, the total tardiness is

$$T = \sum_{j=1}^{4} (\max(C(c\pi_i, 3) - d(c\pi_i), 0))$$

$$T = \max(9 - 8, 0) + \max(12 - 8, 0) + \max(15 - 7, 0) + \max(17 - 8, 0) = 1 + 4 + 8 + 9 = 22$$

Step 4. Let $t = 2$. If $t > n$ then stop; otherwise go back to step 2.

3. Discrete artificial bee colony algorithm

The artificial bee colony (ABC) algorithm is a new swarm intelligence based optimizer proposed in [28–32] for multi-variable 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-dimensional real-valued vector, whereas the fitness of the solution corresponds 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 steps are repeated until a termination criterion is met [28–32]:

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

It is obvious that the above ABC algorithm cannot be used for a discrete/combinatorial optimization problem due to its continuous nature. We follow the same structure for the discrete version described in Section 3.1.

3.1. Initialization

The permutation representation is used in the GA and DABC algorithms. In other words, a solution is represented by a permutation of jobs $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$. There are NP numbers of individuals in the population. The initial population in the GA and DABC algorithms is constructed in such a way that the first solution is established by the NEH heuristic of [21] and the rest of the solutions are constructed randomly.

The NEH heuristic has two phases. In phase I, jobs are ordered in descending sums of their processing times. In phase II, a job permutation is established by evaluating the partial permutations based on the initial order of the first phase. Suppose a current permutation is already determined for the first k jobs, $k + 1$ partial permutations are constructed by inserting job $k + 1$ in $k + 1$ possible slots of the current permutation. Among these $k + 1$ permutations, the one generating the minimum total tardiness is kept as the current permutation for the next iteration. Then job $k + 2$ from phase I is considered and so on until all jobs have been sequenced.

3.2. 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 yield neighboring solutions in the literature [2,35]. The insert operator of a permutation π is defined by removing a job π from its original position j and inserts it into another position p such that $(p \in \{j, j-1\})$, whereas the swap operator produces a neighbor by interchanging two jobs of π. To enrich the neighborhood structure and diversify the population, six neighboring strategies, denoted as S_i, based on the insert and swap operators with the perturbation strength p as well as the destruction and

construction procedure with the destruction size of d, denoted as *DestructConstruct*(), of IG algorithm in [26] are separately utilized to generate neighboring food sources for the employed bees as follows:

S_1: Apply one insert move ($p = 1$) to a permutation π.
S_2: Apply one swap move ($p = 1$) to a permutation π.
S_3: Apply two insert moves ($p = 2$) to a permutation π.
S_4: Apply two swap moves ($p = 2$) to a permutation π.
S_5: Apply three insert moves ($p = 3$) to a permutation π.
S_5: Apply three swap moves ($p = 3$) to a permutation π.
S_6: Apply a *DestructConstruct*() with the destruction size of $d = 4$

Each method for the generation of neighboring food sources may have different performance during the evolution process. Therefore, each food source (individual) in the population is assigned to one of the six strategies to generate a neighboring food source. After generating a neighboring food source, a local search is applied to further improve the solution quality (nectar amount) with a small probability of $p_{LS} = 0.01$. As for the selection, new good source is always accepted if it is better than the current food source, which is similar to the basic ABC algorithm carrying out a greedy selection procedure.

Regarding the *DestructConstruct*() procedure, in the destruction step, a given number d of jobs, randomly chosen and without repetition, are removed from the solution, thus resulting in two partial solutions. The first one with the size d of jobs is denoted as π^R and includes the removed jobs in the order in which they are removed. The second one with the size $n - d$ of jobs is the original solution without the removed jobs, which is denoted as π^D. Finally, the construction phase requires a constructive heuristic procedure. We employ the NEH insertion heuristic. In order to reinsert jobs in π^R into the destructed solution π^D, the first jobs π^R_1 is inserted into all possible $n - d$ positions in the destructed solution π^D generating $n - d$ partial solutions. Among these $n - d$ partial solutions including job π^R_1, the best partial solution with the minimum total tardiness is chosen and kept for the next iteration. Then the second job π^R_2 is considered and so on until π^R is empty or a final solution is obtained. Hence π^D is again of size. For the details of *DestructConstruct*() procedure, we refer to Ruiz and Stützle [26] where it is well illustrated with an example instance for the makespan criterion.

The motivation for assigning one of the six strategies to each individual in the population is due to the fact that the DABC algorithm works like a multi-populated algorithm using a different strategy in each sub-population. By employing these strategies, the DABC algorithm implicitly takes advantage of the IG algorithm of [26] and the iterated local search (ILS) algorithm of [36], respectively. In the former one, the destruction size (d) is a parameter to be carefully chosen whereas in the latter one, the perturbation strength (p) should be determined with care. The perturbation can be achieved by removing a job from a position and inserting it into another position randomly or swapping of any two jobs randomly. In the original IG algorithm of [26], the destruction size of $d = 4$ is suggested for the makespan criterion after a detailed design of experiments. For this reason, we use a *DestructConstruct*() procedure with the destruction size of $d = 4$. Regarding the perturbation strength of ILS algorithm, in [36], p values ranging from 1 to 20 were tested and p values between 4 and 7 were suggested. However, in our experiments, p values within the range from 1 to 3 swap or insert moves generated better results.

The size of the employed bees is set to the population size *NP*. The local search procedure will be explained in detail in Section 3.4.

3.3. Onlooker bee phase

In the basic ABC algorithm, an onlooker bee selects a food source π_k depending on its winning probability value which is similar to the wheel selection in GAs [2]. However, the tournament selection is widely used in GA applications due to its simplicity and ability to escape from local optima [2]. 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 π_k in such a way that two food sources are randomly picked up from the population, and compared to each other, then the better one is chosen. In addition, an onlooker bee utilizes the same strategy as used by the employed bee to produce a new neighboring solution. Then, a local search is employed to further improve the nectar amount of the onlooker bee. If the new food source obtained is better than or equal to the current one, the new food source will be replaced by the current one and become a new member in the population. The onlooker bee phase in the DABC algorithm provides the intensification of local search on the relatively good solutions chosen with a tournament size of 2. The aim is to improve the solution quality of promising solutions in the population. This is achieved by first applying the assigned strategy to a food source π_k and then a very effective local search later on.

The size of the onlooker bees is $2 * NP$ individuals. The local search procedure will be explained in detail in Section 4.

3.4. Scout bee phase

In the basic ABC algorithm, a scout bee produces a food source randomly in the predefined search space. 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,

a tournament selection with the size of 2 is again used to discard a solution in such a way that two random food sources are picked up and the worst one is selected. Then the scout generates a food source by performing a destruction and construction procedure with a destruction size of $d = 4$ to the best solution in the population. This destructed and constructed solution will be replaced by the food source determined by tournament selection. Thus, poor solutions in the population will be replaced by the perturbations of the best so far solution. For this reason, the scout bee phase serves for additional diversification by perturbations of the best so far solution in the population that will be replaced by a relatively small number of unpromising solutions in the population. The size of the scout bees is $0.1 * NP$ individuals.

As for the DABC algorithm, the following computational procedure is used as given in Fig. 4.

1. Set the population size NP, $S_{\max}$, p_{LS} S_i for each food source.
2. Initialize the population:
   a. The first one is by NEH whereas others are randomly established.
   b. $\pi_1 = NEH(\pi_1)$
   c. $\pi = \{\pi_2, \pi_2, \ldots, \pi_{NP}\}$ and evaluate each solution in the population.
3. Employed bee phase:

   For $i = 1, 2, \ldots, NP$, repeat the following sub-steps:

   a. Produce a new food source u_i for the i^{th} employed bee who is associated with the strategy S_i and evaluate the new solution.
   b. If $r < p_{LS}$, perform *LocalSearch*() procedure to u_i.
   c. If u_i is better than π_i, let $\pi_i = u_i$ and update best so far solution π_B.
4. Onlooker bee phase.

   For $i = 1, 2, \ldots, 2 \times NP$, repeat the following sub-steps:

   a. Select a food source π_k in the population for the onlooker bee by using the tournament selection with size of 2 (Better one is chosen)
   b. Generate a new solution u_k for the onlooker bee by using the S_k and apply *LocalSearch*().
   c. If u_k is better than π_k, let $\pi_k = u_k$ and update the best so far solution π_B.
5. Scout phase.

   a. A tournament selection with the size of 2 is again used to discard a solution in such a way that two random food sources are picked up from population and the worst one is selected.
   b. Then the scout generates a food source by performing a *DestructConstruct*() procedure with a destruction size of $d = 4$ to the best so far solution π_B in the population and the obtained solution is replaced with the food source determined by the tournament selection. The size of the scout bees is $0.2 * NP$ individuals.
   c. Scout phase.
6. Memorize the best solution achieved so far.
7. If the termination criterion is reached, return the best solution found so far; otherwise go to Step 4.

3.5. Local search

Regarding the local search algorithm denoted as *LocalSearch*(), we use the insertion neighborhood structure since we develop a speed-up method for it. Insertion neighborhood is iteratively applied in the *LocalSearch*() procedure. In the insertion neighborhood, a job is removed from its original position, and then it is inserted to all possible positions of the remaining jobs. Of course, the better one is chosen and compared to an incumbent solution to update if better. This process is applied as long as the incumbent solution improves as shown in Fig. 5.

4. Proposed genetic algorithm

Genetic algorithms (GA) are a family of parallel search heuristics inspired by the biological process of natural selection and evolution [2]. In GA optimization, solutions are encoded into chromosomes to establish a population being evolved through generations. At each generation, parents are selected and mated from the population to carry out the crossover operator leading to new solutions called children. Then, some of the individuals are mutated or perturbed. Finally, they are pooled together to select new individuals for next generation. This procedure is repeated until the stopping criterion is achieved.

However, in the proposed GA, we employ multiple crossover strategies to enhance the solution quality. Basically, we employ three crossover strategies as follows:

S_1: PTL crossover in [37].

```
procedure DABC
π = [π_1, π_2, ..., π_NP]
π_1 = NEH(π_1)
S_i = rand()%S_max
π_B = arg min_{i=1,2,..,NP} (π_i)
do
// Employed Bee Phase
    u_i = π_i   (i=1,2,..,NP)
    u_i = S_i(u_i)   (i=1,2,..,NP)
    if (r < p_LS)
        u_i = LocalSearch(u_i)   (i=1,2,..,NP)
        if (f(u_i) < f(π_i))   (i=1,2,..,NP)
            π_i = u_i   (i=1,2,..,NP)
            if (f(u_i) < f(π_B))   (i=1,2,..,NP)
                π_B = u_i   (i=1,2,..,NP)
            endif
        endif
    endif
// Onlooker Bee Phase
    π_k = TournamentSelect(π ∈ NP)   (i=1,2,..,2*NP)
    u_k = S_k(π_k)
    u_k = LocalSearch(u_k)   (i=1,2,..,NP)
    if (f(u_k) < f(π_i))   (i=1,2,..,NP)
        π_i = u_k   (i=1,2,..,2*NP)
        if (f(u_k) < f(π_B))   (i=1,2,..,2*NP)
            π_B = u_k   (i=1,2,..,2*NP)
    endif
// Scout Bee Phase
    π_k = TournamentSelect(π ∈ NP)   (i=1,2,..,0.1*NP)
    u_k = DestructConstruct(π_B)
    π_k = u_k
while(NotTermination)
return π_B
endprocedure
```

Fig. 4. DABC algorithm.

S_2: OX crossover in [38].
S_3: PMX crossover in [39].

```
procedure LocalSearch(π)
π0 = π
k = 1
t = 1;
while(t < n)
    k = (k + 1)%n
    π1 = remove job πk from π0
    π2 = best permutation obtained by inserting πk in all possible positions of π1
    if (f(π2) < f(π0))
        π0 = π2
        t = 1
    else
        t = t + 1
    endif
endwhile
π = π0
return π
endprocedure
```

Fig. 5. Pseudo code for the local search algorithm.

The proposed GA algorithm can be summarized as follows. The initial population is constructed in such a way that the first individual is constructed by NEH heuristic and the rest is randomly established. In the proposed GA, each individual is assigned to one of the three crossover strategies randomly. At each generation, for the in individual in the population, a mate is chosen by the tournament selection with size 2 from the population to produce an offspring through the use of the associated crossover strategy. This process is conducted in a loop until another NP offspring are produced. Some of the offspring generated are mutated with an insert move. Then, a local search is applied to each offspring generated. For the population of the next generation, one to one competition is used in a way that the better one is kept in the population, thus maintaining again a size NP of population. This procedure is repeated until the stopping criterion is achieved. The following computational procedure explains the components of the proposed GA:

1. Set the population size NP, $S_{\max}$.
2. Assign a crossover strategy S_i for each individual in the population randomly.
3. Initialize the population:
   a. The first one is by NEH whereas others are randomly established.
   i. $\pi_1 = NEH(\pi_1)$
   ii. $\pi = \{\pi_2, \pi_2, \ldots, \pi_{NP}\}$ and evaluate each solution in the population.
4. For $i = 1, 2, \ldots, NP$, repeat the following sub-steps:
   a. For the individual π_i, select a mate π_k from the population by the tournament selection with size 2.
   b. Produce a new offspring u_i by recombining them with the strategy $S_i = (\pi_i, \pi_k)$.
   c. Mutate u_i with a mutation probability.
   d. Evaluate the new offspring u_i and apply *LocalSearch*() to u_i.
   e. If u_i is better than π_i, let $\pi_i = u_i$ and update best so far solution π_B.
5. If the termination criterion is reached, return the best solution found so far π_B; otherwise go to Step 3.

5. Computational results

The GA and DABC algorithms were coded in Visual C++ and run on an Intel Pentium IV 3.0 GHz PC with 512 MB memory. They were applied to the 120 benchmark instances of Taillard in [27] ranging from 20 jobs with 5 machines to 500 jobs with 20 machines. All the parameters in this study are determined experimentally. Regarding the parameters of the GA algorithm, the population size is fixed at $NP = 100$. The crossover and mutation probabilities are taken as 1.0 and 0.05, respectively. The PTL, OX and PMX crossover operators are employed as mentioned before. As to the parameters of the DABC algorithm, again the population size is fixed at $NP = 100$. The sizes of employed bees, onlooker bees and scout bees are NP, $2 \times NP$ and $0.1 \times NP$, respectively. Strategies are determined as explained in Section 2. Five ($R = 5$) runs were carried out for each

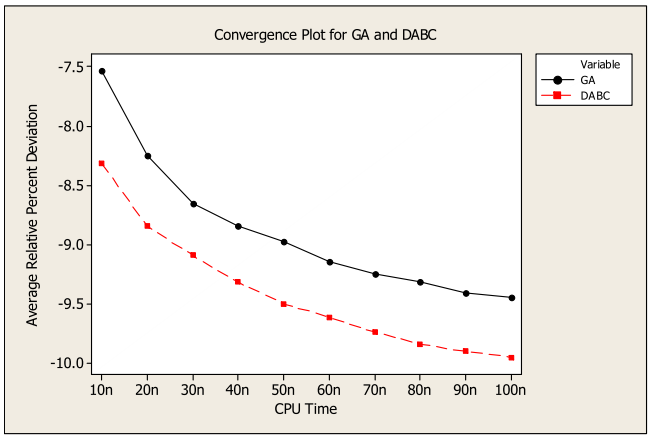

Fig. 6. Convergence plot of GA vs DABC.

Table 3
Computational results of GA and DABC: $T_{max} = 100 \times n$, $\tau = 1$.

	GA				DABC			
	Avg	Min	Max	Std	Avg	Min	Max	Std
20×5	−10.95	−11.52	−9.82	0.72	−11.29	−11.58	−10.81	0.36
20×10	−12.51	−13.18	−11.60	0.66	−12.99	−13.26	−12.50	0.33
20×20	−13.50	−13.95	−12.83	0.49	−13.93	−14.06	−13.71	0.17
50×5	−10.06	−10.55	−9.39	0.46	−10.65	−11.09	−10.18	0.37
50×10	−12.85	−13.68	−11.97	0.70	−13.54	−14.20	−12.97	0.51
50×20	−12.76	−13.66	−11.79	0.77	−13.63	−14.22	−12.82	0.57
100×5	−6.40	−7.15	−5.72	0.57	−6.80	−7.33	−6.22	0.44
100×10	−8.12	−8.76	−7.40	0.55	−8.80	−9.38	−8.26	0.46
100×20	−9.60	−10.43	−8.81	0.66	−10.33	−11.08	−9.67	0.56
200×10	−5.53	−6.39	−4.80	0.63	−5.37	−5.72	−4.92	0.34
200×20	−7.94	−8.66	−7.32	0.54	−8.14	−8.78	−7.63	0.47
500×20	−3.15	−3.93	−2.52	0.58	−3.96	−4.50	−3.49	0.40
Avg	−9.45	−10.16	−8.66	0.61	−9.95	−10.43	−9.43	0.42

problem instance. Each run was compared to the solution produced by the NEH algorithm modified for the NIPFS problem. The minimum (*Min*), average (*Avg*), maximum (*Max*) and standard deviation (*Std*) of five runs are reported. To be more specific, the average relative percent deviation from NEH solution is given as follows:

$$\Delta_{avg} = \sum_{i=1}^{R} \left(\frac{(H_i - NEH) \times 100}{NEH} \right) \Big/ R \tag{17}$$

where H_i, NEH, and R are the objective function values generated by any of heuristic algorithms in each run, the NEH solution value and the number of runs, respectively. As a termination criterion, both algorithms were run for $T_{max} = 10 \times n$ seconds to $T_{max} = 100 \times n$ seconds in order to obtain the convergence graphs of the GA and DABC algorithms. Then for the termination criterion of $T_{max} = 100 \times n$ seconds, the best known solutions are presented for three levels of due date tightness factor. The determination of due date for tardiness criterion is a major issue when dealing with tardiness criterion. There are several methods proposed for determining the due dates when tardiness criterion is considered. We choose a simple form which is called total work content rule (TWK) [40]. In the TWK rule, the due date of job j is determined by $d_j = \tau \times \sum_{k=1}^{m} p_{jk}$ where τ is a due date tightness factor and $\sum_{k=1}^{m} p_{jk}$ is the total processing time of job j on all the machines. To make the job due date loose, medium and tight, τ is taken as 1, 2 and 3 (tight, medium and loose, respectively).

5.1. Computational results for the tight due date

In this setting, $\tau = 1$ is considered. The convergence plot of both GA and DABC algorithms is given in Fig. 6 where it is clear that DABC algorithm is much faster to converge to global or local optima. The computational results for the termination criterion of $T_{max} = 100 \times n$ seconds are given in Table 3.

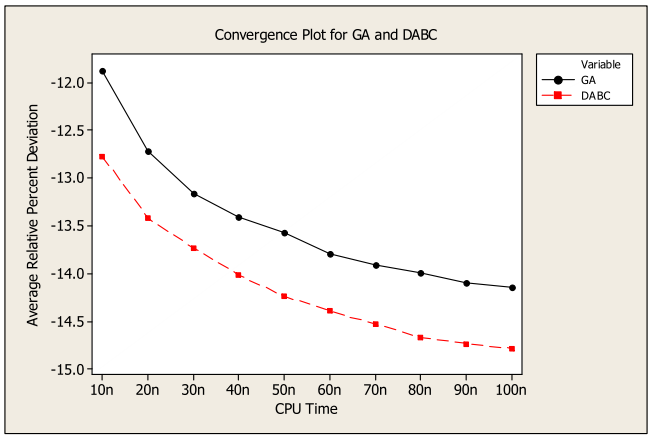

Fig. 7. Convergence plot of GA vs DABC.

Table 4
Computational results of GA and DABC: $T_{max} = 100 \times n$, $\tau = 2$.

	GA				DABC			
	Avg	Min	Max	Std	Avg	Min	Max	Std
20 × 5	−18.45	−18.88	−17.67	0.52	−18.72	−19.10	−18.32	0.35
20 × 10	−28.87	−29.49	−27.80	0.71	−29.33	−29.62	−28.96	0.31
20 × 20	−28.23	−29.04	−27.32	0.72	−29.10	−29.40	−28.37	0.44
50 × 5	−11.62	−12.28	−10.76	0.63	−12.36	−12.80	−11.87	0.40
50 × 10	−15.28	−17.06	−13.82	1.28	−16.57	−17.40	−15.77	0.65
50 × 20	−17.64	−18.89	−16.37	1.05	−18.94	−19.93	−18.13	0.71
100 × 5	−8.62	−9.36	−7.84	0.62	−9.14	−9.62	−8.52	0.47
100 × 10	−10.73	−11.39	−10.03	0.57	−11.46	−11.98	−10.98	0.40
100 × 20	−12.78	−13.76	−11.52	0.91	−13.58	−14.17	−12.83	0.55
200 × 10	−5.99	−6.68	−5.27	0.57	−5.70	−6.18	−5.00	0.47
200 × 20	−8.25	−9.01	−7.40	0.64	−8.36	−8.94	−7.92	0.42
500 × 20	−3.31	−4.13	−2.50	0.69	−4.19	−4.81	−3.70	0.47
Avg	−14.15	−15.00	−13.19	0.74	−14.79	−15.33	−14.20	0.47

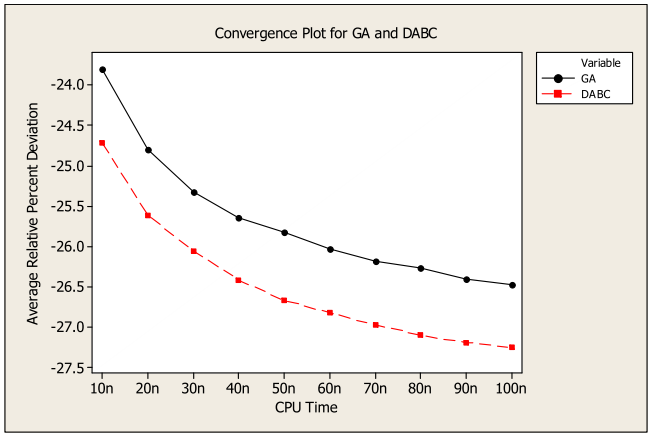

Fig. 8. Convergence plot of GA vs DABC.

From Table 3, it is obvious that the improvements of the DABC algorithm over the NEH solutions are higher than those in the GA algorithm. On overall average, the gap between the GA and DABC algorithm was 0.5% (9.95–9.45). Similar observations can also be made for Min and Max values too. In addition, a paired t-test confirms the significance on $\alpha = 0.05$ in differences between GA and DABC algorithm since the p-value was 0.000 on the behalf of DABC algorithm.

Table 5
Computational results of GA and DABC: $T_{max} = 100 \times n$, $\tau = 3$.

	GA				DABC			
	Avg	Min	Max	Std	Avg	Min	Max	Std
20 × 5	−41.60	−42.28	−40.69	0.76	−42.04	−42.28	−41.55	0.32
20 × 10	−68.94	−69.47	−68.08	0.58	−69.45	−69.54	−69.29	0.13
20 × 20	−81.86	−82.61	−80.21	1.05	−82.21	−82.61	−81.81	0.41
50 × 5	−14.90	−15.69	−14.05	0.66	−15.71	−16.28	−15.26	0.41
50 × 10	−23.22	−25.33	−20.73	1.89	−24.92	−26.30	−23.28	1.24
50 × 20	−30.26	−32.45	−28.12	1.75	−32.38	−34.05	−30.47	1.42
100 × 5	−9.42	−10.31	−8.36	0.76	−10.12	−10.86	−9.39	0.61
100 × 10	−11.90	−12.85	−10.94	0.77	−12.52	−13.24	−11.90	0.52
100 × 20	−15.53	−16.57	−14.30	0.95	−16.73	−17.54	−15.88	0.68
200 × 10	−6.67	−7.57	−5.81	0.73	−6.33	−7.16	−5.70	0.62
200 × 20	−9.44	−10.23	−8.68	0.62	−9.83	−10.65	−9.14	0.58
500 × 20	−3.87	−4.65	−3.15	0.60	−4.76	−5.29	−4.32	0.40
Avg	−26.47	−27.50	−25.26	0.93	−27.25	−27.98	−26.50	0.61

Table 6
Best known solutions for $T_{max} = 100 \times n$.

Ins	$\tau = 1$				$\tau = 2$				$\tau = 3$			
	NEH	GA	DABC		NEH	GA	DABC		NEH	GA	DABC	
20 × 5	13,321	11,967	11,967	0.00	8398	6941.00	6941	0.00	4098	2543	2543	0.00
20 × 5	12,956	10,752	10,764	0.11	8129	6053.00	6042	−0.18	4049	2423	2423	0.00
20 × 5	13,918	12,048	12,041	−0.06	9253	7463.00	7463	0.00	4858	3637	3637	0.00
20 × 5	12,527	11,163	11,163	0.00	6603	5805.00	5784	−0.36	3987	1901	1901	0.00
20 × 5	14,331	12,812	12,812	0.00	9512	7844.00	7844	0.00	5562	3174	3174	0.00
20 × 5	14,589	13,264	13,264	0.00	10252	8217.00	8209	−0.10	5612	3592	3592	0.00
20 × 5	10,777	9087	9017	−0.77	5685	4395.00	4395	0.00	2003	1036	1036	0.00
20 × 5	11,345	10,564	10,564	0.00	6756	5550.00	5542	−0.14	2924	1657	1657	0.00
20 × 5	13,575	11,887	11,887	0.00	8085	6771.00	6645	−1.86	4640	2618	2618	0.00
20 × 5	11,053	10,057	10,057	0.00	6703	5341.00	5341	0.00	3560	1670	1670	0.00
20 × 10	24,219	22,317	22,317	0.00	15931	11988.00	11988	0.00	5883	3024	3020	−0.13
20 × 10	21,254	17,748	17,748	0.00	11124	7321.00	7321	0.00	4368	1026	1026	0.00
20 × 10	20,981	18,568	18,568	0.00	11052	8863.00	8892	0.33	3382	1430	1433	0.21
20 × 10	22,461	19,259	19,259	0.00	12177	10329.00	10329	0.00	5842	2754	2754	0.00
20 × 10	16,124	14,442	14,414	−0.19	8754	5301.00	5301	0.00	1843	48	48	0.00
20 × 10	20,541	19,047	19,029	−0.09	13054	9889.00	9889	0.00	6034	2305	2305	0.00
20 × 10	19,189	16,091	16,091	0.00	10587	6879.00	6912	0.48	4227	836	836	0.00
20 × 10	23,474	18,279	18,196	−0.45	12043	8523.00	8294	−2.69	3865	1020	993	−2.65
20 × 10	20,531	17,942	17,898	−0.25	11118	8053.00	8053	0.00	2489	800	800	0.00
20 × 10	18,464	16,051	16051	0.00	10786	5881.00	5881	0.00	2389	522	522	0.00
20 × 20	41,814	34,821	34,821	0.00	21255	14548.00	14548	0.00	4303	989	989	0.00
20 × 20	35,460	32,761	32,635	−0.38	17368	13957.00	13863	−0.67	7336	630	630	0.00
20 × 20	41,118	34,970	34,970	0.00	21268	14647.00	14647	0.00	2010	443	443	0.00
20 × 20	37,728	31,754	31,682	−0.23	18804	12163.00	12042	−0.99	3153	233	233	0.00
20 × 20	41,816	35,635	35,635	0.00	23498	15570.00	15168	−2.58	4768	888	888	0.00
20 × 20	44,316	37,514	37,514	0.00	25637	17813.00	17813	0.00	7418	1978	1978	0.00
20 × 20	40,475	33,469	33,469	0.00	19198	13378.00	13378	0.00	3253	119	119	0.00
20 × 20	42,993	36,573	36,358	−0.59	24363	16894.00	16601	−1.73	8912	920	920	0.00
20 × 20	38,382	34,532	34,532	0.00	18885	14331.00	14443	0.78	4464	642	642	0.00
20 × 20	48,294	42,507	42,470	−0.09	30250	23169.00	23132	−0.16	11885	4659	4659	0.00
50 × 5	83,374	75,494	75,193	−0.40	71526	63390.00	62893	−0.78	59388	51340	51038	−0.59
50 × 5	73,934	64,439	64,236	−0.32	56209	51298.00	51360	0.12	47234	38737	38301	−1.13
50 × 5	71,247	62,171	61,472	−1.12	58479	49620.00	49664	0.09	47978	37877	37535	−0.90
50 × 5	73,015	64,939	64,439	−0.77	59802	52113.00	51684	−0.82	47621	39384	39271	−0.29
50 × 5	87,412	76,615	76,250	−0.48	73397	63848.00	63408	−0.69	59588	50719	50936	0.43
50 × 5	69,539	63,811	63,369	−0.69	58909	50946.00	50620	−0.64	45416	38778	37940	−2.16
50 × 5	75,334	69,548	68,893	−0.94	64742	57044.00	56668	−0.66	49577	44916	44335	−1.29
50 × 5	73,989	65,703	65,580	−0.19	59508	53317.00	52800	−0.97	48712	41362	40991	−0.90
50 × 5	71,350	63,559	63,148	−0.65	59068	51797.00	51196	−1.16	47446	40115	39972	−0.36
50 × 5	68,091	61,932	61,675	−0.41	56542	48903.00	48695	−0.43	44428	36606	36726	0.33
50 × 10	96,084	84,946	83,909	−1.22	70406	60263.00	59431	−1.38	45426	35447	34354	−3.08
50 × 10	96,122	81,099	81,142	0.05	70256	57351.00	57051	−0.52	41849	34322	33725	−1.74

Table 6 (*continued*)

Ins	$\tau = 1$				$\tau = 2$				$\tau = 3$			
	NEH	GA	DABC		NEH	GA	DABC		NEH	GA	DABC	
50 × 10	96,659	81,443	81,338	−0.13	69064	56763.00	57337	1.01	48117	34099	33296	−2.35
50 × 10	10,5275	87,585	87,857	0.31	75910	62304.00	62109	−0.31	53212	37489	37715	0.60
50 × 10	87,282	75,977	75,553	−0.56	61358	50819.00	50553	−0.52	43700	27757	27555	−0.73
50 × 10	113,153	103,735	102,622	−1.07	90967	78527.00	76774	−2.23	69936	53326	53035	−0.55
50 × 10	105,044	90,017	90,026	0.01	81741	63955.00	64933	1.53	49474	38566	38964	1.03
50 × 10	116,815	101,040	100,845	−0.19	89305	76428.00	76410	−0.02	68499	52415	51695	−1.37
50 × 10	104,654	90,444	88,425	−2.23	80038	65334.00	63883	−2.22	57251	41434	41104	−0.80
50 × 10	105,474	90,365	89,544	−0.91	76785	64003.00	64454	0.70	50412	39702	38193	−3.80
50 × 20	179,620	152,776	151,146	−1.07	128181	102017.00	100282	−1.70	75503	50301	48603	−3.38
50 × 20	198,205	178,170	178,551	0.21	147987	129488.00	127933	−1.20	96025	79423	79527	0.13
50 × 20	159,661	134,996	133,707	−0.95	115658	86084.00	86505	0.49	71120	40382	38117	−5.61
50 × 20	180,857	168,241	165,572	−1.59	138789	118388.00	116191	−1.86	91555	67990	66628	−2.00
50 × 20	161,394	137,647	138,202	0.40	113248	90583.00	88042	−2.81	75877	41834	41089	−1.78
50 × 20	205,208	174,298	174,050	−0.14	152802	124655.00	124852	0.16	105304	76922	76100	−1.07
50 × 20	196,005	169,477	167,336	−1.26	143743	121218.00	119186	−1.68	101772	68770	68095	−0.98
50 × 20	179,859	149,101	148,055	−0.70	125102	98577.00	96411	−2.20	76685	52412	50169	−4.28
50 × 20	175,772	151,486	149,608	−1.24	125688	100341.00	99285	−1.05	77412	51753	48983	−5.35
50 × 20	190,556	162,087	162,058	−0.02	139609	111332.00	110408	−0.83	93996	60348	60104	−0.40
100 × 5	298,174	275,389	275,334	−0.02	285338	250398.00	251383	0.39	262852	225179	223665	−0.67
100 × 5	267,397	235,059	234,621	−0.19	242559	210591.00	209742	−0.40	221265	185468	184197	−0.69
100 × 5	249,896	231,944	230,438	−0.65	236066	207809.00	206974	−0.40	203539	184406	181817	−1.40
100 × 5	245,398	235,459	234,802	−0.28	230042	212818.00	211118	−0.80	203266	187410	185757	−0.88
100 × 5	279,278	259,364	258,262	−0.42	252110	232388.00	232441	0.02	230034	207939	207748	−0.09
100 × 5	242,859	231,576	231,186	−0.17	222308	205016.00	206329	0.64	201969	181424	182094	0.37
100 × 5	298,044	281,771	283,174	0.50	272737	257479.00	257664	0.07	249797	231854	231638	−0.09
100 × 5	244,861	223,719	222,646	−0.48	222169	200988.00	198825	−1.08	194671	175283	173215	−1.18
100 × 5	263,468	240,307	239,975	−0.14	241094	217721.00	215198	−1.16	212581	190631	188671	−1.03
100 × 5	259,241	243,926	243,719	−0.08	237678	217854.00	217592	−0.12	209812	192785	191704	−0.56
100 × 10	377,791	342,505	338,743	−1.10	331746	289840.00	289108	−0.25	277753	237381	237848	0.20
100 × 10	347,990	297,444	294,444	−1.01	299376	249731.00	245270	−1.79	249041	199095	196946	−1.08
100 × 10	388,534	361,571	360,134	−0.40	342316	310438.00	309437	−0.32	293615	260874	259856	−0.39
100 × 10	351,393	321,643	317,798	−1.20	307801	270135.00	267251	−1.07	247390	216108	214477	−0.75
100 × 10	351,471	325,783	323,505	−0.70	321683	277181.00	274618	−0.92	256967	227478	227897	0.18
100 × 10	360,978	332,261	330,560	−0.51	313508	284224.00	283802	−0.15	264504	236711	235243	−0.62
100 × 10	398,162	372,681	372,882	0.05	353925	326250.00	325275	−0.30	306734	276819	275417	−0.51
100 × 10	411,751	367,132	365,465	−0.45	358275	316222.00	314847	−0.43	312166	265280	265410	0.05
100 × 10	414,838	379,481	376,049	−0.90	369191	328085.00	323555	−1.38	318187	272638	270339	−0.84
100 × 10	388,783	360,372	357,984	−0.66	341032	309476.00	308887	−0.19	280976	255659	253657	−0.78
100 × 20	601,079	543,100	539,114	−0.73	502158	442179.00	442523	0.08	398127	346469	344529	−0.56
100 × 20	542,467	482,928	478,498	−0.92	459732	382160.00	380613	−0.40	339899	287282	282703	−1.59
100 × 20	574,721	512,251	513,893	0.32	501413	420858.00	415127	−1.36	402611	316223	316348	0.04
100 × 20	626,271	575,760	573,722	−0.35	511250	476788.00	477808	0.21	425126	382819	379075	−0.98
100 × 20	637,247	592,204	592,374	0.03	556399	494256.00	493102	−0.23	442029	396367	394401	−0.50
100 × 20	525,558	473,867	469,846	−0.85	434001	373313.00	368815	−1.20	343435	273103	270902	−0.81
100 × 20	579,218	505,607	497,865	−1.53	476746	404445.00	402877	−0.39	385684	312395	299552	−4.11
100 × 20	554,906	485,407	476,213	−1.89	451146	382679.00	379894	−0.73	364837	279760	273432	−2.26
100 × 20	604,783	531,843	528,866	−0.56	490017	430264.00	430528	0.06	384188	329712	329727	0.00
100 × 20	545,207	487,618	483,724	−0.80	473561	386954.00	383446	−0.91	349729	285080	282060	−1.06
200 × 10	1319320	1199911	1205319	0.45	1192282	1093384.00	1101369	0.73	1092689	999767	1002725	0.30
200 × 10	1437820	1365143	1373677	0.63	1326598	1265893.00	1265999	0.01	1261013	1166643	1171867	0.45
200 × 10	1441218	1331081	1340491	0.71	1356781	1242598.00	1234266	−0.67	1247566	1128663	1141786	1.16
200 × 10	1154727	108,8191	1092638	0.41	1057139	986035.00	995151	0.92	966442	880971	903754	2.59
200 × 10	1290331	1212741	1222633	0.82	1189336	1107181.00	1113806	0.60	1090815	1018776	1017064	−0.17
200 × 10	1363418	1280086	1281765	0.13	1272808	1177972.00	1178595	0.05	1146758	1081561	1087435	0.54
200 × 10	117,8535	1088757	1093614	0.45	1046922	979124.00	993868	1.51	996140	887615	878955	−0.98
200 × 10	1328200	1277465	1284533	0.55	1237749	1191301.00	1187359	−0.33	1116333	1079061	1079673	0.06
200 × 10	1148941	1060598	1083458	2.16	1011058	947510.00	968436	2.21	956062	874785	881445	0.76
200 × 10	1347501	1278337	1289052	0.84	1285249	1183044.00	1187830	0.40	1158014	1084766	1082135	−0.24
200 × 20	1914686	1765262	1742432	−1.29	1691337	1567074.00	1570572	0.22	1547320	1376841	1353941	−1.66
200 × 20	1862329	1702159	1699957	−0.13	1624698	1491817.00	1505167	0.89	1434341	1308190	1307304	−0.07
200 × 20	1795606	1646350	1641640	−0.29	1619988	1460588.00	1457345	−0.22	1434138	1265184	1240584	−1.94
200 × 20	1737196	1562815	1556739	−0.39	1496851	1361998.00	1365570	0.26	1315248	1163455	1162550	−0.08
200 × 20	1909792	1761000	1753548	−0.42	1726674	1562980.00	1562734	−0.02	1533446	1364960	1351609	−0.98
200 × 20	1877710	1719678	1726145	0.38	1639012	1513100.00	1515802	0.18	1441838	1330849	1326076	−0.36
200 × 20	1881884	1793939	1799706	0.32	1738407	1598545.00	1608012	0.59	1518300	1410288	1408128	−0.15
200 × 20	1940654	1766076	1762022	−0.23	1730548	1567889.00	1573667	0.37	1514828	1355643	1350612	−0.37
200 × 20	1826020	1605706	1612852	0.45	1558229	1408503.00	1394085	−1.02	1344806	1208681	1207407	−0.11

(*continued on next page*)

Table 6 (*continued*)

Ins	$\tau = 1$				$\tau = 2$				$\tau = 3$			
	NEH	GA	DABC		NEH	GA	DABC		NEH	GA	DABC	
200 × 20	1809000	1629668	1634999	0.33	1613568	1427106.00	1419802	−0.51	1406831	1227440	1240100	1.03
500 × 20	8767607	8394179	8384574	−0.11	8167044	7883963.00	7846216	−0.48	7834816	7401599	7374650	−0.36
500 × 20	8903739	8488335	8430748	−0.68	8493104	7962648.00	7875107	−1.10	8103643	7598768	7421148	−2.34
500 × 20	9788215	9377660	9371255	−0.07	9184747	8910955.00	8789412	−1.36	8775336	8407828	8346013	−0.74
500 × 20	8785224	8332951	8216162	−1.40	8203164	7789377.00	7651437	−1.77	7642602	7242521	7224174	−0.25
500 × 20	9475156	9212651	9154929	−0.63	9128619	8714514.00	8656792	−0.66	8619555	8216377	8158655	−0.70
500 × 20	9043257	8610730	8469401	−1.64	8454064	8110775.00	7962821	−1.82	8066411	7589616	7486682	−1.36
500 × 20	9336734	9028213	9019965	−0.09	8839139	8527269.00	8577047	0.58	8375832	8112635	8086507	−0.32
500 × 20	9672745	9447900	9430381	−0.19	9119356	8871310.00	8854399	−0.19	8583457	8329374	8309076	−0.24
500 × 20	8581402	8177156	8114888	−0.76	8019920	7642461.00	7650108	0.10	7470198	7155465	7153987	−0.02
500 × 20	9529863	9228980	9191294	−0.41	9094226	8729464.00	8691778	−0.43	8610038	8229948	8192262	−0.46
Avg				−0.31				−0.40				−0.64

5.2. Computational results for the medium due date

In this setting, $\tau = 2$ is considered. The convergence plot of both GA and DABC algorithms is given in Fig. 7 where it is clear that DABC algorithm is much faster to converge to global or local optima. The computational results for the termination criterion of $T_{max} = 100 \times n$ seconds are given in Table 4.

From Table 4, it is obvious that the improvements over the NEH solutions are higher than the one in the GA algorithm. On overall average, the gap between the GA and DABC algorithm was 0.64% (14.79–14.15). Similar observations can also be made for Min and Max values too. In addition, a paired *t*-test confirms the significance on $\alpha = 0.05$ in differences between GA and DABC algorithm since the *p*-value was 0.001 on the behalf of DABC algorithm.

5.3. Computational results for the loose due date

In this setting, $\tau = 3$ is considered. The convergence plot of both GA and DABC algorithms is given in Fig. 8 where it is clear that DABC algorithm is much faster to converge to global or local optima. The computational results for the termination criterion of $T_{max} = 100 \times n$ seconds are given in Table 5.

From Table 5, it is obvious that the improvements over the NEH solutions are higher than the one in the GA algorithm. On overall average, the gap between the GA and DABC algorithm was 0.78% (27.25–26.47). Similar observations can also be made for Min and Max values too. In addition, a paired *t*-test confirms the significance on $\alpha = 0.05$ in differences between GA and DABC algorithm since the *p*-value was 0.000 on the behalf of DABC algorithm.

5.4. Best known solutions

Finally, for the termination criterion of $T_{max} = 100 \times n$ seconds, the best solutions for both algorithms are given in Table 6 with the relative percent deviations of DABC algorithm from GA (i.e. $(DABC - GA) * 100/GA$).

As seen in Table 6, the DABC algorithm achieved an improvement of 0.31% for $\tau = 1$, 0.40% for $\tau = 2$ and 0.64% for $\tau = 3$. These differences were statistically significant at $\alpha = 0.05$ level since all the *p*-values were 0.014, 0.030 and 0.007, respectively. These analyses confirm the fact that the DABC algorithm presented for the NIPFS problem was statistically better than the GA algorithm.

6. Conclusions

In this paper, we presented a DABC algorithm to solve the NIPFSP with the total tardiness criterion. The paper presents the following contributions: First of all, a discrete artificial bee colony algorithm is presented to solve the problem on hand first time in the literature. Secondly, some novel methods of calculating the total tardiness from makespan are introduced for the no-idle permutation flowshop scheduling problem. Finally, the main contribution of the paper is due to the fact that a novel speed-up method for the insertion neighborhood is developed for the total tardiness criterion. The performance of the discrete artificial bee colony algorithm is evaluated against a traditional genetic algorithm. The computational results show its highly competitive performance when compared to the genetic algorithm. Ultimately, we provide the best known solutions for the total tardiness criterion with different due date tightness levels for the first time in the literature for the Taillard's benchmark suit.

For the future research, the DABC algorithm will be extended to solve the other scheduling problems in the literature. In addition, the DABC algorithm may be easily extended to permutation based combinatorial optimization problems such as quadratic assignment problem, traveling salesman problem and its variants and so on.

Acknowledgement

M. Fatih Tasgetiren acknowledges the support provided by the TUBITAK (The Scientific and Technological Research Council of Turkey) under the grant # 110M622. In addition, this research is partially supported by National Science Foundation of China under Grants 61174187.

References

[1] H.S. Reza, S. Saghafian, Flowshop-scheduling problems with makespan criterion: a review, Int. J. Prod. Res. 43 (14) (2005) 2895–2929.
[2] R. Ruiz, C. Maroto, A comprehensive review and evaluation of flowshop heuristics, Eur. J. Oper. Res. 165 (2) (2005) 479–494.
[3] J.M. Framinan, R. Leisten, Total tardiness minimization in permutation flow shops: a simple approach based on a variable greedy algorithm, Int. J. Prod. Res. 46 (22) (2008) 6479–6498.
[4] S. Hasija, C. Rajendran, Scheduling in flowshops to minimize total tardiness of jobs, Int. J. Prod. Res. 42 (11) (2004) 2289–2301.
[5] R. Ruiz, E. Vallada, C. Fernandez-Martinez, Scheduling in flowshops with no-idle machines, in: U.K. Chakraborthy (Ed.), Computational Intelligence in Flowshop and Job Shop Scheduling, Springer Verlag, Berlin, Heidelberg, 2009, pp. 1–34.
[6] N.E.H. Saadani, A. Guinet, M. Moalla, Three stage no-idle flow-shops, Comput. Ind. Eng. 44 (3) (2003) 425–434.
[7] V.S. Tanaev, Y.N. Sotskov, V.A. Strusevich, Scheduling Theory, Multi-Stage Systems. Kluwer Academic Publishers, Dordrecht, 1994.
[8] P. Baptiste, L.K. Hguny, A branch and bound algorithm for the F/no−idle/Cmax. in: Proceedings of the International Conference on Industrial Engineering and Production Management, IEPM'97, Lyon, France, 1997, pp. 429–438.
[9] D. Baraz, G. Mosheiov, A note on a greedy heuristic for the flow-shop makespan minimization with no machine idle-time, Eur. J. Oper. Res. 184 (2) (2008) 810–813.
[10] I. Adiri, D. Pohoryles, Flow-shop/no-idle or no-wait scheduling to minimize the sum of completion times, Nav. Res. Logist. Q. 29 (3) (1982) 495–504.
[11] P. Vachajitpan, Job sequencing with continuous machine operation, Comput. Ind. Eng. 6 (3) (1982) 255–259.
[12] C.R. Woollam, Flowshop with no idle machine time allowed, Comput. Ind. Eng. 10 (1) (1986) 69–76.
[13] O. Cepek, M. Okada, M. Vlach, Note: on the two-machine no-idle flowshop problem, Nav. Res. Logist. 47 (4) (2000) 353–358.
[14] L. Narain, P.C. Bagga, Minimizing total elapsed time subject to zero total idle time of machines in $n \times 3$ flowshop problem, Indian J. Pure Appl. Math. 34 (2) (2003) 219–228.
[15] N.E.H. Saadani, A. Guinet, M. Moalla, A traveling salesman approach to solve the F/no − idle/Cmax problem. in: Proceedings of the International Conference on Industrial Engineering and Production Management, (IEPM01), Quebec, Canada, 2001, pp. 880–888,
[16] N.E.H. Saadani, A. Guinet, M. Moalla, A travelling salesman approach to solve the F/no−idle/Cmax problem, Eur. J. Oper. Res. 161 (1) (2005) 11–20.
[17] J. Kamburowski, More on three-machine no-idle flow shops, Comput. Ind. Eng. 46 (3) (2004) 461–466.
[18] L. Narain, P.C. Bagga, Flowshop/no-idle scheduling to minimise the mean flowtime, ANZIAM J. 47 (2005) 265–275.
[19] L. Narain, P.C. Bagga, Flowshop/no-idle scheduling to minimize total elapsed time, J. Global Optim. 33 (3) (2005) 349–367.
[20] P.J. Kalczynski, J. Kamburowski, On no-wait and no-idle flow shops with makespan criterion, Eur. J. Oper. Res. 178 (3) (2007) 677–685.
[21] M. Nawaz Jr., E.E. Enscore, I. Ham, A heuristic algorithm for the *m*-machine, *n*-job flow shop sequencing problem, OMEGA 11 (1) (1983) 91–95.
[22] L. Wang, Intelligence optimization algorithm with applications, Tsinghua University Press, Beijing, China, 2001.
[23] Q.-K. Pan, L. Wang, A novel differential evolution algorithm for no-idle permutation flowshop scheduling problems, Eur. J. Ind. Eng. 2 (3) (2008) 279–297.
[24] Q.-K. Pan, L. Wang, No-idle permutation flow shop scheduling based on a hybrid discrete particle swarm optimization algorithm, Int. J. Adv. Manuf. Technol. 39 (7–8) (2008) 796–807.
[25] E. Taillard, Some efficient heuristic methods for the flow shop sequencing problem, Eur. J. Oper. Res. 47 (1) (1990) 65–74.
[26] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, Eur. J. Oper. Res. 177 (3) (2007) 2033–2049.
[27] E. Taillard, Benchmarks for basic scheduling problems, Eur. J. Oper. Res. 64 (2) (1993) 278–285.
[28] D. Karaboga, A new design method based on artificial bee colony algorithm for digital IIR filters, J. Franklin Inst. 346 (2009) 328–348.
[29] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Appl. Soft Comput. 8 (2008) 687–697.
[30] D. Karaboga, B. Akay, A comparative study of artificial bee colony algorithm, Appl. Math. Comput. 214 (1) (2009) 108–132.
[31] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005
[32] D. Karaboga, B. Basturk, A powerful and efficient algorithm for numerical function optimization: artificial bee colony algorithm, J. Global Optim. 39 (2007) 459–471.
[33] Q.-K. Pan, M.F. Tasgetiren, P.N. Suganthan, T.J. Chua, A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem, Inf. Sci. 181 (12) (2011) 2455–2468.
[34] M.F. Tasgetiren, Q.-K. Pan, P.N. Suganthan, Angela H-L Chen, A discrete artificial bee colony algorithm for the permutation flow shop scheduling problem with total flowtime criterion, Inf. Sci. 181 (16) (2011) 3459–3475.
[35] J. Grabowski, J. Pempera, Some local search algorithms for no-wait flow-shop problem with makespan criterion, Comput. Oper. Res. 32 (8) (2005) 2197–2212.
[36] X. Dong, H. Huang, P. Chen, An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion, Comput. Oper. Res. 36 (2009) 1664–1669.
[37] Q.-K. Pan, M.F. Tasgetiren, Y.-C. Liang, A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem with makespan and total flowtime criteria, Comput. Oper. Res. 35 (9) (2008) 2807–2839.
[38] L. Davis, Applying adaptive algorithms to epistatic domains. in: Proceeding of the international joint conference on artificial intelligence, 1985, pp. 162–164.
[39] D.E. Goldberg, R. Lingle, Alleles, loci, and the TSP, in: Proceedings of the First International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Hillside, NJ, 1985, pp. 154–159.
[40] Kenneth R. Baker, J.W.M. Bertrand, An investigation of due date assignment rules with constrained tightness, J. Oper. Manage. 1 (3) (1981) 109–120.