Information Sciences 181 (2011) 3459–3475

Contents lists available at ScienceDirect

Information Sciences

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

A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops

M. Fatih Tasgetiren [a], Quan-Ke Pan [b,*], P.N. Suganthan [c], Angela H-L Chen [d]

[a] Department of Industrial Engineering, Yasar University, Bornova, Izmir, Turkey
[b] College of Computer Science, Liaocheng University, Liaocheng, 252059, PR China
[c] School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore
[d] The Department of Finance, Nanya Institute of Technology, Taoyuan 320, Taiwan, ROC

ARTICLE INFO

Article history:
Received 16 February 2010
Received in revised form 4 April 2011
Accepted 10 April 2011
Available online 21 April 2011

Keywords:
Permutation flowshop scheduling problem
Iterated greedy algorithm
Discrete differential evolution algorithm
Discrete artificial bee colony algorithm
Estimation of distribution algorithm
Genetic local search

ABSTRACT

Obtaining an optimal solution for a permutation flowshop scheduling problem with the total flowtime criterion in a reasonable computational timeframe using traditional approaches and optimization tools has been a challenge. This paper presents a discrete artificial bee colony algorithm hybridized with a variant of iterated greedy algorithms to find the permutation that gives the smallest total flowtime. Iterated greedy algorithms are comprised of local search procedures based on insertion and swap neighborhood structures. In the same context, we also consider a discrete differential evolution algorithm from our previous work. The performance of the proposed algorithms is tested on the well-known benchmark suite of Taillard. The highly effective performance of the discrete artificial bee colony and hybrid differential evolution algorithms is compared against the best performing algorithms from the existing literature in terms of both solution quality and CPU times. Ultimately, 44 out of the 90 best known solutions provided very recently by the best performing estimation of distribution and genetic local search algorithms are further improved by the proposed algorithms with short-term searches. The solutions known to be the best to date are reported for the benchmark suite of Taillard with long-term searches, as well.

1. Introduction

Flowshop scheduling is among the most prevalent problems in the field of deterministic scheduling in engineering industries [3,14,22,29,47]. The permutation flowshop represents a particular case in which solutions are represented by the permutations of n jobs, i.e., $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$. Each job comprises a set of m operations that must be performed by different machines. Each machine can process only one operation at a time. While all jobs have the same permutations on every machine, these jobs, once initiated, cannot be interrupted (preempted), and the release times of all jobs are zero. Given the processing time p_{jk} for job j on machine k, we consider the total flowtime criterion as the objective function to be minimized.

In the above specified context, $F(\pi_j)$, denoted by the flowtime of job π_j, is equivalent to the completion time $C(\pi_j, m)$ of job π_j on the last machine m because the release times of all jobs are zero. As a result, the total flowtime $TFT(\pi)$ of a permutation π can be computed by summing up flowtimes or completion times of all jobs and defined as $TFT(\pi) = \sum_{j=1}^{n} F(\pi_j) = \sum_{j=1}^{n} C(\pi_j, m)$. Then the optimal permutation $\pi^* = \{\pi_1^*, \pi_2^*, \ldots, \pi_n^*\}$ in the set of all permutations of Π is determined by

* 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), achen@nanya.edu.tw (A.H-L Chen).

0020-0255/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.ins.2011.04.018

$TFT(\pi^*) \leqslant TFT(\pi)$ for each permutation π belonging to Π. Under these specifications, the completion time for the n-job and m-machine problem is computed as follows:

$$C(\pi_1, 1) = p_{\pi_1,1} \tag{1}$$

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

$$C(\pi_1, k) = C(\pi_1, k-1) + p_{pi_1,k} \quad k = 2, \ldots, m \tag{3}$$

$$C(\pi_j, k) = \max\{C(\pi_{j-1}, k), C(\pi_j, k-1) + p_{\pi_j,k}\} \quad j = 2, \ldots, n; \quad k = 2, \ldots, m \tag{4}$$

Many different algorithms have been proposed over time in an attempt to find the exact solution to minimizing the total flowtime (TFT). Several variants of branch and bound algorithms were developed [4,5,11,41]; Ignall and Scharge [11] were the first to apply the branch and bound scheme based on two lower bounds in the two-machine flow shop problem. Then Bansal [4] extended their idea to the m-machine case. Recent publications [5,41] have detailed the development of lower bounding methods either based on Lagrangian relaxation or by introducing slack variables. These exact methods have been successfully implemented in a limited number of small instances due to lengthy execution times. For large instances in the n-job and m- machine total flowtime problem, some efficient heuristics have been developed in [2,7,8,10,23,26,31,45]. The NEH constructive method, proposed by Nawaz et al. [28], was claimed to be the best for makespan minimization in flow shops, but not effective for total flowtime minimization. Therefore, to minimize total flowtime, the heuristics in both Woo and Yim [45] and Framinan and Leisten [7] were founded on different insertion schemes from NEH. Furthermore, a composite heuristic proposed by Allahverdi and Aldowaisan [2] adopted the insertion with pair-wise exchange from Framinan and Leisten [7] to improve their solutions. So far, no heuristic is optimal for total flowtime minimization. In comparison to heuristic methods, metaheuristic methods always obtain better results, as they compose many different kinds of algorithmic components [9,12,21,27,32,42,43,46,48]. Very recently, an estimation of distribution algorithm (EDA) hybridized with variable neighborhood search (VNS) has been introduced in [13]. In addition to EDA, two genetic local search algorithms have been proposed in [24,25]. The first one employs a local search called insertion search with cut and repair, denoted by hGLS, and the second one is the genetic algorithm hybridized with a tabu search, denoted by tsGLS. These three algorithms improved almost all the best known solutions in the existing literature.

Among metaheuristics, modeling the collective behavior of self-organized systems and applying these models to solve real-world problems has been ongoing and has become a class of its own, known as swarm intelligence. Earlier works implementing the ant colony optimization (ACO) and particle swarm optimization (PSO) algorithms were conducted to simulate the swarm behavior of ant colonies and flocks of birds, respectively. Recently, some algorithms were proposed by modeling the specific intelligent behaviors of honeybee swarms [1,15–19,29,40,44]. Karaboga in [15–19] introduced an artificial bee colony (ABC) algorithm to optimize multi-variable and multi-modal continuous functions. Numerical comparisons demonstrated that the performance of the ABC algorithm is as competitive as other population-based algorithms with the advantage of employing fewer control parameters [15–19]. Furthermore, a discrete version of the ABC algorithm has been recently applied to the lot-streaming flowshop scheduling problem in [29]. Tereshko attempted to model the forage behavior of a honeybee colony based on reaction–diffusion equations [40]. Wede and Farooq proposed a routing algorithm, called BeeAdHoc, for energy efficient routing in mobile ad hoc networks [44]. Clearly, swarm intelligence can be understood as an algorithmic framework inspired by the aggregate behavior of the social insects and animals [1]. It has drawn the attention of researchers because of its advantages such as scalability, fault tolerance, adaptation, speed, modularity, autonomy, and parallelism [20].

As there is no detailed work that describes the use of the ABC algorithm to deal with the PFSP under the TFT criterion, we present a novel discrete ABC (DABC) algorithm as well as the hybrid version of our previous discrete differential evolution (hDDE) algorithm in [30] in order to solve the PFSP with the TFT criterion in this paper. The proposed algorithms are hybridized with a local search procedure, denoted by *LocalSearch*() based on swap and insertion neighborhood structures. The main purpose of the hybridization stems from the fact that DABC and DDE carry out the global search by exploration of the search space, whereas local search is responsible for intensifying the search on the local minima. Therefore, the balance in global and local searches has been effectively achieved. Through an experimental analysis, it is shown that the performance of the proposed algorithms is as competitive as the recent three best performing algorithms in terms of solution quality and CPU usage time. Ultimately, 44 out of the 90 best-known solutions recently provided by the EDA, tsGLS, and hGLS algorithms are further improved by the proposed algorithms with short-term search. For long-term search, the new best-known solutions are reported for Taillard's benchmark suite [35].

The remaining paper is organized as follows. Section 2 introduces the DABC; and Section 3 presents the hDDE algorithm. The details of the local search algorithms developed for the PFSP with TFT criterion are provided in Section 4. Section 5 discusses the computational results over benchmark problems. Finally, Section 6 comprises the concluding remarks.

2. Discrete artificial bee colony algorithm

Inspired by the intelligent foraging behaviors of honeybee swarms, Karaboga proposed the artificial bee colony (ABC) algorithm that implemented a new swarm intelligence based optimizer [15–19]. It classifies foraging artificial bees into three groups, namely, *employed bees*, *onlookers*, and *scouts*. An *employed bee* is responsible for flying to and making collections from the food source which the bee swarm is exploiting. An *onlooker* waits in the hive and decides on whether a food source is acceptable or not. This is done by watching the dances performed by the employed bees. A *scout* randomly searches for new food sources by means of some internal motivation or possible external clue. 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 where the fitness of the solution corresponds to the *nectar amount* of the associated food resource. As with other intelligent swarm-based approaches, the ABC algorithm is an iterative process. The approach begins with a population of randomly generated solutions (or food sources); then, the following steps are repeated until a termination criterion is met [15–19]:

1. Initialize the foraging process.
2. Send the employed bees to exploit the discovered food sources.
3. Using the onlooker bees, choose the food sources and determine their nectar amounts.
4. Send scouts to search for new food sources.
5. Remember the best food source found so far.
6. If a termination criterion has not been satisfied, go to step 2; otherwise stop the procedure and report the best food source found so far.

However, the above ABC algorithm, originally designed for the continuous nature of optimization problems, cannot be used for discrete/combinatorial cases; therefore, in this work, some modifications to the above ABC algorithm have been made for the discrete version, as described below.

2.1. Initialization

In the hDDE and DABC algorithms, the solution is represented by a permutation of jobs $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$. For *NP* individuals, the initial population is generated in such a way that the first solution is established by the NEH heuristic of [28], and the rest of the solutions are constructed randomly.

The NEH heuristic has two phases. In the first phase, jobs are ordered in descending sums of their processing times. In the second phase, a job permutation is established by evaluating the partial permutations based on the initial order of the first phase. Suppose a job permutation is already determined for the first k jobs; then, $k + 1$ partial permutations are constructed by inserting job $k + 1$ in the $k + 1$ possible slots of the current permutation. The partial permutation with the minimum total flowtime is kept as the current permutation for the next iteration. Then, job $k + 2$ from the first phase is considered. This is repeated until all of the jobs have been sequenced.

2.2. Employed bee phase

According to the basic ABC algorithm, the employed bees generate food sources in the neighborhood of their current positions. From prior literature, we know that two common operators, *insert* and *swap*, are used to generate neighboring solutions [33]. The insert operator removes a job from its original position j of a permutation π, and then inserts this job into another position k such that $(k \in \{j, j-1\})$, whereas the swap operator produces a neighbor by interchanging two jobs of a permutation π. By adjusting the perturbation strength p of the insert and swap operators as well as the destruction size d of the destruction and construction procedures, denoted as *DestructConstruct*( ), six neighboring strategies denoted as S_i are separately formulated by borrowing them from the iterated greedy (IG_RS) algorithm in [33] and the iterated local search (ILS) algorithm in [6]. These methods are then utilized to generate neighboring food sources for the employed bees as follows:

S_1: Performing one insert move ($p = 1$) to a permutation π.
S_2: Performing one swap move ($p = 1$) to a permutation π.
S_3: Performing two insert moves ($p = 2$) to a permutation π.
S_4: Performing two swap moves ($p = 2$) to a permutation π.
S_5: Performing a *DestructConstruct*( ) procedure with the destruction size of $d = 8$.
S_6: Performing a *DestructConstruct*( ) procedure with the destruction size of $d = 12$.

Each method used for generating neighboring food sources may have different performances 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, a new source will always be accepted if it is better than the current food source, which is similar to the basic ABC algorithm carrying out a greedy selection procedure.

The motivation for assigning one of the six strategies to each individual in the population is derived from the efficacy of the DABC algorithm, which 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_RS algorithm of [33] and the ILS algorithm of [6]. The destruction size (d) parameter needs to be carefully chosen for the IG_RS algorithm, whereas the perturbation strength (p) should be determined with care for the ILS algorithm. The perturbation can be achieved by removing a job from a position and inserting it into another position randomly, or by swapping of any two jobs randomly. In the original IG_RS algorithm of [33], detailed experiments have shown that a destruction size of 4 is suggested for the makespan criterion. However, our experiments show that taking a larger destruction size, especially $d = 8$, is more effective than smaller sizes for the total flowtime criterion. In addition, taking a larger destruction size of $d = 12$ also contributes to the diversification of the population. Regarding the perturbation strength of the ILS algorithm, perturbation values (p's) ranging from 1 to 20 were tested, and p values between 4 and 7 were suggested in [6]. However, in our experiments, p values that were within the range of 1 to 2 swap or insert moves generated better results. The number of the employed bees is set to the population size NP, and the local search procedure will be explained in detail in Section 4.

2.3. 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 roulette wheel selection in GAs [15–19]. However, the tournament selection is widely used in GA applications due to its simplicity and ability to escape from local optima. For this reason, we propose a tournament selection with a 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, allowing the better one to be chosen. In addition, an onlooker bee utilizes the same strategy that is used by the employed bee to produce a new neighboring solution. Then, a well-devised local search is employed to further improve the nectar amount of the onlooker bee. If the new food source obtained is better than the current one, the new food source will replace the current one and become a new member in the population. The onlooker bee phase in the DABC algorithm provides the intensification of a local search on the relatively promising solutions chosen with a tournament selection. The aim is to further improve the quality of solutions in the population. This is achieved by applying the assigned strategy to a food source π_k that is then improved by an effective local search. The number of onlooker bees considered is $2 \times NP$. The local search procedure will be explained in detail in Section 4.

2.4. 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 because 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 the worse of two randomly selected food sources that have been picked out from the population. Then, the scout generates a food source by performing a destruction and construction procedure with a destruction size of $d = 4$ on the best solution in the current population. This destructed and constructed solution will be replaced by the food source determined by tournament selection. Therefore, poor solutions in the population will be replaced by the best solution of the perturbations in the current population. There are $0.2 \times NP$ scout bees in this phase.

3. Discrete differential evolution algorithm

Differential evolution (DE) is one of the latest evolutionary optimization methods proposed by Storn and Price in [34]. DE is originally a continuous algorithm where individuals are represented by chromosomes based on floating-point numbers. A DE algorithm mutates individuals in such a way that the weighted difference between two randomly selected population members is added to a third member to generate a mutated solution. Then, a trial solution is generated in such a way that the mutated solution is recombined with the target solution. Thereafter, a selection operator is applied to compare the fitness function values of both competing solutions (namely, the target and trial solutions) to determine who can survive for the next generation.

Because of its continuous nature, the DE algorithm cannot tackle discrete/combinatorial optimization problems. To overcome this limitation, Pan et al. [30] and Tasgetiren et al. [39] proposed a simple and novel discrete DE (DDE) algorithm whose solutions are based on discrete job permutations. In the DDE algorithm, the target individual is represented by a permutation of jobs $\pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$. The mutant individual is obtained by perturbing each individual in the target population that is different from the one presented in [30,39]. To obtain the mutant individual, the following equation can be used:

$$v_i^t = \begin{cases} insert(\pi_i^{t-1}) & if\ r < P_m \\ swap(\pi_i^{t-1}) & otherwise \end{cases} \tag{5}$$

where π_i^{t-1} is the individual in the target population, P_m is the perturbation probability, *insert* is the simple random insertion move, and *swap* is the simple random interchange of two randomly chosen jobs. A uniform random number r between 0 and 1 is generated. If r is less than P_m, then the perturbation operator is applied to generate the mutant individual as $v_i^t = insert(\pi_i^{t-1})$; otherwise, the individual is perturbed by the swap operator as $v_i^t = swap(\pi_i^{t-1})$. Following the perturbation phase, the trial individual is obtained such that

$$u_i^t = \begin{cases} CR(v_i^t, \pi_i^{t-1}) & \text{if } r < P_c \\ v_i^t & \text{otherwise} \end{cases} \tag{6}$$

where CR is any type of crossover operator and P_c is the crossover probability. In other words, if a uniform random number r is less than the crossover probability P_c, then the crossover operator is applied to generate the trial individual $u_i^t = CR(v_i^t, \pi_i^{t-1})$. Otherwise, the trial individual is chosen as $u_i^t = v_i^t$. By doing so, the trial individual is made from the outcome of the perturbation operator or from the crossover operator. Finally, the selection is based on "survival of the fitter" among the trial and target individuals such that

$$\pi_i^t = \begin{cases} u_i^t & \text{if } f(u_i^t) < f(\pi_i^{t-1}) \\ \pi_i^{t-1} & \text{otherwise} \end{cases} \tag{7}$$

4. Hybridization with local search

In order to intensify the search on the local minima and improve the solution quality, both DABC and hDDE are hybridized with some local search methods. For this reason, a well-devised local search algorithm denoted as *LocalSearch*() is fused into the DABC and hDDE algorithms. The proposed *LocalSearch*() procedure is based on a systematic application of both *insert* and *swap* moves. Because we employ six different strategies in both the employed bee and onlooker bee phases, the DABC algorithm takes advantage of both the IG_RS and ILS algorithms. In other words, some individuals work like an ILS with different perturbation strengths while others imitate an IG_RS with different destruction sizes. For the PFSP with the TFT criterion, we employed the following very effective local search in Fig. 1 embedded in the DABC and hDDE algorithms:

The key procedure of the IG_RS algorithm [33] is the destruction and construction procedure, which is given in Fig. 2. In the destruction step, a given number d of jobs, randomly chosen and without repetition, are removed from the solution,

```
procedure LS(π)
   flag = false
   π0 = π
   do
      π1 = InsertLS(π0)
      π2 = SwapLS(π1)
      if (f(π2) < f(π0))
         flag = true
         π0 = π2
      else
         flag = false
   while(flag == true)
   π = π0
   return π
endprocedure
```

Fig. 1. Local search algorithm.

```
procedure DestructConstruct(π, d)
   π^D = π^R = Destruct_d(π)
   π = Construct(π^D, π^R)
   return π
endprocedure
```

Fig. 2. Destruction and construction procedure.

$$
\begin{aligned}
&\mathit{procedure}\quad \mathit{SwapLS}(\pi)\\
&\pi_0 = \pi\\
&i = 1\\
&\mathit{do}\\
&\quad j = i + 1\\
&\quad \mathit{do}\\
&\quad\quad \pi_1 = \pi_0\\
&\quad\quad \mathit{swap}(\pi_1, i, j)\\
&\quad\quad \mathit{if}\quad (f(\pi_1) < f(\pi_0))\\
&\quad\quad\quad \pi_0 = \pi_1\\
&\quad\quad\quad i = 1\\
&\quad\quad\quad j = i + 1\\
&\quad\quad \mathit{else}\\
&\quad\quad\quad j = j + 1\\
&\quad\quad \mathit{endif}\\
&\quad \mathit{while}(j < n)\\
&\quad i = i + 1\\
&\mathit{while}(i < n)\\
&\pi = \pi_0\\
&\mathit{endprocedure}
\end{aligned}
$$

Fig. 3. Swap local search procedure.

$$
\begin{aligned}
&\mathit{procedure}\quad \mathit{InsertLS}(\pi)\\
&\pi_0 = \pi\\
&i = 1\\
&\mathit{do}\\
&\quad j = i + 1\\
&\quad \mathit{do}\\
&\quad\quad \pi_1 = \pi_0\\
&\quad\quad \mathit{insert}(\pi_1, i, j)\\
&\quad\quad \mathit{if}\quad (f(\pi_1) < f(\pi_0))\\
&\quad\quad\quad \pi_0 = \pi_1\\
&\quad\quad\quad i = 1\\
&\quad\quad\quad j = i + 1\\
&\quad\quad \mathit{else}\\
&\quad\quad\quad j = j + 1\\
&\quad\quad \mathit{endif}\\
&\quad \mathit{while}(j < n)\\
&\quad i = i + 1\\
&\mathit{while}(i < n)\\
&\pi = \pi_0\\
&\mathit{return}\quad \pi\\
&\mathit{endprocedure}
\end{aligned}
$$

Fig. 4. Insert local search procedure.

$procedure \quad InsertFPR(\pi)$

$\pi_0 = \pi$

$k = 1$

$t = 0;$

$while(t < n)$

$\quad k = (k+1)\%n$

$\quad \pi_1 = remove \quad job \quad \pi_k \quad from \quad \pi_0$

$\quad \pi_2 = \ best \quad permutation \quad obtained \quad by \quad inserting \quad \pi_k \quad in \quad all \quad possible \quad positions \quad of \quad \pi_1$

$\quad if \quad (f(\pi_2) < f(\pi_0))$

$\qquad \pi_0 = \pi_2$

$\qquad t = 0$

$\quad else$

$\qquad t = t + 1$

$\quad endif$

$endwhile$

$\pi = \pi_0$

$return \quad \pi$

$endprocedure$

Fig. 5. Insert local search procedure with first pivoting rule.

therefore resulting in two partial solutions. The first one d jobs is denoted as π^R and includes the removed jobs in the order in which they are removed. The second one $n - d$ jobs is the original solution without the removed jobs, which is denoted byπ^D. Then, in the construction phase, a constructive heuristic procedure is needed. For this purpose, we employ the NEH insertion heuristic from [28]. In order to reinsert jobs from π^R into the destructed solution π^D, the first job, π_1^R, 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_1^R), the best partial solution with the minimum total flowtime is chosen and kept for the next iteration. Then, the second job, π_2^R, is considered. This is repeated until π^R is empty or a final solution is obtained. Hence, π^D is again of size n. For details regarding the *DestructConstruct*() procedure, we refer to Ruiz and Stutzle [33], where the procedure is well-illustrated with an example for the makespan criterion. It should be noted that the *DestructConstruct*() procedure is used by both algorithms in different ways. In the hDDE algorithm, the *DestructConstruct*() procedure is first applied to the best solution (π_B) in the population with a destruction size of $d = 8$. Then, the *LocalSearch*() procedure, the IG_RS algorithm with a larger destruction size, is used. On the other hand, in the DABC algorithm, the *DestructConstruct*() procedure is embedded in the strategy set. After applying the assigned strategy to each solution in the population, the *LocalSearch*() procedure is applied with a small probability $p_{LS} = 0.01$ in the employed bee phase and applied to a solution π_k determined by the tournament selection in the onlooker bee phase.

Similar to those in Jarboui et al. [13], two neighborhood structures, namely *InsertLS*() and *SwapLS*(), are considered as our local search procedures. *InsertLS*() evaluates all possible insert moves of pairs of job position (i,j) as shown in Fig. 3, whereas *SwapLS*() considers all possible interchange of pairs of job positions (i,j) as shown in Fig. 4. Note that in local search procedures, *swap* and *insert* moves are systematically carried out in such a way that if any improvement is made, the search starts from scratch on the *improved solution* again.

Furthermore, we also employed an insertion procedure denoted as *InsertFPR*(). In the *InsertFPR*() procedure, a job is removed from a permutation and inserted into $n - 1$ positions; then, the permutation with the best out of the $n - 1$ insertions is retained for the next iteration. The same procedure is repeated in a greedy manner for the n number of jobs. The pseudocode of the *InsertFPR*() procedure is given in Fig. 5.

To further clarify, the *DestructConstruct*() and *LocalSearch*() procedures are sequentially applied to the best solution in the target population at each generation in the hDDE algorithm. On the other hand, the *SwapLS*() and *InsertFPR*() procedures are sequentially applied to each trial individual generated by the hDDE algorithm with a small probability of $p_{LS} = 0.01$. Finally, the pseudocode of the hDDE algorithm is given in Fig. 6.

As for the DABC algorithm, the following computational procedure is used, as also depicted in Fig. 7.

1. Set the parameters, NP, $S_{\max}$, p_{LS}, and S_i for each food source.
2. Initialize the population:

a. The first individual is generated by NEH whereas others are randomly established, i.e., $\pi_1 = NEH(\pi_1)$ and $\pi = \{\pi_2, \pi_2, \ldots, \pi_{NP}\}$ and evaluate each solution in the population.
3. Employed bee phase:
   a. For $i = 1, 2, \ldots, NP$, repeat the following sub-steps:
      i. Produce a new food source u_i for the ith employed bee that is associated with the strategy S_i and evaluate the new solution.
      ii. If $r < p_{LS}$, perform the *InsertFPR*( ) and *SwapLS*( ) procedures on u_i, sequentially.
      iii. If u_i is better than π_i, let $\pi_i = u_i$ and update π_B, the best solution so far.

4. Onlooker bee phase:
   a. For $i = 1, 2, \ldots, 2 \times NP$, repeat the following sub-steps:
      i. Select a food source π_k in the population for the onlooker bee by using the tournament selection (better TFT is chosen).

$$
\begin{array}{l}
\textit{procedure} \quad \textit{hDDE} \\
\pi_1 = NEH(\pi_1) \\
\pi = [\pi_2, \pi_3, .., \pi_{NP}] \\
\pi_B = \underset{i=1,2,..,NP}{\arg\min}(\pi_i) \\
\textit{do} \\
\quad v_i = \begin{cases} insert(\pi_i) & if \quad r < P_m \\ swap(\pi_i) & else \end{cases}_{i=1,2,..,NP} \\
\quad u_i = CR(v_i, \pi_i)_{i=1,2,..,NP} \\
\quad \textit{if} \quad (r < P_{LS}) \\
\quad\quad u_i = InsertFPR(u_i)_{i=1,2,..,NP} \\
\quad\quad u_i = SwapLS(u_i)_{i=1,2,..,NP} \\
\quad\quad \textit{if} \quad \left(f(u_i) < f(\pi_i)_{i=1,2,..,NP} \right) \\
\quad\quad\quad \pi_i = i_{i=1,2,..,NP} \\
\quad\quad\quad \textit{if} \quad (f(u_i) < f(\pi_B))_{i=1,2,..,N} \\
\quad\quad\quad\quad \pi_B = u_i \\
\quad\quad\quad \textit{endif} \\
\quad\quad \textit{endif} \\
\quad \textit{elseif} \\
\quad\quad \left(f(u_i) < f(\pi_i)_{i=1,2,..,NP} \right) \\
\quad\quad \pi_i = u_i{}_{i=1,2,..,NP} \\
\quad \textit{endif} \\
\quad \pi_B = DestructConstruct(\pi_B, d) \\
\quad \pi_B = LocalSearch(\pi_B) \\
\quad \textit{while}(NotTer\min ation) \\
\quad \textit{return} \quad \pi_B \\
\quad \textit{endprocedure}
\end{array}
$$

Fig. 6. hDDE Algorithm.

$procedure \quad DABC$
$\pi_1 = NEH(\pi_1)$
$\pi = [\pi_2, \pi_3, .., \pi_{NP}]$
$S_i = rand()\%S_{\max}$
$\pi_B = \arg\min_{i=1,2,..,NP}(\pi_i)$
do
$// \quad Employed \quad Bee \quad Phase$
$\quad u_i = \pi_i \quad {}_{i=1,2,..,NP}$
$\quad u_i = S_i(u_i) \quad {}_{i=1,2,..,NP}$
$\quad if \quad (r < p_{LS})$
$\quad\quad u_i = InsertFPR(u_i) \quad {}_{i=1,2,..,NP}$
$\quad\quad u_i = SwapLS(u_i) \quad {}_{i=1,2,..,NP}$
$\quad\quad if \quad (f(u_i) < f(\pi_i)) \quad {}_{i=1,2,..,NP}$
$\quad\quad\quad \pi_i = u_i \quad {}_{i=1,2,..,NP}$
$\quad\quad\quad if \quad (f(u_i) < f(\pi_B)) \quad {}_{i=1,2,..,NP}$
$\quad\quad\quad\quad \pi_B = u_i \quad {}_{i=1,2,..,NP}$
$\quad\quad\quad endif$
$\quad\quad endif$
$\quad endif$
$// \quad Onlooker \quad Bee \quad Phase$
$\quad \pi_k = TournamentSelect(\pi_k \in NP) \quad {}_{k=1,2,..,2*NP}$
$\quad u_k = S_k(\pi_k)$
$\quad u_k = LocalSearch(u_k)$
$\quad if \quad (f(u_k) < f(\pi_k)) \quad {}_{k=1,2,..,NP}$
$\quad\quad \pi_k = u_k \quad {}_{k=1,2,..,2*NP}$
$\quad\quad if \quad (f(u_k) < f(\pi_B)) \quad {}_{k=1,2,..,2*NP}$
$\quad\quad\quad \pi_B = u_k \quad {}_{k=1,2,..,2*NP}$
$\quad\quad else$
$\quad\quad\quad S_i = rand()\%S_{\max}$
$\quad endif$
$// \quad Scout \quad Bee \quad Phase$
$\quad \pi_k = TournamentSelect(\pi_k \in NP) \quad {}_{k=1,2,..,0.2*NP}$
$\quad u_k = DestructionConstruction(\pi_B, d)$
$\quad \pi_k = u_k$
$while(NotTermination)$
$return \quad \pi_B$
$endprocedure$

Fig. 7. DABC Algorithm.

Table 1
Comparison of DABC and hDDE to IG_RS: $T_{\max} = 0.4 \times n \times m$ s.

$n \times m$	IG_RS				hDDE				DABC			
	Min	*Max*	*Avg*	*Std*	*Min*	*Max*	*Avg*	*Std*	*Min*	*Max*	*Avg*	*Std*
20 × 5	14033	14033	14033.0	0.0	14033	14033	14033	0.0	14033	14033	14033	0.0
	15151	15151	15151.0	0.0	15151	15151	15151	0.0	15151	15151	15151	0.0
	13301	13301	13301.0	0.0	13301	13301	13301	0.0	13301	13301	13301	0.0
	15447	15447	15447.0	0.0	15447	15447	15447	0.0	15447	15447	15447	0.0
	13529	13529	13529.0	0.0	13529	13529	13529	0.0	13529	13529	13529	0.0
	13123	13123	13123.0	0.0	13123	13123	13123	0.0	13123	13123	13123	0.0
	13557	13557	13557.0	0.0	13557	13557	13557	0.0	13557	13557	13557	0.0
	13948	13957	13948.9	2.9	13948	13948	13948	0.0	13948	13948	13948	0.0
	14295	14295	14295.0	0.0	14295	14295	14295	0.0	14295	14295	14295	0.0
	12943	12976	12946.3	10.4	12943	12943	12943	0.0	12943	12943	12943	0.0
20 × 10	20911	20911	20911.0	0.0	20911	20911	20911	0.0	20911	20911	20911	0.0
	22440	22440	22440.0	0.0	22440	22440	22440	0.0	22440	22440	22440	0.0
	19833	19833	19833.0	0.0	19833	19833	19833	0.0	19833	19833	19833	0.0
	18710	18747	18719.5	15.2	18710	18710	18710	0.0	18710	18710	18710	0.0
	18641	18641	18641.0	0.0	18641	18641	18641	0.0	18641	18641	18641	0.0
	19245	19249	19246.6	2.1	19245	19245	19245	0.0	19245	19245	19245	0.0
	18363	18363	18363.0	0.0	18363	18363	18363	0.0	18363	18363	18363	0.0
	20241	20241	20241.0	0.0	20241	20241	20241	0.0	20241	20241	20241	0.0
	20330	20330	20330.0	0.0	20330	20330	20330	0.0	20330	20330	20330	0.0
	21320	21320	21320.0	0.0	21320	21320	21320	0.0	21320	21320	21320	0.0
20 × 20	33623	33623	33623.0	0.0	33623	33623	33623	0.0	33623	33623	33623	0.0
	31587	31587	31587.0	0.0	31587	31587	31587	0.0	31587	31587	31587	0.0
	33920	33920	33920.0	0.0	33920	33920	33920	0.0	33920	33920	33920	0.0
	31661	31661	31661.0	0.0	31661	31661	31661	0.0	31661	31661	31661	0.0
	34557	34586	34559.9	9.2	34557	34557	34557	0.0	34557	34557	34557	0.0
	32564	32564	32564.0	0.0	32564	32564	32564	0.0	32564	32564	32564	0.0
	32922	32922	32922.0	0.0	32922	32922	32922	0.0	32922	32922	32922	0.0
	32412	32467	32423.9	20.2	32412	32412	32412	0.0	32412	32412	32412	0.0
	33600	33600	33600.0	0.0	33600	33600	33600	0.0	33600	33600	33600	0.0
	32262	32262	32262.0	0.0	32262	32262	32262	0.0	32262	32262	32262	0.0
50 × 5	64848	65138	64967.2	93.4	64803	65044	64926.8	76.5	64803	64939	64881.7	44.1
	68097	68415	68266.7	121.3	68051	68313	68165.5	92.6	68086	68266	68162.6	63.3
	63391	63877	63629.1	126.8	63226	63710	63396.8	139.7	63162	63529	63384.6	111.5
	68388	68996	68665.5	197.7	68345	68554	68477.1	77.0	68242	68609	68460.4	104.7
	69486	69783	69625.2	106.4	69360	69684	69538.3	94.6	69448	69621	69513.5	50.5
	67013	67247	67121.2	61.7	66841	67114	67010.2	79.3	66878	67137	67019.1	89.7
	66318	66670	66455.7	109.4	66271	66484	66351.9	67.5	66271	66370	66289.1	31.6
	64571	64917	64718.4	101.5	64365	64727	64562.6	119.3	64381	64599	64488.9	85.7
	63101	63414	63249.4	109.7	63015	63279	63123.5	75.8	63081	63178	63134.1	26.5
	69043	69723	69267.4	188.5	68906	69302	69121.6	105.0	68989	69189	69079.6	53.4
50 × 10	87456	88360	87938.6	251.4	87143	87761	87494.6	172.2	87340	87808	87573.9	146.5
	82992	83928	83593.6	308.0	82949	83711	83291.6	239.3	83068	83519	83261.1	130.7
	80134	80951	80438.6	257.3	80105	80545	80299	174.2	80139	80389	80248	87.4
	86992	87648	87236.0	231.5	86547	87092	86840.1	160.0	86525	86994	86827.4	147.0
	86784	87365	87057.7	194.4	86511	87098	86720.5	196.6	86453	86988	86656	171.3
	86815	87365	87042.8	200.8	86730	87231	86997.3	164.8	86687	87012	86843.7	99.1
	89431	89965	89681.9	159.5	89024	89515	89280.2	160.2	88996	89393	89194	143.5
	87135	87822	87412.3	219.1	86886	87480	87242.6	195.5	86883	87284	87095.2	145.4
	85810	86575	86242.7	222.7	85646	86268	85927.7	174.8	85637	86128	85871.4	146.8
	88525	89039	88723.5	139.2	88139	88746	88425.7	202.3	87998	88583	88275.9	169.0
50 × 20	125835	126789	126488.7	287.1	125877	126633	126269.1	245.7	125842	126393	126109.6	169.7
	119842	120680	120238.2	315.1	119270	120111	119579.2	265.8	119270	119686	119473.1	157.4
	116966	117894	117422.9	298.3	116628	117514	116942	238.1	116712	117235	116933.1	209.4
	121097	122263	121705.4	364.8	120983	121550	121208.3	170.4	120897	121452	121109	203.0
	118872	119206	119059.0	103.9	118767	119288	118948	156.3	118457	119059	118805.1	197.6
	120997	121779	121294.3	251.6	120703	121669	121136.3	266.4	120850	121205	120988.2	136.2
	123275	124627	124096.6	447.6	123084	123624	123391	145.1	123043	124073	123412.8	296.1
	122872	123400	123178.3	157.2	122672	123253	123023.8	203.6	122529	123326	122925.8	208.9
	122274	123529	122871.1	419.6	122018	122653	122387	217.0	121872	122769	122238.3	231.8
	124513	125114	124808.7	211.6	124327	124880	124616.9	177.9	124079	124974	124443.1	265.7
100 × 5	254975	256749	256014.5	673.5	254319	255715	255004.3	395.4	254738	255302	255123.1	194.2
	244512	246276	245114.0	576.5	243410	244541	243952.5	346.7	243834	245101	244456.1	317.9
	239849	240904	240310.0	332.3	238772	239867	239248.4	283.4	239242	239860	239612.3	214.0
	228577	229677	229209.4	332.6	228518	229331	228727.1	244.2	228925	229555	229216.2	168.0

Table 1 (*continued*)

$n \times m$	IG_RS				hDDE				DABC			
	Min	*Max*	*Avg*	*Std*	*Min*	*Max*	*Avg*	*Std*	*Min*	*Max*	*Avg*	*Std*
	241810	242780	242306.2	293.0	241243	241923	241544.6	182.8	241959	242339	242195.1	143.9
	234183	236032	235015.1	566.8	233696	234778	233946.8	309.1	234017	234877	234410.3	240.8
	242148	244253	242827.0	564.1	241013	242251	241598.5	377.0	241727	242555	242068.4	237.3
	232346	233536	233050.2	460.6	231716	232606	232259.2	306.7	232238	233111	232774.8	286.5
	249831	251285	250603.6	471.8	249180	250247	249651.5	322.1	249884	250485	250134.1	191.9
	244513	245620	244883.4	383.0	243838	244866	244388.9	314.7	244335	244904	244650.1	200.6
100 × 10	300637	303664	302318.7	938.5	300201	302144	301247	642.7	301204	302543	301783.5	436.8
	277698	280971	278976.6	1069.4	275920	277892	277086.5	725.0	276470	277695	277086.9	419.4
	291776	293735	292731.7	649.9	289366	291726	290512.8	805.9	289400	291464	290702.8	591.8
	304457	307066	305652.3	709.1	303403	305061	304059.5	584.5	303062	305339	304191	660.9
	287315	288841	287785.3	578.1	285950	288341	287019.8	823.9	286742	288257	287431	520.2
	272009	275952	274241.7	1078.8	271601	273350	272621.3	598.1	272282	273436	272821.4	402.1
	281711	285961	283909.0	1133.8	280921	283425	282575.1	815.6	281716	283470	282794	561.7
	294307	295897	295231.5	605.7	292664	294736	293727.3	627.0	293071	295003	294040.2	583.0
	304256	306196	305466.9	611.8	303742	306398	305021.4	804.3	304457	305602	304927.3	399.1
	293331	296635	295163.6	913.0	293138	295187	293942	691.4	293775	295220	294526.1	500.8
100 × 20	369642	374331	371736.4	1613.7	368702	370504	369715	629.0	369297	371105	370036.5	634.0
	375281	380144	377357.3	1315.6	374894	377425	376050.8	815.3	374321	376337	375613	701.7
	372929	376018	374349.4	1073.7	372057	376458	373549.3	1202.7	373210	375493	373825.5	682.5
	376937	379614	378157.1	955.1	375540	377092	376465.3	565.7	374205	377603	376101	992.3
	371844	374114	372902.0	698.4	370646	374128	372685.9	1028.4	371334	372837	372196.9	606.3
	374506	377640	376237.7	989.2	373826	376861	375473.3	920.7	373689	376553	375352.8	929.4
	375869	379958	378045.7	1019.0	376807	377755	377286.8	304.4	375188	378509	377238.7	920.9
	387667	392956	389794.7	1496.2	386803	389312	388238	885.5	387582	389094	388201.5	583.0
	378390	380964	379905.5	916.2	377730	379836	378537.1	675.3	377113	378835	378191.8	613.7
	382453	385311	384071.7	856.3	380773	383618	382454.5	930.3	380725	384133	382831.5	987.4

Table 2
Two-sided paired *t*-test.

Algorithms compared	*Min*	*Max*	*Avg*	*Std*
hDDE vs IG_RS	$p = 0.000$	$p = 0.000$	$p = 0.000$	$p = 0.000$
DABC vs IG_RS	$p = 0.000$	$p = 0.000$	$p = 0.000$	$p = 0.000$
DABC vs hDDE	$p = 0.024$	$p = 0.215$	$p = 0.059$	$p = 0.004$

ii. Generate a new solution u_k for the onlooker bee by using the strategy S_k and apply the *LocalSearch*( ) procedure. If u_k is better than π_k, let $\pi_k = u_k$ and update π_B, the best solution found so far.Iitemiii. If the generated solution u_k is not better than the selected π_k, randomly switch to another strategy by re-obtaining $S_k = rand()\%S_{\max}$.

5. Scout phase:
   a. A tournament selection with a size of 2 is again used to discard the worse of the two randomly selected food sources from the population. Then, the scout generates a food source by performing a *DestructConstruct*( ) procedure with a destruction size of $d = 4$ to π_B, the best solution in the current population. The obtained solution is replaced with the food source determined by the tournament selection.
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 3.

5. Computational results

The DABC and hDDE 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 90 benchmark instances of Taillard in [35] ranging from 20 jobs with 5 machines to 100 jobs with 20 machines. All of the parameters in this study were determined experimentally. Regarding the parameters of the hDDE algorithm, a small population size of $NP = 10$ is employed. The destruction and construction procedure with a destruction size of $d = 8$ was used in the hDDE algorithm. The crossover and mutation probabilities were taken as 0.9 and 0.5, respectively. The probability that a local search applied to each trial individual was taken as $p_{LS} = 0.01$, and the PTL crossover operator in [30,36–39] was employed. To be fair, especially with Jarboui et al. [13], the same termination criterion is used as $T_{\max} = 0.4 \times n \times m$ s for the short-term search, whereas it is fixed at $T_{\max} = 3 \times n \times m$ s for the long-term search. Note that a similar machine speed is also used so that the comparisons will be fair enough especially with the EDA algorithm. As for the

Table 3
Comparison to the best performing algorithms: $T_{\max} = 0.4 \times n \times m$ s.

$n \times m$	EDA	tsGLS		*hGLS*		hDDE		DABC	
	Min	*Min*	*Avg*	*Min*	*Avg*	*Min*	*Avg*	*Min*	*Avg*
20 × 5	**14033**	**14033**	14051	**14033**	14037.8	**14033**	**14033**	**14033**	**14033**
	15151	**15151**	15216	**15151**	15180.4	**15151**	**15151**	**15151**	**15151**
	13301	**13301**	13355	**13301**	13328.7	**13301**	**13301**	**13301**	**13301**
	15447	**15447**	15476	**15447**	15475.3	**15447**	**15447**	**15447**	**15447**
	13529	**13529**	13534	**13529**	**13529**	**13529**	**13529**	**13529**	**13529**
	13123	**13123**	13135	**13123**	**13123**	**13123**	**13123**	**13123**	**13123**
	13548	**13548**	13581	**13548**	13586.3	**13548**	**13548**	**13548**	**13548**
	13948	**13948**	13954	**13948**	13964.7	**13948**	**13948**	**13948**	**13948**
	14295	**14295**	14351	**14295**	14319.6	**14295**	**14295**	**14295**	**14295**
	12943	**12943**	12969	**12943**	12969.4	**12943**	**12943**	**12943**	**12943**
20 × 10	**20911**	**20911**	20945	**20911**	20925.1	**20911**	**20911**	**20911**	**20911**
	22440	**22440**	22540	**22440**	22459.8	**22440**	**22440**	**22440**	**22440**
	19833	**19833**	19854	**19833**	19845.7	**19833**	**19833**	**19833**	**19833**
	18710	**18710**	18792	**18710**	18751.4	**18710**	**18710**	**18710**	**18710**
	18641	**18641**	18705	**18641**	18661.9	**18641**	**18641**	**18641**	**18641**
	19245	**19245**	19336	**19245**	19294.6	**19245**	**19245**	**19245**	**19245**
	18363	**18363**	18403	**18363**	18364.3	**18363**	**18363**	**18363**	**18363**
	20241	**20241**	20294	**20241**	20255.9	**20241**	**20241**	**20241**	**20241**
	20330	**20330**	20352	**20330**	**20330**	**20330**	**20330**	**20330**	**20330**
	21320	**21320**	21362	**21320**	21329.5	**21320**	**21320**	**21320**	**21320**
20 × 20	**33623**	**33623**	33801	**33623**	**33719**	**33623**	**33623**	**33623**	**33623**
	31587	**31587**	31601	**31587**	31590.7	**31587**	**31587**	**31587**	**31587**
	33920	**33920**	34014	**33920**	**33925**	**33920**	**33920**	**33920**	**33920**
	31661	**31661**	31727	**31661**	31691.1	**31661**	**31661**	**31661**	**31661**
	34557	**34557**	34615	**34557**	34587.3	**34557**	**34557**	**34557**	**34557**
	32564	**32564**	32590	**32564**	32570.1	**32564**	**32564**	**32564**	**32564**
	32922	**32922**	33042	**32922**	32989.9	**32922**	**32922**	**32922**	**32922**
	32412	**32412**	32480	**32412**	32429.4	**32412**	**32412**	**32412**	**32412**
	33600	**33600**	33623	**33600**	33611.9	**33600**	**33600**	**33600**	**33600**
	32262	**32262**	32309	**32262**	32271.6	**32262**	**32262**	**32262**	**32262**
50 × 5	64817	64892	65076	64853	64924.6	**64803**	64926.8	**64803**	64881.7
	68066	68132	68415	68173	68263.2	**68051**	68165.5	68086	68162.6
	63240	63425	63863	63367	63524.0	63226	63396.8	**63162**	63384.6
	68287	68478	68796	68281	68522.1	68345	68477.1	**68242**	68460.4
	69478	69628	69758	69551	69670.8	**69360**	69538.3	69448	69513.5
	66882	67109	67431	67013	67120.0	**66841**	67010.2	66878	67019.1
	66274	66334	66664	66294	66405.5	**66271**	66351.9	**66271**	66289.1
	64418	64459	64806	64560	64635.8	**64365**	64562.6	64381	64488.9
	62981	63154	63288	63029	63190.1	63015	63123.5	63081	63134.1
	68843	69184	69356	69037	69187.0	68906	69121.6	68989	69079.6
50 × 10	87238	87944	88575	87599	87783.5	**87143**	87494.6	87340	87573.9
	83116	83542	84040	83001	83312.0	**82949**	83291.6	83068	83261.1
	80132	80558	80893	80224	80453.4	**80105**	80299	80139	80248.0
	86725	86956	87511	86787	87050.8	86547	86840.1	**86525**	86827.4
	86626	87002	87497	86646	86910.4	86511	86720.5	**86453**	86656.0
	86735	87058	87631	86826	87003.0	86730	86997.3	**86687**	86843.7
	89014	89196	89980	88996	89371.1	89024	89280.2	**88996**	89194.0
	87025	87358	87936	**86860**	87345.6	86886	87242.6	86883	87095.2
	85688	85975	86580	85841	86104.3	85646	85927.7	**85637**	85871.4
	88149	88574	89450	88293	88578.9	88139	88425.7	**87998**	88275.9
50 × 20	**125831**	127011	127700	126073	126448.3	125877	126269.1	125842	126109.6
	119247	120104	120856	119300	119737.0	119270	119579.2	119270	119473.1
	116696	117522	117928	116856	117194.8	**116628**	116942	116712	116933.1
	120834	120983	122009	121028	121404.4	120983	121208.3	120897	121109.0
	118457	119319	119852	118736	118943.0	118767	118948	**118457**	118805.1
	120820	121393	122269	121066	121516.3	**120703**	121136.3	120850	120988.2
	123271	124418	125128	123580	123879.3	123084	123391	**123043**	123412.8
	122820	123937	124576	122770	123175.7	122672	123023.8	**122529**	122925.8
	121872	122727	123424	121872	122364.3	122018	122387	**121872**	122238.3
	124486	125126	125511	124354	124849.6	124327	124616.9	**124079**	124443.1
100 × 5	**254250**	255088	256555	254619	255198.4	254319	255004.3	254738	255123.1
	243227	244174	245174	243817	244825.0	243410	243952.5	243834	244456.1
	238580	239641	240372	239075	239697.2	238772	239248.4	239242	239612.3
	228520	228941	229706	**228291**	228787.0	228518	228727.1	228925	229216.2

Table 3 (*continued*)

$n \times m$	EDA	tsGLS		hGLS		hDDE		DABC	
	Min	*Min*	*Avg*	*Min*	*Avg*	*Min*	*Avg*	*Min*	*Avg*
	241397	241726	242601	241255	241742.3	**241243**	241544.6	241959	242195.1
	233161	234038	234888	233583	234260.1	233696	233946.8	234017	234410.3
	241213	241611	242557	241458	241930.4	**241013**	241598.5	241727	242068.4
	231865	232726	233487	232283	232813.0	**231716**	232259.2	232238	232774.8
	249038	250075	251123	249269	250235.7	249180	249651.5	249884	250134.1
	243647	244888	246120	243879	244647.4	243838	244388.9	244335	244650.1
100 × 10	301001	301648	303414	300634	302291.8	**300201**	301247	301204	301783.5
	275601	278502	279644	277209	278049.9	275920	277086.5	276470	277086.9
	288943	292004	293269	290198	291703.2	289366	290512.8	289400	290702.8
	303443	304215	307526	303669	305501.3	303403	304059.5	**303062**	304191.0
	286646	288118	289376	287136	288222.6	**285950**	287019.8	286742	287431.0
	271956	272957	275255	273172	273951.3	**271601**	272621.3	272282	272821.4
	281090	282685	284325	281306	283139.0	**280921**	282575.1	281716	282794.0
	293067	295122	296781	293628	294827.2	**292664**	293727.3	293071	294040.2
	303893	305205	307181	304276	305295.3	**303742**	305021.4	304457	304927.3
	293492	295312	297675	293465	295056.2	**293138**	293942	293775	294526.1
100 × 20	**368641**	372405	374169	370603	371741.8	368702	369715	369297	370036.5
	374838	380290	382053	375982	377991.6	374894	376050.8	**374321**	375613.0
	372423	376796	378348	373554	375336.2	**372057**	373549.3	373210	373825.5
	374832	380049	381695	376236	378454.3	375540	376465.3	**374205**	376101.0
	371268	375873	378121	373524	374314.3	**370646**	372685.9	371334	372196.9
	375348	378648	380811	374705	377063.5	373826	375473.3	**373689**	375352.8
	376353	378691	382177	376998	378533.3	376807	377286.8	**375188**	377238.7
	387189	391433	393051	388058	389421.6	**386803**	388238	387582	388201.5
	377729	381638	383543	378474	380306.2	377730	378537.1	**377113**	378191.8
	381623	384637	386841	383283	384028.7	380773	382454.5	**380725**	382831.5

Table 4
Two-sided paired *t*-test for the best performing algorithms.

Algorithms compared	*Min*	*Avg*
DABC vs EDA	$p = 0.277$	Not available
DABC vs tsGLS	$p = 0.000$	$p = 0.000$
DABC vs hGLS	$p = 0.001$	$p = 0.000$
DABC vs hDDE	$P = 0.024$	$p = 0.059$

parameters of the DABC algorithm, the population size was also fixed at $NP = 10$. The sizes of employed bees, onlooker bees, and scout bees were $NP = 10$, $2 \times NP$ and $0.2 \times NP$, respectively. Strategies were determined as explained in Section 2. $R = 10$ runs were carried out for each problem instance. The minimum (*Min*), average (*Avg*), maximum (*Max*), and standard deviation (*Std*) of 10 replications are reported and compared to those yielded by the best performing algorithms from the literature.

As both DABC and hDDE algorithms extensively make use of the IG_RS algorithm from [33] in different manners in terms of its parameters, we also coded the traditional IG_RS algorithm with the destruction size of $d = 4$. In addition, we employed

Table 5
Computational time of algorithms compared.

$n \times m$	tsGLS	hGLS	VNS	EDA	hDDE	DABC
20 × 5	0.50	0.11	0.13	0.30	0.44	0.18
20 × 10	0.82	0.22	0.30	1.29	1.32	0.92
20 × 20	1.41	0.39	0.74	1.51	1.62	1.84
50 × 5	32.74	16.55	31.98	57.22	43.46	66.11
50 × 10	59.28	40.33	56.18	105.45	84.94	111.74
50 × 20	128.99	82.23	142.08	240.96	175.00	201.99
100 × 5	191.54	94.17	174.26	124.55	174.81	185.59
100 × 10	383.40	251.62	324.37	266.02	340.21	364.18
100 × 20	816.45	588.86	644.98	570.27	659.66	734.03
Avg	179.46	119.39	152.78	151.95	164.61	185.18
Machine	AMD 1.83 GHz C++	AMD 1.83 GHz C++	PIV 3.2 GHz C++	PIV 3.2 GHz C++	PIV 3.0 GHz C++	PIV 3.0 GHz C++

Table 6
New best known solutions with CPU time limit of T_{max} = 3 × n × m s.

$n \times m$	New best known	Algorithms
20 × 5	14033	All
	15151	All
	13301	All
	15447	All
	13529	All
	13123	All
	13557	All
	13948	All
	14295	All
	12943	All
20 × 10	20911	All
	22440	All
	19833	All
	18710	All
	18641	All
	19245	All
	18363	All
	20241	All
	20330	All
	21320	All
20 × 20	33623	All
	31587	All
	33920	All
	31661	All
	34557	All
	32564	All
	32922	All
	32412	All
	33600	All
	32262	All
50 × 5	64803	hGLS, IG_RS, hDDE, DABC
	68051	hDDE, DABC
	63162	hDDE, DABC
	68226	DABC
	69360	hDDE, DABC
	66841	IG_RS, hDDE, DABC
	66253	hDDE
	64359	hDDE
	62981	EDA
	68811	hGLS
50 × 10	87143	hDDE
	82820	hGLS
	79987	tsGLS, hGLS
	86466	hGLS
	86391	hGLS
	86637	IG_RS
	88807	DABC
	86727	DABC
	85441	DABC
	87998	hDDE, DABC
50 × 20	125831	EDA
	119247	EDA
	116459	hGLS, DABC
	120746	DABC
	118184	hDDE
	120586	DABC
	123018	hGLS, DABC
	122520	hGLS, DABC
	121872	EDA, hGLS, DABC
	123954	IG_RS, DABC
100 × 5	253332	hDDE
	242911	hDDE
	238159	DABC
	227931	hDDE
	240647	DABC

Table 6 (*continued*)

$n \times m$	New best known	Algorithms
	232932	hDDE
	240519	DABC
	231267	DABC
	248329	DABC
	243318	hDDE
100 × 10	299227	DABC
	274967	DABC
	288450	DABC
	301676	DABC
	285537	hDDE
	270746	DABC
	280368	DABC
	291498	DABC
	303136	DABC
	292031	DABC
100 × 20	367062	DABC
	372917	DABC
	370685	DABC
	372625	DABC
	370281	IG_RS
	372122	hDDE
	374151	DABC
	385434	DABC
	375320	hDDE
	379650	DABC

the same local search based only on the insertion neighborhood structure as in Ruiz and Stutzle [33]. For the short-term search of $T_{\max} = 0.4 \times n \times m$ s, the computational results are given in Table 1.

As seen in Table 1, the hDDE and DABC algorithms generated significantly better results than the IG_RS algorithm. In particular, for the first 30 instances belonging to 20 × 5, 20 × 10, and 20 × 20 classes, the DABC and hDDE algorithms were able to find the best-known solutions for each of the 10 replications. In other words, their standard deviations were all zero, whereas IG_RS did not necessarily succeed in finding the best known solutions. In terms of all statistics, DABC and hDDE are superior to IG_RS. However, this conclusion should be tested statistically. For this reason, we carried out a two-sided paired *t*-test for the algorithms compared, with the results shown in Table 2.

Table 2 confirms that hDDE and DABC algorithms are significantly better than IG_RS in every statistic because the *p*- values were all zero at the $\alpha = 0.05$ level. When hDDE and DABC are compared, DABC generated significantly better results than hDDE in terms of *Min* values because the *p*-value was 0.024. However, the *t*-test indicates that the DABC and hDDE algorithms were equivalent in terms of *Max* and *Avg* values because the *p*-values were higher than the $\alpha = 0.05$ level. As for the *Std* values, the *t*-test indicates that DABC was more robust than hDDE because the *p*-value was equal to 0.004 at the $\alpha = 0.05$ level. From this analysis, it can be concluded that the DABC and hDDE algorithms were statistically superior to the IG_RS algorithm.

Table 3 compares the best performing algorithms from prior literature in terms of *Min* and *Avg* values because the hGLS and tsGLS algorithms only reported these statistics (EDA reported only *Min* results).

From Table 3, DABC and hDDE algorithms are clearly shown to be superior to the hGLS and tsGLS algorithms in terms of the quality of the best solutions, i.e., *Min* results. However, the DABC performance was equivalent to EDA. Again, in terms of *Avg* results, DABC generated better results than the hGLS and tsGLS algorithms. However, hDDE was very competitive with DABC in the case of *Avg* values. These conclusions should be tested statistically, too. For this reason, we again carried out a two-sided paired *t*-test for the competing algorithms. The *t*-test results are given in Table 4.

From the *t*-tests in Table 4, it can be concluded that EDA and DABC were statistically equivalent algorithms in terms of *Min* values because the *p*-value was 0.277 at the $\alpha = 0.05$ level. Note that *Avg* values were not provided for the EDA in [13]. Furthermore, DABC was able to generate statistically better results than tsGLS, hGLS, and hDDE because all of the *p*-values were less than the $\alpha = 0.05$ level. In terms of the *Avg* values, DABC was again statistically superior to both tsGLS and hGLS; however, it was equivalent to hDDE. From the analysis above, it can be concluded that the currently best performing algorithms for PFSP with TFT criterion in the existing literature are the DABC, hDDE, and EDA algorithms. Ultimately, with the short-term search results given in Table 3, 44 out of the 90 best known solutions are further improved by the DABC and hDDE algorithms. These results indicate that the performances of the DABC and hDDE algorithms in terms of obtaining the best-known solutions are quite remarkable.

Table 5 summarizes the computational times of the algorithms compared. From Table 5, the DABC algorithm was the most computationally expensive one. On the other hand, hDDE was very competitive with EDA and VNS from Jarboui et al. [13]. The least expensive one was hGLS from Tseng and Lin [24]. However, the hDDE and DABC algorithms yielded superior solutions at the expense of only minimally longer CPU times. Furthermore, the long-term search results justify the competitive performance of our algorithms when compared to the best performing EDA, hGLS, and tsGLS algorithms in the existing literature.

The computational results for the long-term search of $T_{\max} = 3 \times n \times m$ s are given in Table 6, where the results for hGLS, tsGLS, and EDA were taken from [13], [24], and [25], respectively. The new best-known solutions are reported for almost 60 out of 90 problem instances in Table 5. It should be stated that in Tseng and Lin [24], extremely long runs are carried out for 50×5, 50×10, and 50×20 instances, and the current best-known solutions are further improved at the expense of extremely increased CPU times. For $50 \times 5$and 50×10 instances, Tseng and Lin's runs took 30 min and 75 min per run, respectively. However, our algorithms took only $3 \times 50 \times 5 = 750$ s (12.5 min) and $3 \times 50 \times 10 = 1500$ s (25 min) per run for the 50×5 and 50×10 problems, respectively. Furthermore, it is again interesting to note that in 10 out of 30 problems (i.e., 50×5, 50×10, and 50×20 problems), our results with the short-term search are even better than the long-term search of Tseng and Lin [24,25]. When considering the long-term searches, the DABC and hDDE algorithms have further improved 18 out of the 30 best-known solutions of Tseng and Lin [24].

6. Conclusions

In this paper, we considered the applications of the DABC and hDDE algorithms to the PFSP under the TFT criterion. The DABC algorithm is hybridized with a variant of iterated greedy algorithms employing a local search procedure based on insertion and swap neighborhood structures. In addition, we also presented a hybrid version of our previous discrete differential evolution algorithm employing the same local search procedure. To the best of our knowledge, our proposal is the first application of DABC to the PFSP with the TFT criterion. The performances of the proposed algorithms were tested by using Taillard's benchmark suite that is commonly used in the scheduling literature. The proposed algorithms were superior to the traditional IG_RS algorithm, and it has been shown that the performances of the DABC and hDDE algorithms are highly competitive with (if not better than) the best performing estimation distribution and genetic local search algorithms that have appeared recently in the existing literature. Ultimately, 44 out of the 90 best-known solutions provided recently by the EDA, tsGLS, and hGLS algorithms are further improved by the DABC and hDDE algorithms with short-term searches. With long-term searches, the new best-known solutions are reported for all problems in the benchmark suite of Taillard. For future research, DABC will be extended to solve other scheduling problems such as no-wait flowshop, no-idle flowshop, blocking flowshop, etc. in the existing literature.

References

[1] B. Akay, D. Karaboga, A modified artificial bee colony algorithm for real-parameter optimization, Information Sciences (2010), doi:10.1016/j.ins.2010.07.015.
[2] A. Allahverdi, T. Aldowaisan, New heuristics to minimize total completion time in m-machine flowshops, International Journal of Production Economics 77 (2002) 71–83.
[3] T.C.E. Cheng, P.-J. Lai, C.-C. Wu, W.-C. Lee, Single-machine scheduling with sum-of-logarithm-processing-times-based learning considerations, Information Sciences 179 (2009) 3127–3135.
[4] S.P. Bansal, Minimizing the sum of completion times of n-jobs over m-machines in a flowshop – a branch and bound approach, AIIE Transactions 9 (1997) 306–311.
[5] C.S. Chung, J. Flynn, O. Kirca, A branch and bound algorithm to minimize the total flow time for m-machine permutation flowshop problems, International Journal of Production Economics 79 (2002) 185–196.
[6] X. Dong, H. Huang, P. Chen, An iterated local search algorithm for the permutation flowshop problem with total flowtime criterion, Computers and Operations Research 36 (2009) 1664–1669.
[7] J.M. Framinan, R. Leisten, An efficient constructive heuristic for flowtime minimisation in permutation flow shops, Omega 31 (2003) 311–317.
[8] J.M. Framinan, R. Leisten, R. Ruiz-Uzano, Comparison of heuristics for flowtime minimisation in permutation flowshops, Computers and Operations Research 32 (2005) 1237–1254.
[9] Y. Gajpal, C. Rajendran, An ant-colony optimization algorithm for minimizing the completion-time variance of jobs in flowshops, International Journal of Production Economics 101 (2006) 259–272.
[10] C.J. Ho, J.N.D. Gupta, Flowshop scheduling with dominant machines, Computers and Operations Research 22 (1995) 237–246.
[11] E. Ignall, L. Schrage, Application of the branch and bound technique to some flow-shop scheduling problems, Operations Research 13 (1965) 400–412.
[12] B. Jarboui, S. Ibrahim, P. Siarry, A. Rebai, A combinatorial particle swarm optimisation for solving permutation flowshop problems, Computers and Industrial Engineering 54 (2008) 526–538.
[13] B. Jarboui, M. Eddaly, P. Siarry, An estimation of distribution algorithm for minimizing the total flowtime in permutation flowshop scheduling problems, Computers and Operations Research 36 (2009) 2638–2646.
[14] M. Kang, D.I. Kang, J. Suh, J. Lee, An energy-efficient real-time scheduling scheme on dual-channel networks, Information Sciences 178 (2008) 2553–2563.
[15] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005.
[16] 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.
[17] D. Karaboga, B. Basturk, On the performance of artificial bee colony (ABC) algorithm, Applied Soft Computing 8 (2008) 687–697.
[18] D. Karaboga, A new design method based on artificial bee colony algorithm for digital IIR filters, Journal of The Franklin Institute 346 (2009) 328–348.
[19] 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.
[20] I. Kassabalidis, M.A. El-Sharkawi, R.J. II Marks, P. Arabshahi, A.A. Gray, Swarm intelligence for routing in communication networks, Global Telecommunications Conference, GLOBECOM'01, IEEE 6 (2001) 3613–3617.
[21] J. Kennedy, R.C. Eberhart, Y. Shi, Swarm Intelligence, San Mateo, Morgan Kaufmann, CA, USA, 2001.

[22] 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.
[23] X. Li, Q. Wang, C. Wu, Efficient composite heuristics for total flowtime minimization in permutation flowshops, Omega 37 (2009) 155–164.
[24] L.-Y. Tseng, Y.-T. Lin, A hybrid genetic local search algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research 198 (2009) 84–92.
[25] L.-Y. Tseng, Y.-T. Lin, A genetic local search algorithm for minimizing total flowtime in the permutation flowshop scheduling problem, International Journal of Production Economics (2010), doi:10.1016/j.ijpe.2010.05.003.
[26] J. Liu, C.R. Reeves, Constructive and composite heuristic solutions to the $P|\sum C_i$ scheduling problem, European Journal of Operational Research 132 (2001) 439–452.
[27] J.A. Lozano, P. Larranaga, I. Inza, E. Bengoetxea, Towards a New Evolutionary Computation Advances on Estimation of Distribution Algorithms, Springer, Berlin, 2006.
[28] M. Nawaz, E.E. Enscore Jr., I.A. Ham, Heuristic algorithm for the m-machine, n-job flow shop sequencing problem, OMEGA 11 (1983) 91–95.
[29] 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, Information Sciences 181 (2011) 2455–2468.
[30] Q.K. Pan, M.F. Tasgetiren, Y.-C. Liang, A discrete differential evolution algorithm for the permutation flowshop scheduling problem, Computers and Industrial Engineering 55 (2008) 795–816.
[31] C. Rajendran, Heuristic algorithm for scheduling in a flowshop to minimize total flowtime, International Journal of Production Economics 29 (1993) 65–73.
[32] C. Rajendran, H. Ziegler, Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs, European Journal of Operational Research 155 (2004) 426–438.
[33] R. Ruiz, T. Stützle, A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem, European Journal of Operational Research 177 (2007) 2033–2049.
[34] R. Storn, K. Price, Differential evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. ICSI, Technical Report TR-95-012, 1995.
[35] E. Taillard, Benchmarks for basic scheduling problems, European Journal of Operational Research 64 (1993) 278–285.
[36] M.F. Tasgetiren, M. Sevkli, Y.-C. Liang, G. Gencyilmaz, Particle swarm optimization algorithm for permutation flowshop sequencing problem, in: Proceedings of Ant Colony Optimization and Swarm Intelligence (ANTS2004), LNCS 3172, Springer-Verlag, 2004, pp. 381–389.
[37] M.F. Tasgetiren, Y-C Liang, M. Sevkli, G. Gencyilmaz, A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem, European Journal of Operational Research 177 (2007) 1930–1947.
[38] M.F. Tasgetiren, Q-K. Pan, Y-C. Liang, A discrete particle swarm optimization algorithm for the no-wait flowshop scheduling problem with makespan and total flowtime criteria, Computers and Operations Research 35 (2008) 2807–2839.
[39] M.F. Tasgetiren, Q-K. Pan, Y-C. Liang, A discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times, Computers and Operations Research 36 (2009) 1900–1915.
[40] V. Tereshko, Reaction-diffusion model of a honeybee colony's foraging behaviour, in: PPSN VI: Proceedings of the Sixth International Conference on Parallel Problem Solving from Nature, Springer-Verlag, London, UK (2000) 807–816.
[41] S.L. Van de Velde, Minimizing the sum of the job completion times in the two-machine flow shop by Lagrangian relaxation, Annals of Operations Research 26 (1990) 257–268.
[42] V.S. Vempati, C.L. Chen, S.F. Bulllngton, An effective heuristic for flow shop problems with total flow time as criterion, Computers and Industrial Engineering 25 (1993) 219–222.
[43] L. Wang, D.Z. Zheng, An effective hybrid heuristic for flow shop scheduling, International Journal of Advanced Manufacturing Technology 21 (2003) 38–44.
[44] H.R. Wedde, M. Farooq, The wisdom of the hive applied to mobile ad-hoc networks, in: Swarm Intelligence Symposium, SIS 2005, IEEE Proceedings (2005) 341–348.
[45] H.S. Woo, D.S. Yim, A heuristic algorithm for mean flowtime objective in flowshop scheduling, Computers and Operations Research 25 (1998) 175–182.
[46] X.S. Yang, Engineering optimizations via nature-inspired virtual bee algorithms, in: Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach, LNCS 3562 (2005) 317–323.
[47] Y.Q. Yin, D.H. Xu, K.B. Sun, H.X. Li, Some scheduling problem with general position-dependent and time-dependent learning effects, Information Sciences 179 (2009) 2416–2425.
[48] Y. Zhang, X. Li, Q. Wang, Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization, European Journal of Operational Research (2008), doi:10.1016/j.ejor.2008.04.033.