Swarm and Evolutionary Computation 4 (2012) 25–32

Contents lists available at SciVerse ScienceDirect

Swarm and Evolutionary Computation

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

A swarm intelligence approach to the early/tardy scheduling problem

Shyam Sundar, Alok Singh *

Department of Computer and Information Sciences, University of Hyderabad, Hyderabad 500 046, India

ARTICLE INFO

Article history:
Received 18 August 2011
Received in revised form
1 November 2011
Accepted 8 December 2011
Available online 16 December 2011

Keywords:
Artificial bee colony algorithm
Constrained optimization
Early/tardy scheduling
Heuristic
Swarm intelligence

ABSTRACT

This paper describes an application of artificial bee colony (ABC) algorithm, which is a new swarm intelligence approach, for a version of the single machine early/tardy scheduling problem where no unforced machine idle time is allowed. A local search is used inside the ABC algorithm to further improve the schedules obtained through it. A variant of the basic ABC approach is also considered in this paper where the best solution obtained through ABC algorithm is improved further via an exhaustive local search. We have compared these two approaches with 16 heuristic approaches reported in the literature on existing set of benchmark instances as well as on some large instances. Computational results show the effectiveness of our approaches.

1. Introduction

This paper addresses a single machine scheduling problem with earliness and tardiness costs and no machine idle time. Using the same notational conventions as used in [1,2], the problem can be formulated as follows: given n independent jobs $J_1, J_2, \ldots, J_n$, each of which has to be sequentially processed without preemption on a single machine. The machine and the jobs are supposed to be continuously available. Each job J_j needs a processing time p_j on this machine and, ideally, should be finished exactly on its due date d_j. Earliness E_j and tardiness T_j of a job J_j in a schedule can be defined as $E_j = \max(0, d_j - C_j)$ and $T_j = \max(C_j - d_j, 0)$ respectively, where C_j is the time at which job J_j finishes in this schedule. The objective of the early/tardy scheduling problem (ETSP) considered in this paper is to find a schedule that minimizes

$$\sum_{j=1}^{n}(h_j E_j + w_j T_j) \quad (1)$$

where h_j and w_j are the earliness and tardiness penalties of job J_j.

The ETSP is an $\mathcal{NP}$-Hard problem, as it is a generalization of weighted tardiness scheduling problem, which is $\mathcal{NP}$-Hard [3].

Scheduling models which penalize both earliness and tardiness are based on the philosophy of just-in-time production, i.e., producing something, just at the instant it is needed. This is indeed desirable in many real life production environments. For example, producing goods early invariably incurs the holding cost. Additional problems occur with perishable goods which may spoil by the time they are needed. Similarly, producing goods late may lead to loss of sales, loss of goodwill and increased shipping cost due to rush shipping [1].

No unforced machine idle time is allowed in the scheduling model discussed here, i.e., a machine can be idle only when no jobs are available for processing. This suits those production environments where either machine has to be operated continuously to meet the demands or operating and startup costs of the machine exceed the cost of producing some jobs early. Some specific examples, which can be found in [4,5], show the importance of these kinds of scheduling models in real life scenarios.

For the ETSP, many exact and heuristic approaches have been proposed. Among exact approaches, Abdul-Rajaq and Potts [6] presented a branch and bound method in which lower bounding procedure was based on subgradient optimization and dynamic programming state-space relaxation method, while Li [7] and Liaw [8] used Lagrangian relaxation and the multiplier adjustment methods for lower bounding procedure in their branch-and-bound methods. Valente and Alves [9] further improved the lower bounds proposed by Li [7] and Liaw [8]. Tanaka [10] proposed an exact method based on successive sublimation dynamic programming. This method begins with a very basic relaxation of the original problem and then successively solves relaxations with more and more detailed information through dynamic programming.

As far as heuristic approaches for the ETSP are concerned, Ow and Morton [11] proposed several early/tardy dispatch rules and a filtered beam search procedure for the ETSP. Though filtered beam search procedure was better than early/tardy dispatch rules, but found to be too slow for large instances with more than 100

* Corresponding author. Tel.: +91 40 23134011; fax: +91 40 23010780.
E-mail addresses: mc08pc17@uohyd.ernet.in (S. Sundar), alokcs@uohyd.ernet.in (A. Singh).

2210-6502/$ – see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.swevo.2011.12.002

jobs. Li [7] proposed a heuristic procedure based on neighborhood search that is better than filtered beam search procedure of [11] in terms of both solution quality and running time. Valente and Alves [12] presented some more dispatch rules and greedy heuristics. They also used improvement procedures to further reduce the cost of the schedules obtained by the heuristics.

Valente et al. [1] proposed twelve variants of a hybrid generational genetic algorithm. All twelve genetic algorithms (GAs) were elitists, i.e., from the current generation, these GAs copy the best 10% of the population unaltered to the next generation. All these GAs employs random-key encoding [13] and parameterized uniform crossover [14]. No mutation operator was used by any of these genetic algorithms, and, therefore, some solutions are generated randomly every generation for the sake of maintaining diversity in the population. The basic version of genetic algorithm with only above mentioned features was named GA. The genetic algorithms, which employ as local search a single pass or up to eight passes (no further passes are performed if no improvement is achieved during a pass) of an adjacent interchange procedure were named GA-SAI and GA-MAI respectively. During each pass, the adjacent interchange procedure starts with the first job in the schedule and interchanges two adjacent jobs whenever doing so reduces the cost of the schedule. For each of these three versions, another variant is created that further reduces the cost of the best solution obtained by the genetic algorithm through successive application of a non-adjacent pairwise interchange procedure till no further reduction in the cost of the solution is possible. This resulted in three more versions GA-MNAI, GA-SAI-MNAI and GA-MAI-MNAI. During each pass, starting from the first job in the schedule, the non-adjacent pairwise interchange procedure tries to interchange the concerned job with all the other jobs. The interchange that leads to a schedule of minimum cost is carried out, and, then the next job in the schedule is considered. Initial population in all these six versions consist of randomly generated solutions. For each of these six versions, one more variant is created where initial population is seeded with one solution obtained through the EXP-ET dispatching procedure [11] and one solution obtained through the NSearch neighborhood search algorithm [7]. Remaining members of initial population were generated randomly. This yielded the six more versions viz. GA-INI, GA-SAI-INI, GA-MAI-INI, GA-MNAI-INI, GA-SAI-MNAI-INI, GA-MAI-MNAI-INI. These 12 genetic algorithms were compared against the EXP-ET dispatching procedure and the NSearch neighborhood search algorithm. These two methods were chosen for comparison because of the fact that the NSearch was the best heuristic approach known at that time and the EXP-ET dispatching procedure was the best among all the dispatching procedures. All the genetic algorithm variants outperformed the EXP-ET dispatching procedure in terms of solution quality by a large margin. All variants except GA, GA-INI and GA-SAI, obtained better quality solutions in comparison to the NSearch algorithm. However, all the genetic algorithm variants were slower than the NSearch algorithm and the EXP-ET dispatching procedure. The EXP-ET dispatching procedure was even faster than NSearch algorithm.

Singh [2] proposed a hybrid permutation-coded steady-state genetic algorithm for the ETSP. This genetic algorithm uses uniform order-based crossover and multiple swap mutations. Like [1], it also employs the multiple passes of adjacent interchange procedure as local search. This genetic algorithm was named SS-GA. Like MNAI variants of [1], another version of SS-GA was also considered where the best solution obtained by the SS-GA was improved through successive applications of non-adjacent pairwise interchange procedure. This latter version was called SS-GA-MNAI. Both these versions outperformed the 14 approaches (12 genetic algorithms, the EXP-ET dispatching procedure and the NSearch neighborhood search algorithm) considered in [1].

Pan et al. [15] and Tasgetiren et al. [16] describe approaches for problems closely related to ETSP. In [15], a discrete particle swarm optimization algorithm is presented for minimizing the total earliness and tardiness penalties with a common due date on a single machine, whereas [16] described a discrete differential evolution algorithm for the single machine total weighted tardiness problem with sequence dependent setup times.

In recent past, it has been seen that computational swarm intelligence techniques are playing active role in solving hard optimization problems. These techniques are based on the collective behavior and self-organization characteristics of swarm like honey bees. Inspired by intelligent foraging behavior of honey bee swarm, Karaboga [17] developed the Artificial Bee Colony (ABC) algorithm. Because of its simplicity and wide applicability, the ABC algorithm has attracted many researchers' attention in a short span of time and has been already applied successfully to many hard optimization problems [17–25]. However, not many applications of ABC algorithm exist in the literature for solving permutation problems. To our knowledge, only [24,25] describe ABC algorithms for permutation problems. Pan et al. [24] describes an application of ABC algorithm for solving the lot-streaming flowshop scheduling problem, whereas [25] applies ABC algorithm for minimizing the total flowtime in permutation flow shops. Therefore, the domain of permutation problems are under-investigated as far as ABC algorithm is concerned. This has motivated us to develop an approach based on the ABC algorithm for the ETSP, which is a permutation problem. Similar to [1,2], a variant of the basic approach is also presented where the best solution obtained through ABC approach is improved further by successive passes of non-adjacent pairwise interchange procedure. We have compared our approaches with those presented in [1,2] which are the best heuristic approaches known so far for the problem. Computational results show the effectiveness of our approaches.

The remainder of this paper is organized as follows: Section 2 provides an introduction to the ABC algorithm, Section 3 describes our approaches for the ETSP. Computational results are reported in Section 4, whereas Section 5 contains some concluding remarks.

2. Artificial Bee Colony algorithm

The ABC algorithm is a recently developed metaheuristic technique based on intelligent foraging behavior of honey bee swarm. Entomologists divide the colony of real honey bees into three groups viz. scout, employed and onlooker bees. Scout bees are those bees participating in foraging process by searching new food sources randomly around the hive. As soon as a scout discovers a food source, this scout bee is reclassified as an employed bee. The jobs of employed bees are to exploit their associated food sources. They are responsible for carrying loads of nectars from the food sources to the hive, and communicate information about their food sources to the onlooker bees by means of performing a dance in a designated area of the hive called dance area. The richness of the food source currently being exploited by the employed bee plays main role in the dance performed by it. Onlooker bees are those bees that wait in the dance area for the employed bees to share information about their food sources. Onlooker bees watch numerous dances before selecting a food source. The probability, through which an onlooker bee selects a food source, is proportional to the nectar content of that food source. Therefore, richer the nectar content of a food source, the more onlookers it attracts. Once an onlooker is associated with a food source, it is reclassified as an employed bee. Whenever, a food source is completely exhausted, then all employed bees exploiting it leave it and become either scouts or

onlookers. Hence, scout bees do the job of exploration, whereas employed bees and onlookers bees do the job of exploitation.

Inspired by this intelligent foraging behavior of honey bees, Karaboga [17] proposed the ABC algorithm. The ABC algorithm proposed initially was intended for optimization problems in continuous domain. It was extended by Singh [22] for subset selection problems and by Pan et al. [24] for permutation problems. Like real bees, in the ABC framework, artificial bees are also classified into same three groups viz. scout, employed and onlooker bees. Each food source represents a candidate solution to the problem under consideration and is associated with an employed bee. Hence, the number of employed bees is equal to the number of food sources. The nectar content of a food source represents the fitness of the candidate solution being represented by that food source.

The ABC algorithm starts with a fixed number of food sources (solutions) which are generated randomly. Then, a search process is carried out repeatedly until termination criterion is satisfied. In each iteration, each employed bee determines a new food source in the neighborhood of its associated food source. If the nectar content of this new food source is more than that of its currently associated food source, then this employed bee associates itself to this newly generated food source leaving the old one, otherwise it continues with the old one. The exact process of determination of a food source in the neighborhood of a particular food source depends on the nature of the problem being solved by the ABC algorithm, i.e., it varies from problem to problem. This phase where every employed bee determines a neighboring food source can be termed as "employed bee phase". Once this phase ends, "onlooker bee phase" begins, in which all employed bees share the nectar information of their corresponding food sources with the onlookers. In this phase, each onlooker selects a food source associated with an employed bee with the help of a probability-based selection method. The probability p_j of selecting a jth food source is computed as:

$$p_j = \frac{F_j}{\sum_{i=1}^{k} F_i} \tag{2}$$

where F_j is the nectar amount of the jth food source and k is the total number of food sources. In the genetic algorithm parlance, such a probability based selection is called "roulette wheel" selection. Due to this selection method, a food source having higher nectar content is preferred by more onlookers. Once all onlookers have selected their food sources, each of them determines a food source in the neighborhood of their selected food source in the manner similar to the employed bee phase and evaluates its fitness. Among all the neighboring food sources determined by the onlookers associated with a particular food source i and the food source i itself, the best food source is determined. This best food source becomes the new location of the food source i in the next iteration. If ith food source is not improved over a predetermined number of iterations *limit*, then this ith food source is abandoned by its associated employed bee and this employed bee becomes a scout. This scout is again converted to an employed bee by associating it with a new randomly generated food source. This randomly generated food source becomes the new location of the food source i. The next iteration of the ABC algorithm begins after the new locations of all food sources are determined.

A literature survey on the ABC algorithm and other algorithms inspired by bee swarm intelligence is given in [26].

3. ABC algorithm for the ETSP

In this section we describe the main components of our ABC algorithm for the ETSP.

3.1. Solution encoding

Since, the ETSP is inherently a linear permutation problem, therefore, we have encoded each solution by a linear permutation of jobs that denotes the ordering of execution of the jobs in that solution.

3.2. Initialization

Each initial solution, which is essentially a random permutation of n jobs, is generated by following an iterative process. Initially, let U be the set containing all n jobs and let S be an empty schedule. A job J_i is selected uniformly at random from U. This job is added to S at the first position and deleted from U. Next, iteratively a job J_j is selected from U using the roulette wheel selection method, where the probability of selecting a job is inversely proportional to sum of its earliness and tardiness costs with respect to the jobs already in S. The selected job J_j is deleted from U and added to S at the first vacant position. It is to be noted that if during the beginning of an iteration a job is found to have zero cost, then that job is selected immediately and roulette wheel selection method is not used in that iteration. This whole process is repeated again and again until U becomes empty.

Each initial solution is uniquely associated with an employed bee and the fitness of each solution is computed.

3.3. Probability of selecting a food source

In ABC algorithm, an onlooker bee chooses a food source with the help of binary tournament selection method. In this method, two different food sources are selected uniformly at random from the population. With probability b_t, better food source is selected, otherwise worse one is selected with probability $1 - b_t$.

3.4. Determination of a neighboring food source

In order to find a high quality solution, the problem structure should be exploited effectively so that the search process leads toward the optimal. As the problem-structure of the ETSP is based on permutation of n jobs, we apply two operators, i.e., multi-point insert operator and 3-point swap operator in a mutual exclusive way for determining a neighboring solution in the search space of $n!$, where n is the number of jobs. Insert operator is applied with probability p_q, otherwise swap operator is applied.

3.4.1. Multi-point insert operator

Like traditional ABC algorithms, our method for determining a neighboring solution also utilizes components from another solution. It is based on a simple observation that in case of a scheduling problem like ETSP, if a job is present at a particular position in one good schedule (solution), then the likelihood that this job is present exactly at the same position or around the same position in many other good schedules is high. To generate a solution s' in the neighborhood of a solution s, another solution t is randomly selected from employed bee population. If t is different from s then we have used the following procedure. Initially, s' is an empty schedule, i.e., all positions in s' are marked empty. nbr different positions in t are randomly chosen and jobs in these positions are inserted into the corresponding positions in s'. The remaining $n - nbr$ empty positions in s' are filled with the jobs not yet inserted. These not yet inserted jobs are sorted in ascending order according to the positions they occupy in s. Then beginning at the first empty position, these jobs are inserted one-by-one in their sort order at the empty positions in s'. nbr is a parameter to be determined empirically. Its value should be small in comparison

j	p_j	h_j	w_j	d_j
1	4	7	8	125
2	6	4	6	89
3	7	3	10	120
4	2	3	8	107
5	1	10	4	152
6	7	1	7	85
7	3	7	2	106
8	9	8	10	105
9	3	1	3	157
10	4	8	6	129
11	10	3	3	156
12	9	10	8	87
13	4	7	2	94
14	3	10	4	153
15	2	10	5	74
16	8	9	5	123
17	6	1	4	102
18	7	2	1	136
19	7	4	3	91
20	1	7	2	154

(a) 20 input jobs

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
9	1	19	18	17	16	15	14	13	12	11	10	20	8	7	6	5	4	3	2

(b) Schedule s

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
19	1	18	17	16	15	14	13	11	12	10	9	8	7	6	5	4	3	2	20

(c) Schedule t

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
9	1	19	18	17	16	15	13	14	12	11	10	20	8	6	7	5	4	3	2

(d) Schedule s′

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2

(e) Schedule before 3PS operation

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	20	19	18	17	16	15	3	13	12	11	10	9	8	7	6	5	14	4	2

(f) Schedule after 3PS operation

Fig. 1. An illustrative example for determining a neighboring food source.

to total number of jobs n, otherwise the generated solution s' may be far off from s in the search space. We have taken nbr to be equal to $\frac{n}{10}$.

If t is same as s then there is no point in using the multi-point insert operator and the 3-point swap operator which is described next is applied on s in an attempt to diversify the population.

3.4.2. 3-point swap operator

A swap operation is like a perturbation strategy which not only explores the search space but also prevents the solution, as far as possible, from trapping in a local optima. We have used 3-point swap (3PS) which was introduced in [27]. To generate a solution s' in the neighborhood of a solution s through 3PS following procedure is used. Initially, s' is a copy of s. Then three positions i, j and k are selected randomly in s', then two swap operations are applied among the jobs at these three selected positions, i.e., the job at position i is swapped with the job at position j, and then the new job at position i is swapped with the job at position k. The 3PS leads to a schedule that is at a distance (number of swaps required to convert a schedule to another) of 2 from the original schedule. This may accelerate the search toward global optimum [27] in comparison to using only a single swap.

Fig. 1, explains our multi-point insert and 3-point swap operators with the help of an example. Fig. 1(a) shows 20 input jobs along with their processing times, due dates and earliness and tardiness penalties. Fig. 1(b) and (c) show schedules s and t as defined in Section 3.4.1. $nbr = 2$ and hence two positions say positions 8 and 15 are chosen randomly. Position 8 and 15 in schedule t respectively contain job J_{13} and J_6. These jobs are inserted into the exactly same positions in schedule s' as shown in Fig. 1(d). The remaining 18 jobs are sorted in ascending order according to the position they occupy in s. Therefore, the order is $J_9, J_1, J_{19}, J_{18}, J_{17}, J_{16}, J_{15}, J_{14}, J_{12}, J_{11}, J_{10}, J_{20}, J_8, J_7, J_5, J_4, J_3, J_2$. Beginning at first empty position in schedule s', one-by-one these jobs are inserted in this order to the empty positions in s'. This leads to schedule shown in Fig. 1(d). Fig. 1(e) and (f) respectively shows the schedules before and after 3PS. In this example, $i = 8, j = 18$ and $k = 19$. The job at position 8 (J_{14}) is swapped with the job at position 18 (J_4), and then the new job at position 8 (J_4) is swapped with the job at position 19 (J_3) to get the schedule shown in Fig. 1(f).

3.5. Local search

To further improve the solution quality, we have used multiple passes of 3-swap procedure [28] as local search. A pass of 3-swap procedure starts with the first job in the schedule and considers in succession all $n - 2$ triples of consecutive jobs. All possible permutations of the jobs in a triple are considered, and then the best permutation is selected as the new ordering of the three consecutive jobs in that triple. The parameter $npass$ controls the number of passes used. However, if one complete pass fails to improve the solution quality then that means that solution cannot be improved further through 3-swap procedure, therefore, in this situation local search terminates before $npass$ passes.

It is to be noted that in order to keep the running time manageable, we have applied local search only on those solutions where the difference between its fitness and the fitness of the best solution found so far is less than $range\%$ of the fitness of the best solution found so far.

Algorithm 1: Pseudo-code of the ABC Algorithm

```
generate a population of n_e solutions E_1, E_2, ..., E_{n_e} randomly;
best ← best solution among E_1, E_2, ..., E_{n_e};
while termination criteria is not satisfied do
    for i ← 1 to n_e do
        E' ← Get_Neighboring_Solution(E_i);
        if difference in fitness of E' and best is less than range% of best
        then
            apply local search to E';
        if E' is better than E_i then
            E_i ← E';
        else if E_i has not changed over last limit iterations then
            apply swap operator on E_i;
        if E_i is better than best then
            best ← E_i;
    for i ← 1 to n_o do
        p_i ← Binary_Tournament(E_1, E_2, ..., E_{n_e});
        On_i ← Get_Neighboring_Solution(E_{p_i});
        if difference in fitness of On_i and best is less than range% of
        best then
            apply local search to On_i;
        if On_i is better than best then
            best ← On_i;
    for i ← 1 to n_o do
        if On_i is better than E_{p_i} then
            E_{p_i} ← On_i;
```

3.6. Other features

If a food source (solution) does not improve for a predetermined number of iterations *limit*, then employed bee associated with that food source abandons it and becomes a scout. The parameter *limit* is an important control parameter of the ABC algorithm as it controls the delicate balance between exploration and exploitation. Instead of generating a new food source randomly from scratch (as described in Section 3.2) for making this new scout employed again, we have generated a random solution in the neighborhood of the just abandoned solution via 3-point swap operator (as described in Section 3.4.2). The reason behind using such an strategy is that the efficiency of search increases as the random solution generated from scratch is expected to have much worse fitness than the solution obtained from an existing solution through 3-point swap operator. Besides, generating a food source from scratch requires more effort in comparison to generating it through 3-point swap operator. For an scout bee, Pan et al. [24] always generates a solution in the neighborhood of the best solution in the population. We have also tried this strategy, but we got better results with our strategy.

Another distinguishing feature of our ABC approach is that there is no upper limit on the number of scout bees in an iteration, i.e., in an iteration there may not be even a single scout bee, whereas in another iteration there may be several scout bees. In our approach, the number of scout bees in an iteration is equal to the number of those employed bees whose associated solutions have not been improved over last *limit* iterations.

Algorithm 1 gives the pseudo-code of our ABC approach for the ETSP where n_e & n_o are respectively the number of employed and onlooker bees and Get_Neighboring_Solution(X) is a function that returns a solution in the neighborhood of the solution X. Algorithm 2 gives its pseudo-code where $u01$ is a uniform variate in [0, 1]. Binary_Tournament($X_1, X_2, \ldots, X_k$) is another function that selects a solution from solutions $X_1, X_2, \ldots, X_k$ for an onlooker using binary tournament method (as described in Section 3.3) and returns the index of the solution selected.

Algorithm 2: Pseudo-code of Get_Neighboring_Solution

```
input  : A solution s
output: A neighboring solution s'
select another solution t randomly from the employed bee
population;
if ((u01 < p_q) ∧ (t is different from s)) then
    generate s' from s and t using the multi-point insert
    operator;
else
    create a copy s' of s;
    apply 3-point swap operator on s';
return s';
```

3.7. ABC-MNAI approach

Similar to MNAI variants of [1,2], we have also implemented another variant of our ABC approach where the best schedule obtained through it is further improved by repeatedly applying passes of non-adjacent pairwise interchange procedure till a complete pass fails to further improve the schedule. We call this variant ABC-MNAI.

4. Computational results

This section reports the computational results of ABC and ABC-MNAI approaches for the ETSP and compare them with other 16 approaches described in [1,2]. Our approaches have been implemented in C and executed on a Linux based 3.0 GHz core 2 duo system with 2 GB RAM. In all our computational experiments with ABC and ABC-MNAI, we have used a population of 100 bees consisting of 50 employed and 50 onlooker bees, $limit = 50$ and $b_t = 0.8$. We have taken $p_q = 0.4$, $range = 10$, $nbr = \frac{n}{10}$ and $npass = 2$. On each instance with n jobs, we have allowed our approaches to execute for 1000 iterations if $n \leq 250$, otherwise for 1500 iterations. We have chosen these parameters empirically after a large number of trials. Though these parameter values provide good results on most of the instances, they may not be optimal for all instances.

Our approaches are tested on the same instances as used in [1, 2]. These instances were generated by Valente et al. [1] in a manner similar to those used in [6–8]. Based on the number of jobs, these instances are categorized into four groups viz. instances with 15, 50, 75 and 100 jobs. Each group consists of 100 instances. Each job J_j in these instances consists of an integer processing time p_j, earliness penalty h_j and tardiness penalty w_j drawn uniformly at random from the interval [1, 10]. An integer due date d_j for each job J_j is drawn uniformly at random from the interval $[T(1 - LF - RDD/2), T(1 - LF + RDD/2)]$, where T is the sum of the processing times of all jobs, LF is the lateness factor and RDD is the range of due dates. LF is set to 0.2 and 0.4, whereas RDD is set to 0.2, 0.4, 0.6, 0.8 and 1.0. Ten instances were generated for each of the 10 combinations of these two parameters. Therefore, each group consists of 100 instances. In order to test our approaches on large instances, we have generated 4 additional groups of instances with 250, 500, 750 and 1000 jobs using the same procedure as described above.

Tables 1 and 2 compare the results of our ABC approach with each of the 16 approaches presented in [1,2]. In Table 1, comparison is done on the basis of number of instances out of 100 in each of the 8 groups where ABC performs better (B), equal (E) or worse (W) in terms of solution quality. Exact solution values of 14 approaches reported in [1] were obtained through personal communication. These 14 approaches were executed only on instances with up

Table 1
Comparisons of ABC with 16 approaches in terms of number of instances on which ABC performs better, same or worse.

	ABC																						
	$n = 15$			$n = 50$			$n = 75$			$n = 100$			$n = 250$			$n = 500$			$n = 750$			$n = 1000$		
	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W	B	E	W
EXP-ET	100	0	0	100	0	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
Nsearch	38	62	0	97	3	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA	53	47	0	100	0	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-MNAI	28	72	0	88	12	0	97	3	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-INI	34	66	0	95	5	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-MNAI-INI	29	71	0	92	8	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-SAI	1	99	0	75	25	0	99	1	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-SAI-MNAI	1	99	0	63	37	0	92	8	0	99	1	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-SAI-INI	3	97	0	69	31	0	97	3	0	97	2	1	–	–	–	–	–	–	–	–	–	–	–	–
GA-SAI-MNAI-INI	3	97	0	64	36	0	90	9	1	95	4	1	–	–	–	–	–	–	–	–	–	–	–	–
GA-MAI	0	100	0	39	61	0	79	20	1	97	3	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-MAI-MNAI	0	100	0	37	63	0	70	29	1	92	7	1	–	–	–	–	–	–	–	–	–	–	–	–
GA-MAI-INI	0	100	0	33	67	0	79	20	1	81	17	2	–	–	–	–	–	–	–	–	–	–	–	–
GA-MAI-MNAI-INI	0	100	0	32	68	0	72	27	1	74	24	2	–	–	–	–	–	–	–	–	–	–	–	–
SS-GA	0	100	0	16	84	0	31	64	5	57	38	5	85	6	9	88	2	10	89	1	10	89	0	11
SS-GA-MNAI	0	100	0	16	84	0	29	66	5	55	40	5	77	6	17	59	3	38	34	1	65	28	1	71
Overall	0	100	0	7	93	0	19	75	6	44	51	5	77	6	17	59	3	38	34	1	65	28	1	71

Table 2
Average percentage deviation and average computation times for 16 approaches and ABC.

	$n = 15$		$n = 50$		$n = 75$		$n = 100$		$n = 250$		$n = 500$		$n = 750$		$n = 1000$	
	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT
EXP-ET	18.59	0.0	15.20	0.0	12.99	0.0	14.68	0.0	–	–	–	–	–	–	–	–
Nsearch	1.15	0.0	2.66	1.8	2.32	9.9	2.52	25.5	–	–	–	–	–	–	–	–
GA	1.91	1.7	6.47	15.2	7.31	39.4	9.64	75.4	–	–	–	–	–	–	–	–
GA-MNAI	0.69	1.7	1.29	15.5	1.01	40.7	0.87	78.8	–	–	–	–	–	–	–	–
GA-INI	0.84	2.0	1.92	16.7	2.40	44.0	2.63	91.8	–	–	–	–	–	–	–	–
GA-MNAI-INI	0.64	2.0	1.15	16.9	1.26	44.9	1.16	94.1	–	–	–	–	–	–	–	–
GA-SAI	0.00	2.3	1.37	21.9	2.24	63.4	3.13	114.5	–	–	–	–	–	–	–	–
GA-SAI-MNAI	0.00	2.3	0.55	22.1	0.90	64.4	1.02	117.1	–	–	–	–	–	–	–	–
GA-SAI-INI	0.02	2.8	0.67	26.7	1.21	74.5	1.30	148.1	–	–	–	–	–	–	–	–
GA-SAI-MNAI-INI	0.02	2.8	0.48	26.9	0.72	75.1	0.71	149.9	–	–	–	–	–	–	–	–
GA-MAI	0.00	3.6	0.26	46.6	0.68	130.0	0.98	267.7	–	–	–	–	–	–	–	–
GA-MAI-MNAI	0.00	3.6	0.21	46.7	0.43	130.6	0.53	269.6	–	–	–	–	–	–	–	–
GA-MAI-INI	0.00	3.9	0.26	50.8	0.58	130.5	0.64	260.2	–	–	–	–	–	–	–	–
GA-MAI-MNAI-INI	0.00	3.9	0.25	50.9	0.53	130.9	0.46	261.5	–	–	–	–	–	–	–	–
SS-GA	0.00	0.4	0.04	0.9	0.09	1.2	0.16	1.8	0.05	6.7	0.09	26.6	0.17	64.7	0.27	117.6
SS-GA-MNAI	0.00	0.4	0.04	0.9	0.08	1.2	0.09	1.8	0.02	7.0	0.01	28.8	0.00	66.2	0.00	121.0
ABC	0.00	0.1	0.00	0.3	0.01	0.7	0.01	1.2	0.01	7.0	0.01	31.8	0.03	56.5	0.04	90.8

to 100 jobs. Singh [2] also reports the results of SS-GA and SS-GA-MNAI on instances with up to 100 jobs only. As the source codes for these two approaches were available, we have executed these two approaches along with ABC and ABC-MNAI on 4 new groups of large instances. In addition, SS-GA and SS-GA-MNAI have been re-executed on older instances with 15, 50, 75 and 100 jobs so that their execution times can be compared exactly with those of ABC and ABC-MNAI. The last row in Table 1 compare the solution obtained through ABC approach with the best solution among all the 16 approaches for instances up to 100 jobs and with the best solution among SS-GA and SS-GA-MNAI for the remaining instances. Table 2 compares the ABC approach with other approaches in terms of average percentage deviation (APD) and average computation time (ACT) in seconds on each of the eight groups of instances. On a particular instance let A be the solution obtained by an approach and let B be the best known or optimal solution (if known), then, the percentage deviation of this approach on this instance is defined as $\frac{(A-B)}{B} \times 100$. Reported APD values are average of percentage deviations over 100 instances of each group. Only on instances with 15 jobs, we know the optimum solutions [1]. For the remaining groups, we have taken the best solution among all the approaches applied to instances of that group as the best known solution and computed the APD values with respect to it. As the ABC approach improved the best known solution values on instances with 50, 75 and 100 jobs, therefore, the APD values reported for these groups of instances differ from those in [1,2].

As can be seen from the Tables 1 and 2, ABC is clearly better than all the other approaches on instances of Valente et al. [1] in terms of solution quality. As shown by the last row of Table 1, for these instances even the 16 approaches taken together cannot compete with the ABC approach. Our approach improves the best known solution values of 7, 19 and 44 instances respectively belonging to groups with 50, 75 and 100 jobs. The APD values for our ABC approach are also least among all the approaches for these instances. As far as performance of ABC on large instances is concerned, SS-GA-MNAI performs better than the ABC on instances of size 750 and 1000, whereas ABC performs better on instances with 250 and 500 jobs. For instances of size 750 and 1000, the respective APD values are 0.03 and 0.04 for ABC indicating that solutions obtained through ABC approach is only slightly inferior than the best solutions for these instances. ABC anyway is better than SS-GA on all 4 groups of large instances. Therefore, the superior performance of SS-GA-MNAI on large instances can

Table 3
Comparisons of ABC-MNAI with 16 approaches in terms of number of instances on which ABC-MNAI performs better, same or worse.

	ABC-MNAI																							
	$n = 15$			$n = 50$			$n = 75$			$n = 100$			$n = 250$			$n = 500$			$n = 750$			$n = 1000$		
	B	*E*	*W*	*B*	*E*	*W*	*B*	*E*	*W*	*B*	*E*	*W*	*B*	*E*	*W*	*B*	*E*	*W*	*B*	*E*	*W*	*B*	*E*	*W*
EXP-ET	100	0	0	100	0	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
Nsearch	38	62	0	97	3	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA	53	47	0	100	0	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-MNAI	28	72	0	88	12	0	97	3	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-INI	34	66	0	95	5	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-MNAI-INI	29	71	0	92	8	0	100	0	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-SAI	1	99	0	75	25	0	99	1	0	100	0	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-SAI-MNAI	1	99	0	63	37	0	92	8	0	99	1	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-SAI-INI	3	97	0	69	31	0	97	3	0	97	2	1	–	–	–	–	–	–	–	–	–	–	–	–
GA-SAI-MNAI-INI	3	97	0	64	36	0	90	9	1	95	4	1	–	–	–	–	–	–	–	–	–	–	–	–
GA-MAI	0	100	0	39	61	0	80	20	0	97	3	0	–	–	–	–	–	–	–	–	–	–	–	–
GA-MAI-MNAI	0	100	0	37	63	0	71	29	0	92	7	1	–	–	–	–	–	–	–	–	–	–	–	–
GA-MAI-INI	0	100	0	33	67	0	79	21	0	81	17	2	–	–	–	–	–	–	–	–	–	–	–	–
GA-MAI-MNAI-INI	0	100	0	32	68	0	72	28	0	74	24	2	–	–	–	–	–	–	–	–	–	–	–	–
SS-GA	0	100	0	16	84	0	31	65	4	57	38	5	87	7	6	94	2	4	98	0	2	97	0	3
SS-GA-MNAI	0	100	0	16	84	0	29	67	4	55	40	5	84	9	7	81	3	16	84	0	16	83	0	17
Overall	0	100	0	7	93	0	19	76	5	44	51	5	84	9	7	81	3	16	84	0	16	83	0	17

Table 4
Average percentage deviation and average computation times for 16 approaches and ABC-MNAI.

	$n = 15$		$n = 50$		$n = 75$		$n = 100$		$n = 250$		$n = 500$		$n = 750$		$n = 1000$	
	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT	APD	ACT
EXP-ET	18.59	0.0	15.20	0.0	12.99	0.0	14.69	0.0	–	–	–	–	–	–	–	–
Nsearch	1.15	0.0	2.66	1.8	2.32	9.9	2.53	25.5	–	–	–	–	–	–	–	–
GA	1.91	1.7	6.47	15.2	7.31	39.4	9.64	75.4	–	–	–	–	–	–	–	–
GA-MNAI	0.69	1.7	1.29	15.5	1.01	40.7	0.88	78.8	–	–	–	–	–	–	–	–
GA-INI	0.84	2.0	1.92	16.7	2.40	44.0	2.63	91.8	–	–	–	–	–	–	–	–
GA-MNAI-INI	0.64	2.0	1.15	16.9	1.26	44.9	1.16	94.1	–	–	–	–	–	–	–	–
GA-SAI	0.00	2.3	1.37	21.9	2.24	63.4	3.14	114.5	–	–	–	–	–	–	–	–
GA-SAI-MNAI	0.00	2.3	0.55	22.1	0.90	64.4	1.02	117.1	–	–	–	–	–	–	–	–
GA-SAI-INI	0.02	2.8	0.67	26.7	1.21	74.5	1.31	148.1	–	–	–	–	–	–	–	–
GA-SAI-MNAI-INI	0.02	2.8	0.48	26.9	0.72	75.1	0.71	149.9	–	–	–	–	–	–	–	–
GA-MAI	0.00	3.6	0.26	46.6	0.68	130.0	0.99	267.7	–	–	–	–	–	–	–	–
GA-MAI-MNAI	0.00	3.6	0.21	46.7	0.43	130.6	0.53	269.6	–	–	–	–	–	–	–	–
GA-MAI-INI	0.00	3.9	0.26	50.8	0.58	130.5	0.65	260.2	–	–	–	–	–	–	–	–
GA-MAI-MNAI-INI	0.00	3.9	0.25	50.9	0.53	130.9	0.47	261.5	–	–	–	–	–	–	–	–
SS-GA	0.00	0.4	0.04	0.9	0.09	1.2	0.16	1.8	0.05	6.7	0.10	26.6	0.18	64.7	0.28	117.6
SS-GA-MNAI	0.00	0.4	0.04	0.9	0.08	1.2	0.09	1.8	0.02	7.0	0.02	28.8	0.01	66.2	0.01	121.0
ABC-MNAI	0.00	0.1	0.00	0.3	0.01	0.7	0.01	1.2	0.00	7.1	0.00	32.5	0.00	59.4	0.00	98.6

be attributed to repeated application of passes of non-adjacent pairwise interchange procedure. This fact has motivated us to consider ABC-MNAI.

Our ABC approach along with SS-GA and SS-GA-MNAI have been executed on a Linux based 3.0 GHz core 2 duo system with 2 GB RAM. As shown in Table 2, ABC is clearly faster than SS-GA and SS-GA-MNAI except for instances of size 250 and 500 where it is slightly slower. Approaches of Valente et al. [1] were executed on a Pentium II, 350 MHz system. As these approaches were executed on a different system, therefore it is not possible to exactly compare the execution times of these approaches with ABC. However, we can safely say that our ABC approach is faster than the 12 GA variants of [1] on instances with 50, 75 and 100 jobs.

Tables 3 and 4 compare our ABC-MNAI approach with each of the 16 approaches presented in [1,2]. Tables 3 and 4 are respectively similar in content to Tables 1 and 2 except for the fact that best solution values obtained through ABC-MNAI is used in APD computations wherever these values are better than all other 16 approaches. These tables clearly show the superiority of ABC-MNAI in terms of solution quality over all other approaches on all groups of instances. In particular, ABC-MNAI is better than SS-GA-MNAI on all 4 groups of large instances in terms of solution quality. As shown by the last row of Table 3, for each of the 8 groups of instances, ABC-MNAI performs even better than all other approaches applied to that group taken together. ABC-MNAI also has smallest APD values for all groups of instances. These two facts taken together show the robustness of ABC-MNAI. Regarding comparison of execution times, conclusions on the lines similar to ABC can be drawn here also.

As far as comparison between ABC-MNAI and ABC is concerned, ABC-MNAI improves 2, 1, 49, 90, 97, 98 solutions obtained through ABC on each group of 100 instances with 75, 100, 250, 500, 750, 1000 jobs respectively. As ABC-MNAI improves the best solution obtained by ABC, execution time of ABC-MNAI is always slightly more than ABC.

To test the statistical significance of the results obtained by ABC-MNAI vis-à-vis any other approach, we have used the sign test [29,30]. In order to use this test for comparing ABC-MNAI with any other method, we have to count the number of instances, say N_1, on which ABC-MNAI obtains better quality solution than the other method and the number of instances, say N_2, on which ABC-MNAI obtains the same solution as the other method. Let $N_t = N_1 + \lfloor \frac{N_2}{2} \rfloor$ and let N be the total number of instances. If

$N_t \geq \frac{N}{2} + 1.96 \times \frac{\sqrt{N}}{2}$ then ABC-MNAI is significantly better with $p < 0.05$ [30]. In each group, we have 100 instances, i.e., $N = 100$, and, therefore, if $N_t \geq 59.8$ then ABC-MNAI is significantly better with $p < 0.05$ for that group. As can be easily seen from Table 3, except for the results of ABC-MNAI on two smallest groups of instances with 15 and 50 jobs, results of ABC-MNAI are statistically significant vis-à-vis any other method proposed in [1,2] with $p < 0.05$. As ABC-MNAI improves the best solution obtained through ABC, improvement, if any, in solution quality of ABC by the ABC-MNAI cannot be attributed to random fluctuations, and, hence, statistical significance is irrelevant when we compare ABC-MNAI with ABC.

5. Conclusions

In this paper we have developed two artificial bee colony based hybrid approaches viz. ABC and ABC-MNAI. We have tested our approaches against 16 previously proposed approaches [1,2] not only on existing set of instances, but also on large instances. The ABC approach performed better than all the 16 methods on existing set of instances, but it performed worse on two largest group of instances with 750 and 1000 jobs in comparison to SS-GA-MNAI approach of [2]. This motivated the development of ABC-MNAI, which is same as ABC except for the fact that the best solution obtained through ABC approach is improved further through successive application of passes of non-adjacent pairwise interchange procedure. ABC-MNAI outperformed all other approaches in terms of solution quality. It also requires less execution time in comparison to other best performing approaches on most group of instances. We have shown that for all 8 groups of instances, even if we compare the results of ABC-MNAI approach against the best results among all the approaches applied to that group, our results are still better.

As a future work, we intend to extend our approaches to other variants of early/tardy scheduling problem. Ideas presented in this paper can be used for designing artificial bee colony algorithm based approaches for other permutation problems also.

Acknowledgments

We thank Dr. Jorge M. S. Valente for providing the ETSP test instances along with their solution values obtained by each of the 14 approaches considered in Valente et al. [1]. We are grateful to Department of Science & Technology, Government of India for their financial support to carry out this research work. We also thank the associate editor and three anonymous reviewers for their insightful comments and suggestions which helped considerably in improving the quality of this paper.

References

[1] J.M.S. Valente, J.F. Gonçalves, R.A.F.S. Alves, A hybrid genetic algorithm for the early/tardy scheduling problem, Asia-Pacific Journal of Operational Research 23 (2006) 393–405.
[2] A. Singh, A hybrid permutation-coded evolutionary algorithm for the early/tardy scheduling problem, Asia-Pacific Journal of Operational Research 27 (2010) 713–725.
[3] J.K. Lenstra, A.H.G. Rinooy Kan, P. Brucker, Complexity of machine scheduling problems, Annals of Discrete Mathematics 1 (1977) 343–362.
[4] K. Korman, A pressing matter, Video (1994) 46–50.
[5] K. Landis, Group technology and cellular manufacturing in the Westvaco Los Angeles VH department, Project Report in IOM 581, School of Business, University of Southern California, 1993.
[6] T.S. Abdul-Razaq, C.N. Potts, Dynamic programming state-space relaxation for single-machine scheduling, Journal of the Operational Research Society 39 (1988) 141–142.
[7] G. Li, Single machine earliness and tardiness scheduling, European Journal of Operational Research 96 (1977) 546–558.
[8] C.F. Liaw, A branch-and-bound algorithm for the single machine earliness and tardiness scheduling problem, Computers & Operations Research 26 (1999) 679–693.
[9] J.M.S. Valente, R.A.F.S. Alves, Improved lower bounds for the early/tardy scheduling problem with no idle time, Journal of the Operational Research Society 56 (2005) 604–612.
[10] S. Tanaka, An exact algorithm for single-machine scheduling without idle time, in: Proceedings of the third Multidisciplinary International Scheduling Conference, *MISTA* 2007, 2007, pp. 614–617.
[11] P.S. Ow, T.E. Morton, The single machine early-tardy problem, Management Sciences 35 (1989) 177–191.
[12] J.M.S. Valente, R.A.F.S. Alves, Improved heuristics for the early/tardy scheduling problem with no idle time, Computers & Operations Research 32 (2005) 557–569.
[13] J.C. Bean, Genetic algorithms and random keys for sequencing and optimization, ORSA Journal of Computing 6 (1994) 154–160.
[14] W.M. Spears, K.A. DeJong, On the virtues of parameterized uniform crossover operator, in: Proceedings of the fourth International Conference on Genetic Algorithms, 1991, pp. 230–236.
[15] Q.-K. Pan, M.F. Tasgetiren, Y.C. Liang, Minimizing total earliness and tardiness penalties with a common due date on a single-machine using a discrete particle swarm optimization algorithm, Lecture Notes in Computer Science 4150 (2006) 460–467.
[16] 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 & Operations Research 36 (2009) 1900–1915.
[17] D. Karaboga, An idea based on honey bee swarm for numerical optimization, Technical Report TR06, Computer Engineering Department, Erciyes University, Turkey, 2005.
[18] B. Basturk, D. Karaboga, An Artificial bee colony (ABC) algorithm for numeric function optimization, in: Proceeding of the IEEE Swarm Intelligence Symposium, Indianapolis, In, USA, 2006.
[19] 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.
[20] D. Karaboga, B. Basturk, Artificial Bee Colony (ABC) optimization algorithm for solving constrained optimization problems, Lecture notes in Artificial Intelligence 4529 (2007) 789–798.
[21] D. Karaboga, B. Basturk, On the performance of Artificial Bee Colony (ABC) algorithm, Applied Soft Computing 8 (2008) 687–697.
[22] A. Singh, An Artificial Bee Colony algorithm for the leaf-constrained minimum spanning tree problem, Applied Soft Computing 9 (2009) 625–631.
[23] S. Sundar, A. Singh, A swarm intelligence approach to the quadratic minimum spanning tree problem, Information Sciences 180 (2010) 3182–3191.
[24] 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.
[25] M.F. Tasgetiren, Q.-K. Pan, P.N. Suganthan, H.-L. Chen Angela, A discrete Artificial Bee Colony algorithm for the total flowtime minimization in permutation flow shops, Information Sciences 181 (2011) 3459–3475.
[26] D. Karaboga, B. Akay, A survey: algorithms simulating bee swarm intelligence, Artificial Intelligence Review 31 (2009) 61–85.
[27] K.-H. Liang, X. Yao, C. Newton, D. Hoffman, A new evolutionary approach to cutting stock problems with and without contiguity, Computers & Operations Research 29 (2002) 1641–1659.
[28] J.M.S. Valente, R.A.F.S. Alves, Heuristics for the single machine scheduling problem with quadratic earliness and tardiness penalties, Computers & Operations Research 35 (2008) 3696–3713.
[29] D.J. Sheskin, Handbook of Parametric and Nonparametric Statistical Procedures, fourth ed. Chapman & Hall, 2006.
[30] J. Derrac, S. García, D. Molina, F. Herrera, A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms, Swarm and Evolutionary Computation 1 (2011) 3–18.