Using the Bees Algorithm to schedule jobs for a machine

D.T. Pham, E. Koç, J.Y. Lee, J. Phrueksanant
Manufacturing Engineering Centre, Cardiff University, Cardiff, CF24 3AA, UK

Abstract

The *Bees Algorithm* is a new population-based optimisation algorithm that mimics the natural foraging behaviour of a swarm of bees. In its basic version, the algorithm performs a kind of neighbourhood search combined with a random search to enable it to locate the global optimum. This paper describes the application of the Bees Algorithm to a combinatorial optimisation problem, the scheduling of jobs with a common due date for a machine to minimise the penalties associated with early or late completion. Following a brief description of the algorithm, the paper presents the results obtained and compares them with those of other existing optimisation techniques to demonstrate the efficiency and robustness of the new algorithm.

Keywords: Bees Algorithm, Combinatorial Optimisation, Job Scheduling, Swarm Intelligence

1 Introduction

Combinatorial optimisation problems have attracted much attention of researchers over the years. Many of combinatorial optimization problems are NP-hard [1, 2], which means they cannot be solved to optimality within polynomially-bounded computation times. Several algorithms, including population-based algorithms, have been developed that can find near-optimal solutions within reasonable running times. The swarm-based algorithm introduced in this paper to solve a single machine scheduling problem is a new population-based search algorithm capable of locating good solutions efficiently. The algorithm is inspired by the food foraging behaviour of honey bees and could be regarded as an “intelligent” optimisation tool [3].

The paper is organised as follows. Section 2 reviews related work in the area of intelligent optimisation. Section 3 details the foraging behaviour of bees and the core ideas of the proposed Bees Algorithm. Section 4 briefly describes the single machine scheduling problem and recent work on that problem. Section 5 presents the Bees Algorithm for Job Scheduling and discusses the results obtained.

2 Intelligent swarm-based optimisation

Swarm-based optimisation algorithms (SOAs) mimic nature's methods to drive a search towards the optimal solution. A key difference between SOAs and direct search algorithms such as hill climbing and random walk is that SOAs use a population of solutions for every iteration instead of a single solution. As a population of solutions is processed in each iteration, the outcome is also a population of solutions. If an optimisation problem has a single optimum, SOA population members can be expected to converge to that optimum solution. However, if an optimisation problem has multiple optimal solutions, an SOA can be used to capture them in its final population. SOAs include the Ant Colony Optimisation (ACO) algorithm [4], the Genetic Algorithm (GA) [5] and the Particle Swarm Optimisation (PSO) algorithm [6].

Common to all population-based search methods is a strategy that generates variations of the solution being sought. Some search methods use a greedy criterion to decide which generated solution to retain. Such a criterion would mean accepting a new solution if, and only if, it increases the value of the objective function (assuming the given problem is one of optimisation). A very successful non-greedy population-based algorithm is the ACO algorithm which emulates the behaviour of real ants. Ants are capable of finding the shortest path from the food source to their nest using a chemical substance called a pheromone to guide their search. The pheromone is deposited on the ground as the ants move and the probability that a passing stray ant will follow this trail depends on the quantity of pheromone laid. ACO was first used for functional optimisation by Bilchev et al. [7] and further attempts were reported in Bilchev et al. [7] and Mathur et al. [8].

The GA is based on natural selection and genetic recombination. The algorithm works by choosing solutions from the current population and then applying genetic operators – such as mutation and crossover – to create a new population. The algorithm efficiently exploits historical information to speculate on new search areas with improved performance [5]. When applied to optimisation problems, the GA has the advantage of performing global search. The GA may be hybridised with domain-dependent heuristics for improved results. For example, Mathur et al. [8] describe a hybrid of the ACO algorithm and the GA for continuous function optimisation. The PSO is an optimisation procedure based on the social behaviour of groups or organisations, for example the flocking of birds or the schooling of fish [6]. Individual solutions in a population are viewed as "particles" that evolve or change their positions with time. Each particle modifies its position in search space according to its own experience and also that of a neighbouring particle by remembering the best position visited by itself and its neighbours, thus combining local and global search methods [6].

There are other SOAs with names suggestive of possibly bee-inspired operations [9-12]. However, as far as the authors are aware, those algorithms do not closely follow the behaviour of bees. In particular, they do not seem to implement the techniques that bees employ when foraging for food.

3 The Bees Algorithm

3.1 Bees in nature

A colony of honey bees can extend itself over long distances (more than 10 km) and in multiple directions simultaneously to exploit a large number of food sources [9, 10]. A colony prospers by deploying its foragers to good fields. In principle, flower patches with plentiful amounts of nectar or pollen that can be collected with less effort should be visited by more bees, whereas patches with less nectar or pollen should receive fewer bees [11, 12].

The foraging process begins in a colony by scout bees being sent to search for promising flower patches. Scout bees move randomly from one patch to another. During the harvesting season, a colony continues its exploration, keeping a percentage of the population as scout bees [10]. When they return to the hive, those scout bees that found a patch which is rated above a certain quality threshold (measured as a combination of some constituents, such as sugar content) deposit their nectar or pollen and go to the “dance floor” to perform a dance known as the “waggle dance” [9]. This mysterious dance is essential for colony communication, and contains three pieces of information regarding a flower patch: the direction in which it will be found, its distance from the hive and its quality rating (or fitness) [9, 12]. This information helps the colony to send its bees to flower patches precisely, without using guides or maps. Each individual’s knowledge of the outside environment is gleaned solely from the waggle dance. This dance enables the colony to evaluate the relative merit of different patches according to both the quality of the food they provide and the amount of energy needed to harvest it [12]. After waggle dancing on the dance floor, the dancer (i.e. the scout bee) goes back to the flower patch with follower bees that were waiting inside the hive. More follower bees are sent to more promising patches. This allows the colony to gather food quickly and efficiently. While harvesting from a patch, the bees monitor its food level. This is necessary to decide upon the next waggle dance when they return to the hive [12]. If the patch is still good enough as a food source, then it will be advertised in the waggle dance and more bees will be recruited to that source.

3.2 Proposed Bees Algorithm

As mentioned, the Bees Algorithm is an optimisation algorithm inspired by the natural foraging behaviour of honey bees to find the optimal solution [6].

```
1. Initialise population with random solutions.
2. Evaluate fitness of the population.
3. While (stopping criterion not met) //Forming new population.
4. Select sites for neighbourhood search.
5. Recruit bees for selected sites (more bees for best e sites) and evaluate
   fitnesses.
6. Select the fittest bee from each patch.
7. Assign remaining bees to search randomly and evaluate their fitnesses.
8. End While.
```

Figure 1: Pseudo code for the Bees Algorithm

Figure 1 shows the pseudo code for the algorithm in its simplest form. The algorithm requires a number of parameters to be set, namely: number of scout bees (b), number of sites selected out of b visited sites (m), number of best sites out of m selected sites (e), number of bees recruited for the best e sites (nep), number of bees recruited for the other ($m-e$) selected sites (nsp), initial size of patches (ngh) which includes the site and its neighbourhood and stopping criterion. The algorithm starts with the b scout bees being placed randomly in the search space. The fitnesses of the sites visited by the scout bees are evaluated in step 2. In step 4, bees that have the highest fitnesses are chosen as "selected bees" and sites visited by them are chosen for neighbourhood search. Then, in steps 5 and 6, the algorithm conducts searches in the neighbourhood of the selected sites, assigning more bees to search near to the best e sites. The bees can be chosen directly according to the fitnesses associated with the sites they are visiting. Alternatively, the fitness values are used to determine the probability of the bees being selected. Searches in the neighbourhood of the best e sites which represent more promising solutions are made more detailed by recruiting more bees to follow them than the other selected bees. Together with scouting, this differential recruitment is a key operation of the Bees Algorithm. However, in step 6, for each patch only the bee with the highest fitness will be selected to form the next bee population. In nature, there is no such restriction. This restriction is introduced here to reduce the number of points to be explored. In step 7, the remaining bees in the population are assigned randomly around the search space scouting for new potential solutions. These steps are repeated until a stopping criterion is met. At the end of each iteration, the colony will have two parts to its new population – representatives from each selected patch and other scout bees assigned to conduct random searches.

4 Single machine scheduling problem

Biskup and Feldmann [13] have developed Single Machine scheduling benchmarks and also proposed two new heuristics to solve the problem. The characteristic of this benchmark set is explained in the following section. The survey revealed that many approaches have been applied to solve this data set recently.

In this problem, a number of jobs will be processed without interruption on a single machine. All jobs are available at time zero, each of which has its own processing time (P_i) and needs exactly one operation. If the completion time (C_j) of job j is smaller than or equal to due date (d), the job's earliness is $E_j = d - C_j$, on the other hand, if it is greater than the due date, the job's tardiness is $T_j = C_j - d$. The goal of this problem is to find a sequence S of n jobs that minimise the total of the earliness and tardiness penalties:

$$f(S) = \sum \left(\alpha_j \cdot E_j + \beta_j \cdot T_j \right) \text{ ....(1)}$$

where α and β are the earliness and tardiness penalties per time unit respectively. Three well-known properties [13,14] which are essential for an optimal schedule are as follows:

1. There are no idle times between consecutive jobs.
2. An optimal schedule has the so-called V-shape property, that is, jobs finished before the due date are ordered according to non increasing ratios p_i/α_i and jobs finished after the due date are ordered according to non-decreasing ratios p_i/β_i.
3. There is an optimal schedule in which either the processing time of the first job starts at time zero or one job is finished at the due date.

All potential optimal schedules can be divided into three cases: 1) the first job starts at time zero and the last early job is finished exactly at time d , 2) the first job starts at time zero and the last early job is finished before d , here a straddling job exists and 3) the first job does not necessarily start at time zero.

In this version of the Bees Algorithm, the foraging process of honeybees has been adapted to find an appropriate idle time before processing the first job and a sequence of jobs. According to the third property, the first job in an optimal schedule might not start at time zero. However, the V-shape property [13,14] has been considered in the Bees Algorithm as well.

5 Results

There are seven different data sets with different numbers of jobs (*n*=10, 20, 50, 100, 200, 500, 1000). The processing time, earliness and tardiness penalties are given to each of the jobs. The common due date can be calculated as:

Common due date (d) = *round* [*SUM_P* * *h*]..………….(2)

where *SUM_P* is the sum of processing time and h is the restrictive factor (h =0.2, 0.4, 0.6, 0.8 were used for this benchmark). The value of the restrictive factor h classifies the problems as less or more restricted against a common due date. Each data set contains 10 instances (from $K=1$ to $K=10$). Therefore the problem has 280 instances in total. These instances can be downloaded from the OR-Library website http://people.brunel.ac.uk/~mastjjb/jeb/orlib/schinfo.html.

The performance of the algorithm was quantified by two indices: 1) percentage relative deviations (Δ), 2) standard deviation. To obtain the average performance of the algorithm, 10 runs were carried out for each problem instance to report the statistics based on the percentage of relative deviations (Δ) from the upper bounds in Biskup and Feldmann [13]. To be more specific, Δ_{avg} was computed as follows:

$$\Delta_{avg} = \sum_{i=1}^{R} \left(\frac{(F_{BA} - F_{ref})}{F_{ref}} \times 100 \right) \Big/ R \ldots\ldots\ldots\ldots..(3)$$

where F_{BA}, F_{ref} and R are the fitness function values generated by the BEES ALGORITHM in each run, the reference fitness function value generated by Biskup and Feldmann [13], and the total number of runs, respectively. For convenience, $\Delta_{\min}$, $\Delta_{\max}$ and Δ_{std} denote the minimum, maximum and standard deviation of percentage of relative deviation in fitness function value over R runs, respectively. Table 1 shows the parameter values used for this experiment.

Table 1: The parameters of the Bees Algorithm

Parameters	Value
b : Population	*
m : Number of selected sites	200
e : Number of elite sites	100
ngh : Initial patch size	6
nep : Number of bees around elite points	50
nsp : Number of bees around other selected points	30

* : When the number of jobs n is less than 100, $b = 2n$. Otherwise, $b = 400$.

The results obtained by the Bees Algorithm were compared with the results from [13-17]. Note that in Biskup and Feldmann [13], the average percentage improvements and their standard deviations are given using the best solution among all the heuristics, namely, evolution search (ES), simulated annealing (SA), threshold accepting (TA) and TA with a back step (TAR). Since the Bees Algorithm is stochastic, its minimum, maximum, average and standard deviation of runs should be given to evaluate its performance. However, Hino et al. [15] conducted 10 runs and selected the best out of 10 runs even updating the idle time. For this reason, the minimum percentage of relative deviation

($\Delta_{\min}$) of the Bees Algorithm was compared to Hino et al. [15] and Pan et al. [16]. Note that the best results so far in the literature are reported in bold in all Tables given in this paper.

Table 2: Minimum deviation of the computational results

n	*h* 0.2					
	DPSO	TS	GA	HTG	HGT	Bees
10	**0.00**	0.25	0.12	0.12	0.12	**0.00**
20	-3.84	-3.84	-3.84	-3.84	-3.84	-3.84
50	**-5.70**	**-5.70**	-5.68	**-5.70**	**-5.70**	**-5.70**
100	**-6.19**	**-6.19**	-6.17	**-6.19**	**-6.19**	**-6.19**
200	**-5.78**	-5.76	-5.74	-5.76	-5.76	**-5.78**
500	-6.42	-6.41	-6.41	-6.41	-6.41	**-6.43**
1,000	**-6.76**	-6.73	-6.75	-6.74	-6.74	**-6.76**
Avg.	**-4.96**	-4.91	-4.92	-4.93	-4.93	**-4.96**

n	*h* 0.4					
	DPSO	TS	GA	HTG	HGT	Bees
10	**0.00**	0.24	0.19	0.19	0.19	**0.00**
20	**-1.63**	-1.62	-1.62	-1.62	-1.62	**-1.63**
50	**-4.66**	**-4.66**	-4.60	**-4.66**	**-4.66**	**-4.66**
100	**-4.94**	-4.93	-4.91	-4.93	-4.93	**-4.94**
200	**-3.75**	-3.74	**-3.75**	**-3.75**	**-3.75**	**-3.75**
500	-3.56	-3.57	**-3.58**	**-3.58**	**-3.58**	-3.57
1,000	-4.37	-4.39	**-4.40**	-4.39	-4.39	-4.35
Avg.	**-3.27**	-3.24	-3.24	-3.25	-3.25	**-3.27**

n	*h* 0.6					
	DPSO	TS	GA	HTG	HGT	Bees
10	**0.00**	0.10	0.03	0.03	0.01	**0.00**
20	**-0.72**	-0.71	-0.68	-0.71	-0.71	**-0.72**
50	**-0.34**	-0.32	-0.31	-0.27	-0.31	**-0.34**
100	**-0.15**	-0.01	-0.12	0.08	0.04	**-0.15**
200	**-0.15**	-0.01	-0.13	0.37	0.07	**-0.15**
500	**-0.11**	0.25	**-0.11**	0.73	0.15	**-0.11**
1,000	**-0.06**	1.01	-0.05	1.28	0.42	-0.05
Avg.	**-0.22**	0.04	-0.20	0.22	-0.05	**-0.22**

n	*h* 0.8					
	DPSO	TS	GA	HTG	HGT	Bees
10	0.00	0.00	0.00	0.00	0.00	0.00
20	**-0.41**	**-0.41**	-0.28	**-0.41**	**-0.41**	**-0.41**
50	**-0.24**	**-0.24**	-0.19	-0.23	-0.23	**-0.24**
100	**-0.18**	-0.15	-0.12	-0.08	-0.11	**-0.18**
200	**-0.15**	-0.04	-0.14	0.26	0.07	**-0.15**
500	**-0.11**	0.21	**-0.11**	0.73	0.13	**-0.11**
1,000	**-0.06**	1.13	-0.05	1.28	0.40	-0.05
Avg.	**-0.16**	0.07	-0.13	0.22	-0.02	**-0.16**

* DPSO : Discrete Particle Swarm Optimisation, TS : Tabu Search
GA : Genetic Algorithm, HTG : Tabu Search + Genetic Algorithm
HGT : Genetic Algorithm + Tabu Search, Bees : Bees Algorithm

Table 3: Comparison of maximum deviations

$\Delta_{\max}$		
h	DPSO	Bees
0.2	-4.90	**-4.95**
0.4	-3.18	**-3.26**
0.6	-0.03	**-0.22**
0.8	**-0.16**	**-0.16**
Avg.	-2.07	**-2.15**

Table 4: Comparison between the Bees Algorithm, DPSO and DE

		$\Delta_{\min}$		$\Delta_{\max}$		Δ_{avg}			Δ_{std}	
n	*h*	DPSO	Bees	DPSO	Bees	DPSO	Bees	DE	DPSO	Bees
10	0.2	**0.00**	**0.00**	0.11	**0.00**	0.01	**0.00**	**0.00**	0.03	**0.00**
	0.4	**0.00**	**0.00**	0.15	**0.00**	0.02	**0.00**	**0.00**	0.05	**0.00**
	0.6	**0.00**	**0.00**	0.01	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
	0.8	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
20	0.2	**-3.84**	**-3.84**	-3.79	**-3.83**	-3.83	**-3.84**	**-3.84**	0.02	**0.00**
	0.4	**-1.63**	**-1.63**	-1.57	**-1.63**	-1.62	**-1.63**	**-1.63**	0.02	**0.00**
	0.6	**-0.72**	**-0.72**	-0.66	**-0.72**	-0.71	**-0.72**	**-0.72**	0.03	**0.00**
	0.8	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**-0.41**	**0.00**	**0.00**
50	0.2	**-5.70**	**-5.70**	-5.61	**-5.69**	-5.68	**-5.70**	-5.69	0.03	**0.00**
	0.4	**-4.66**	**-4.66**	-4.52	**-4.66**	-4.63	**-4.66**	**-4.66**	0.05	**0.00**
	0.6	**-0.34**	**-0.34**	-0.23	**-0.34**	-0.31	**-0.34**	-0.32	0.04	**0.00**
	0.8	**-0.24**	**-0.24**	**-0.24**	-0.22	**-0.24**	**-0.24**	**-0.24**	**0.00**	0.01
100	0.2	**-6.19**	**-6.19**	-6.15	**-6.19**	-6.18	**-6.19**	-6.17	0.02	**0.00**
	0.4	**-4.94**	**-4.94**	-4.82	**-4.93**	-4.90	**-4.94**	-4.89	0.04	**0.00**
	0.6	**-0.15**	**-0.15**	0.26	**-0.14**	-0.09	**-0.14**	-0.13	0.14	**0.00**
	0.8	**-0.18**	**-0.18**	**-0.18**	-0.17	**-0.18**	**-0.18**	-0.17	**0.00**	**0.00**
200	0.2	**-5.78**	**-5.78**	-5.74	**-5.77**	-5.77	**-5.78**	-5.77	0.01	**0.00**
	0.4	**-3.75**	**-3.75**	-3.68	**-3.74**	-3.72	**-3.75**	-3.72	0.02	**0.01**
	0.6	**-0.15**	**-0.15**	0.56	**-0.15**	-0.03	**-0.15**	0.23	0.27	**0.00**
	0.8	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**	**-0.15**	0.20	**0.00**	**0.00**
500	0.2	-6.42	**-6.43**	-6.40	**-6.42**	-6.41	**-6.43**	**-6.43**	0.01	**0.00**
	0.4	-3.56	**-3.57**	-3.51	**-3.56**	-3.54	**-3.57**	**-3.57**	0.01	**0.00**
	0.6	**-0.11**	**-0.11**	**-0.11**	**-0.11**	**-0.11**	**-0.11**	1.72	**0.00**	**0.00**
	0.8	**-0.11**	**-0.11**	**-0.11**	**-0.11**	**-0.11**	**-0.11**	1.01	**0.00**	**0.00**
1000	0.2	**-6.76**	**-6.76**	-6.73	**-6.74**	**-6.75**	**-6.75**	-6.72	**0.01**	**0.01**
	0.4	**-4.37**	-4.35	-4.32	**-4.33**	-4.35	-4.34	**-4.38**	**0.01**	**0.01**
	0.6	**-0.06**	-0.05	-0.03	**-0.05**	-0.04	**-0.05**	1.29	0.01	**0.00**
	0.8	**-0.06**	-0.05	**-0.06**	-0.05	**-0.06**	-0.05	2.79	**0.00**	**0.00**
Avg.		**-2.15**	**-2.15**	-2.07	**-2.15**	-2.14	**-2.15**	-1.87	0.03	**0.00**

Table 2 summarises $\Delta_{\min}$ of the computational results to be compared to Hino et al. [15] and Pan et al. [16] with regard to h. As seen in Table 2 ($h = 0.2$ and $h = 0.4$), there is not a large difference, but for $h = 0.6$ and

$h = 0.8$ there is a great deal of difference, especially with the larger size problems (ranging from 100 to 1000 jobs). The Bees Algorithm, discrete particle swarm optimisation (DPSO) and GA have a similar tendency to yield negative percentage relative deviations ($\Delta_{\min}$), which means they outperform Biskup and Feldmann [13]. However, Tabu Search (TS), HTG (TS+GA) and HGT (GA+TS) show a tendency to diverge after 100 jobs and give positive percentage of relative deviations ($\Delta_{\min}$), which means they are inferior to Biskup and Feldmann [13].

Table 3 shows maximum of percentage of relative deviations ($\Delta_{\max}$) between the Bees Algorithm and DPSO with regard to h. When h is 0.2, 0.4 and 0.6, $\Delta_{\max}$ of the Bees Algorithm is superior to the DPSO and the total average is also much better than the DPSO. In particular, the Bees Algorithm is superior to the DPSO, when h is 0.6. It is also interesting to note that, as seen in Pan et al. [16], even the average of maximum percentage of relative deviation ($\Delta_{\max}$) of the Bees Algorithm is much better than $\Delta_{\min}$ of TS, GA, HTG and HGT.

Table 4 shows comparative results for the Bees Algorithm and DPSO in terms of minimum, maximum and average percentage of relative deviations and standard deviations. The average percentage of relative deviation (Δ_{avg}) of the Bees Algorithm was compared to the DPSO [16] and differential evolution (DE) [17]. It was found that the Bees Algorithm outperforms these two algorithms. As seen from the total averages in Table 4, the Bees Algorithm is slightly better than the DPSO at -2.15. For 200, 500 and 1,000 jobs, when h equals 0.6 or 0.8, the Bees Algorithm and DPSO can perform better than the DE. As can be seen, the standard deviation for the Bees Algorithm is nearly zero, which means that the Bees Algorithm is slightly more robust than DPSO. All the statistics obtained show that the performance of Bees Algorithm is superior to all existing approaches.

6 Conclusion

The Bees Algorithm is a new evolutionary optimisation method that has been used in a wide range of applications. However, this paper is the first to report the application of the Bees Algorithm to a combinatorial problem. The computational results show that the Bees Algorithm performed more strongly than the existing techniques. The idle time factor adjustment at each iteration enabled the metaheuristics to achieve better results. Moreover, in terms of standard deviation, the Bees Algorithm also proved more stable and robust than the other algorithms.

References

[1] Garey M R and Johnson D S, *Computers and Intractability: A Guide to Theory of NP-Completeness*, Freeman, San Francisco, 1979

[2] Aarts E and Lenstra J K, *Local Search in Combinatorial Optimization*, John Wiley & Sons Ltd, England, 1997

[3] Pham D T, Ghanbarzadeh A, Koc E, Otri S, Rahim S and Zaidi M, *The Bees Algorithm*, Technical Note, Manufacturing Engineering Centre, Cardiff University, UK, 2005

[4] Dorigo M and Stutzle T, *Ant Colony Optimization*, MIT Press, Cambridge, 2004

[5] Goldberg D E, *Genetic Algorithms in Search, Optimization and Machine Learning*, Reading, Addison-Wesley Longman, 1989

[6] Eberhart R, and Kennedy J, *Swarm Intelligence*, Morgan Kaufmann, San Francisco, 2001

[7] Bilchev G and Parmee I C, *The Ant Colony Metaphor for Searching Continuous Design Spaces*, in Selected Papers from AISB Workshop on Evolutionary Computing, pp. 25-39, 1995

[8] Mathur M, Karale S B, Priye S, Jayaraman V K and Kulkarni B D, *Ant Colony Approach to Continuous Function Optimization*, Ind. Eng. Chem. Res.39(10), pp. 3814-3822, 2000

[9] Von Frisch K, *Bees: Their Vision, Chemical Senses and Language*, (Revised edn) Cornell University Press, N.Y., Ithaca, 1976

[10] Seeley T D, *The Wisdom of the Hive: The Social Physiology of Honey Bee Colonies*, Massachusetts: Harvard University Press, Cambridge, 1996

[11] Bonabeau E, Dorigo M, and Theraulaz G, *Swarm Intelligence: from Natural to Artificial Systems*, Oxford University Press, New York, 1999

[12] Camazine S, Deneubourg J, Franks N R, Sneyd J, Theraula G and Bonabeau E, *Self-Organization in Biological Systems*, Princeton: Princeton University Press, 2003

[13] Biskup D and Feldmann M, *Benchmarks for scheduling on a single machine against restrictive and unrestrictive common due dates*, Computers & Operations Research, volume 28, pp. 787-801, 2001

[14] Feldmann M, and Biskup D, *Single-machine scheduling for minimizing earliness and tardiness penalties by meta-heuristic approaches*, Computers & Industrial Engineering, volume 44, pp. 307-323, 2003

[15] Hino C M, Ronconi D P, Mendes A B, *Minimizing earliness and tardiness penalties in a single-machine problem with a common due date*, European Journal of Operational Research, volume 160, pp. 190-201, 2005

[16] Pan Q K, Tasgetiren M F, Liang Y C, *A Discrete Particle Swarm Optimization Algorithm for Single Machine Total Earliness and Tardiness Problem with a Common Due Date*, IEEE Congress on Evolutionary Computation 2006, Vancouver, BC, Canada, pp. 3281-3288, 2006

[17] Nearchou A C, *A differential evolution approach for the common due date early/tardy job scheduling problem*, Computers & Operations Research, 2006.