An ABC Algorithm Minimizing Regular Objectives for Blocking Permutation Flow Shop Scheduling Problem

Nouha Nouri[#1], Talel Ladhari[#*2]

[#]*Ecole Supérieure des Sciences Economiques et Commerciales de Tunis, University of Tunis, Tunisia*

[1]nouri.nouha@yahoo.fr

[*]*College of Business, Umm Al-Qura University, Umm Al-Qura, Saudi Arabia*

[2]talel_ladhari2004@yahoo.fr

***Abstract*— In this paper, we expose a new meta-heuristic based on the artificial behavior of bee colony (ABC) for solving the blocking permutation flow shop scheduling problem (BFSP) subject to regular objectives. On three main steps, the proposed algorithm iteratively solved the BFSP under three fixed measure (makespan, total flowtime, or total tardiness criterion). Discrete job permutations and operators are used to represent and generate new solutions for the employed, the onlookers, and scout bees, respectively. The performance of this algorithm is tested on the well-known benchmark sets of Taillard, and Ronconi and Henriques. The computational results assert the effectiveness of this heuristic in comparison with some recently proposed state-of-the-art algorithms. So, new best known solutions for the benchmark sets are found for the considered problem.**

***Keywords*— Flow Shop, Blocking, ABC, Makespan, Total Flowtime, Total Tardiness**

I. INTRODUCTION

The traditional permutation flow shop problem plays a primary role in the scheduling theory. It is one of the first machine scheduling models that has been explored since the publication of Johnson's paper for solving the two machine flow shop problem [15]. The issue is to specify a complete sequence of all jobs, the same for each machine that minimizes some criterion. The most studied optimization measure is the minimization of the makespan (C_{max}) which defines the time at which the last job in the sequence is completed and leaves the system. This problem with makespan criterion is denoted as F|prmu|C_{max}; using the $\alpha|\beta|\gamma$ notation of Graham et al. [12]. Various other scheduling features have been studied and analyzed in the literature: the total flow time, the maximum lateness, the maximum tardiness, the maximum earliness, and other similar criteria. All the above objective functions are so-called regular performance measures. A regular performance measure is a function that is non decreasing in $\{C_1,C_2,...,C_N\}$: measure that is always improved by reducing job completion time. Anyway, it is commonly assumed whatever the objective that buffer storage capacity between machines is limitless.

Now, if we imagine no intermediate buffer exists between machines, then the problem becomes a blocking permutation flow shop (BFSP) [32]. Since there are no buffers in the shop, a job has to stay on a machine until its next machine is free. This classical blocking situation is called 'Release when Starting Blocking (RSb)'.

One of the pioneering works on this problem is [26] who showed that the $F_2|blocking|C_{max}$ problem may be reduced to a special case of the traveling salesman problem which may be resolved in polynomial time using the famous Gilmore and Gomory algorithm [9]. For m>2 the problem is proved to be strongly NP-hard [13]. As well, the same problem with total flow time criteria and two machines is strongly NP-hard [29]. The two-machine flow shop tardiness case is also NP-hard [18].

Heuristic algorithms used in the literature may be broadly divided into constructive and improvement heuristics.
Among the proposed constructive heuristics for the BFSP with makespan criterion, we may refer to the Profile Fitting (PF) [20] developed to deal with the assembly line scheduling problem; the far-famed NEH (Nawaz-Enscore-Ham) [21] originally presented for the traditional flow shop problem (made up of two phases: creation of the initial sequence of the jobs and the iterative insertion procedure); the minmax (MM), the combination of MM and NEH (MME), and the combination of PF and NEH (PFE) [30]. For the BFSP with total flow time criterion, we refer to the modified NEH heuristic called NEH-WPT [35] where the superiority of NEH-WPT over NEH is proved. In [32], a constructive heuristic and a GRASP-based heuristic are settled for the BFSP with total tardiness criterion.

Regarding the improvement heuristics, we may refer to the heuristic used to minimize the cycle time in a blocking flow shop [1] which can also be employed to the BFSP under makespan criterion. As well, the Genetic Algorithm (GA) proposed in [5] to solve large size flow shop problems and

ISSN:2356-5608

based on the slowing down restriction idea deals with the blocking problem as a special case. Subsequently, in [31] authors proposed (Ron) algorithm based on blocking-lower bound that out-performed algorithms presented in their earlier studies. In [11], a fast Tabu search (TS) and an improved TS algorithms (TS+M) based on multi-moves technique are exposed. Experimental results demonstrated that both of them are relatively more rapid and effective in finding better solutions than GA [5] and Ron [31]. More recently, the reference [34] proposed a Hybrid Discrete Differential Evolution (HDDE) algorithm for the $F_m|blocking|C_{max}$. Computational results demonstrated that HDDE algorithm not only out performs TS and TS+M algorithms but also gets better results than the Hybrid Differential Evolution (HDE) algorithm [25]. We refer also to the reference [27] where an Iterated Greedy (IG) algorithm is proposed which makes use of the insertion method of the NEH heuristic.

Meanwhile, in [35] a hybrid modified global-best Harmony Search (hmgHS) algorithm is proposed for solving the blocking problem with the total flow time criterion. Similarly, in [36] the hmgHS algorithm is submitted under the same problem and was proven its superiority over the IG [27]. Later, a Discrete Artificial Bee Colony algorithm ($DABC_D$) is proposed [8] and compared with several other powerful algorithms, including DABC proposed in [33] (denoted by $DABC_T$), HDDE, and the IG algorithm. Three other effective hybrid DABC algorithms are proposed in [14].

As well, by combining the respective advantages of Artificial Immune Systems (AIS) and the annealing process of Simulated Annealing (SA), an effective Revised AIS (RAIS) algorithm is proposed minimizing the makespan [19]. The computational results showed that the heuristic provide better performances than the leading approaches for all problem sizes. In [23], a three-phase algorithm for the $F_m|blocking|C_{max}$ is presented: higher efficiencies of the algorithm over the IG algorithm. In [37] a Discrete Particle Swarm Optimization algorithm with self-adaptive diversity control is proposed to solve the same problem. After that, a high performing Memetic Algorithm (MA) for the blocking problem minimizing makespan is appeared [22], where a constructive heuristic is presented (combining PF and NEH) and a memetic algorithm is proposed based on path-relinking based crossover and a referenced local search. At last, in [6], a chaos-induced discrete self organizing migrating algorithm is applied to the blocking problem and tested on the Taillard problem sets.

Unlikely, there is a great work dedicated to developing both exact and heuristic algorithms for both makespan and total flow time criteria; it seems that little work has dealt with the total tardiness criterion until recently.

In particular, using the LBNEH method suggested in [2], Armentano and Ronconi [3] proposed a Tabu Search procedure to obtain an initial solution for their heuristic. In [32], a new NEH-based method called (FPDNEH) and a Greedy Randomized Adaptive Search Procedure (GRASP) were proposed. As well, in reference [28] an efficient Iterated Local Search algorithm (ILS) coupled with a Variable Neighborhood Search (VNS) is exposed to minimize the tardiness measure. A rapid survey on flow shop with blocking and no-wait constraint in process can be found in [13].

In this work, a discrete algorithm based on the performance of foraging artificial bees colony is proposed to solve the BFSP under three regular objectives. We hybridize the algorithm with a local search technique and provide new schemes for the employed, onlookers, and scout bees phases. Computational experiments are done using the two well known Taillard and Ronconi and Henriques benchmark sets.

Thus, the rest of this paper is organized as follows: In Section 2 we briefly introduce the BFSP problem. We describe in Section 3 our blocking algorithm. The experiments results are reported in the next section. Finally, in section 5, we present our concluding remarks.

II. Problem Statement

We consider the blocking problem to minimize separately, the makespan, the total flow time, and the total tardiness in the M-machine permutation flow shop environment. The problem can be defined as follows. At time 0, there are N independent jobs to be sequentially processed on M machines, having no intermediate buffers. All jobs follow the same route in the machines. Since there is no buffer between each successive pair of machines, intermediate queues of jobs waiting in the system for their next operation are not allowed. Therefore, a job cannot leave a machine until the next machine downstream is free. This blocking situation prevents processing of other jobs on the blocked machine. We call down the following assumptions to define the BFSP:

- There is precisely one task corresponding to the processing of job i (i=1,2,...,N) on each machine j (j=1,2,...,M) which requires a processing time p_{ij}, and may have a given due date D_i which is the time point at which the job should finish.
- Each job may be processed on only one machine at any time, and each machine can process only one job at a time.
- Jobs are ready for processing at the beginning and have no precedent constraints among them. Preemption is not allowed.

Considering the above-mentioned assumptions, the objectives is to find out a sequence for processing all jobs on all machines so as to singly minimize the maximum completion time, the total flow time, and the total tardiness.

We consider only schedules in which the jobs are processed in the same order on all machines (permutation schedules). According to the classifications mentioned in Graham et al. [12], the blocking instances considered in this study are as follows: $F_m|block|C_{max}$, $F_m|block|\sum C_j$, and $F_m|block|\sum T_j$ representing respectively the BFSP with makespan, total completion time, and total tardiness criteria.

Let: $\Pi:=(\pi_1, \pi_2,..., \pi_N)$ be a solution for the problem, where π_i denotes the i^{th} job in the considered sequence; $d_{\pi i,j}$ (i=1,2,...,N; j=0,1,2,...,M) represents the departure time of job π_i on machine j, where $d_{\pi i,0}$ denotes the time job π_i starts its

ISSN:2356-5608

processing on the first machine. The corresponding values of makespan of Π may then be calculated in O(nm) as $C_{max}(\Pi)=C_{\pi N,M}\}(\Pi)$, where $C_{\pi i,M}=d_{\pi i,M}$ is the completion time of job π_i on machine M that can be calculated using the following expressions presented in [24]:

$$d_{\pi_i,0}=0 \quad (1)$$

$$d_{\pi_i,j}=\sum_{k=1}^{j} p_{\pi_i,k} \quad i=1,2,..M-1 \quad (2)$$

$$d_{\pi_i,0}=d_{\pi_{i-1},1} \quad i=2,....N \quad (3)$$

$$d_{\pi_i,j}=\max\{d_{\pi_i,j-1}+p_{\pi_i,j}, d_{\pi_{i-1},j+1}\}$$
$$i=2,...,N;\ j=1,2,...M-1 \quad (4)$$

$$d_{\pi_i,M}=d_{\pi_i,M-1}+p_{\pi_i,M}\ i=1,2,...,N \quad (5)$$

Using again the above recursion, we may also calculate the total flow time (TFT) and the total tardiness (TT) as follows: $TFT(\Pi) = \sum^{N}_{i=1}(C_{\pi i,M})$ and $TT(\Pi) = \sum^{N}_{i=1}(T_i)$ where $T_i = \max\{0,(C_{\pi i}-D_i)\}$. We choose to calculate the due dates D_i following the Total Work Content (TWK) rule [4]: $D_i = \tau*(\sum^{M}_{j=1}(p_{\pi i,j}))$. τ is the due date tightness factor and $(\sum_{j=1}(p_{\pi i,j}))$ is the total processing time of job π_i on all machines. τ is taken in the range [1-3] to make the job's due date loose, medium or tight randomly.

III. The Discrete Artificial Bee Colony Algorithm for the BFSP

The ABC algorithm is a new swarm intelligence based approaches [16] originally designed for continuous function optimization. The artificial bees are sorted into three groups: employed, onlooker, and scout bees. The number of onlookers and employed bees corresponds to the number of food sources SN surrounding the hive. At the start, the algorithm generates randomly initial population of solutions, which are then randomly assigned to the employed bees and evaluated. Later, the population iterates the search processes of the employed, onlooker and scout bees. The employed or onlooker bee produces new solutions by modifying probabilistically their current solutions, and tests their fitness value. When all the employed bees end the search process, they share out information with the onlooker bees. The onlooker bee checks these information and picks a food source with a probability related to its nectar quantity. If a solution is not improved after a fixed number of Limit trials, that solution is neglected by its employed bee.

In the following we describe all parameters and operators used in the proposed ABC for the BFSP under some fixed criterion.

A. Population initialization

We initialize the population which consists of a set of food sources. 'x' initial solutions are obtained using the PF-NEH(x) heuristic developed in [23]. So, while jobs are being arranged in an increasing order of the sum of their processing times at the first stage of the PF-NEH(x) algorithm, x solutions are produced iteratively where at each iteration of the algorithm the first ranked job is picked from the initial jobs ordering and then a finite solution is constructed based on both the PF method and the NEH's insertion-stage. Only the last λ jobs in the sequence produced by the PF method undergo the NEH's insertion-stage. The remaining (SN-x) members of the colony are randomly generated.

Meanwhile, to improve the quality of the initial explored food sources, we perform the referenced local search technique as in [23]: it iteratively moves in the neighbor solutions until a local optimum is reached. A speed-up method is used to reduce the computational time needed for searching. The steps in the referenced local search are described in the following:

Procedure Referenced local search(Π,Π^{ref})

Stage 1: While (Π is improved)\

For i=1 ***To*** N ***Do***

- Let Π'= Π
- Find the job π^{ref}_i in Π', and remove it.
- Test π^{ref}_i in all the possible slots of Π'.
- Insert π^{ref}_i in the slot resulting the lowest objective value.
- If O(Π')<O(Π) Then Π= Π'.

End For

Stage 2: Return the sequence Π

End Referenced local search

B. Employed bee phase

Now, having assigned randomly initial solutions to PS employed bees, candidate solutions are to be explored based on the path relinking approach [10]. In this paper, the path relinking starts from an incumbent solution to be enhanced and stops when there is no other possible movements to reach the target solution. A solution is initial if Φ>=0, else it is a target. Besides, if no intermediate solutions exist then the current solution is kept and so not improved. A numerical example is explained in Fig. 1 where Π_s is the origin, Π_d the destination solution, and Π_i are the intermediate solutions.

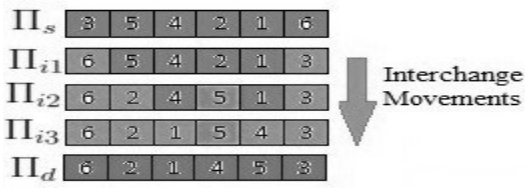

Fig. 1 An example using the path relinking technique

A When a new solution is found for an employed bee, and its fitness value is better refined than that of the incumbent solution, then the employed bee memorizes the new position and reset its trial counter; else the parameter $trial_i$ of the incumbent solution is incremented by one.

C. Onlooker bee phase

ISSN:2356-5608

To improve once more the quality of the colony, each onlooker bee selects a solution of an employed bee and tries to modify it. This selection is performed using the roulette wheel selection operator. Besides, the referenced local search based on the speed-up method is used again to explore neighboring solutions for each onlooker bee. The referenced sequence is randomly chosen. If the new explored solution is better, then the incumbent one is erased and the new explored solution by the local search technique becomes a new individual in the colony. Else, its trial is incremented by one.

D. Scout bee phase

The scout bee phase is evoked to replace if exists the discarded solution by the employed or onlooker bee during the last $Limit$ number of trials, and generates new one by applying the insertion-based local search [8] to the best solution in the colony. This technique receives a permutation as input and then tries to explore all possible permutations those obtained by inserting jobs in different positions from their original places. There is exactly one scout bee that produces at most one solution at iteration of the search process. After completing the three main phases of the ABC algorithm, the population is cleaned by identifying redundant solutions that will be replaced with randomly generated solutions.

E. Final BABC algorithm

The complete Blocking ABC algorithm (BABC) runs in three basic steps: Initialization, position updating and termination, and has the following scheme:

Algorithm BABC

- ***Stage 1:*** Initialization:
  1) *Set* the parameters: SN, PS, Limit, and MCN.
  2) *Generate* the initial population: apply the PF-NEH(x) heuristic to produce 'x' initial solutions, the rest are generated randomly. *Set* $trial_i = 0$, i:1,2,...,SN
  3) *Test* the population: calculate the objective function value O(Π) for each solution (fitness), then store the best-solution.
- ***Stage 2:*** Iterative process
  cycle=1; ***Repeat***
  Employed Bees phase
  For i = 1,2,. . . ,PS Repeat
  1) *Produce* a new solution Π^*_i for the i^{th} employed bee who is associated with the solution Π_i using the path relinking procedure;
  2) *Evaluate* the new solution Π^*_i: If Π^*_i is better than or equal to Π_i, then *set* $\Pi_i = \Pi^*_i$;
  3) If Π_i has not been improved, then *Increment* its trial by 1

  Onlooker Bees phase
  For i = 1,2,. . . ,PS Repeat
  1) *Select* a solution Π_i for the i^{th} onlooker bee using the roulette wheel selection;
  2) *Generate* a new solution Π^*_i for the i^{th} onlooker bee using the referenced local search;
  3) *Evaluate* the new solution Π^*_i: If Π^*_i is better than or equal to Π_i, then *Set* $\Pi_i = \Pi^*_i$;
  4) If Π_i has not been improved, then *Increment* its trial by 1

  Scout bee phase
  1) *Pick* the abandoned solution, if exists, during the last limit number of trails: *Define* i as the solution Π_i with the maximum trial;
  2) *Replace* Π_i based on the best solution on the population by applying the insertion-based local search, and *Set* its trial to 0;

  Overlapping solutions
  1) *Search* for redundant solutions, and replace them based on randomly generated one;
- ***Memorize*** the best solution achieved so far;
  Cycle = Cycle + 1; ***Until*** cycle = MCN
- ***Return*** the best solution found so far;

IV. Computational Results

To test the performance of the proposed algorithm, comprehensive experimental evaluations are presented based on the well-known flow shop benchmark set of Taillard. The benchmark set is composed of 12 subsets of given problems with the size ranging from 20 jobs and 5 machines to 500 jobs and 20 machines, and each subset consists of 10 instances. We treat them as the blocking flow shop scheduling problems with makespan and total flow time criterion. To deal with the total tardiness criterion, we consider test problems generated by Ronconi and Henriques [32]. There are five groups of job sizes: 20, 50, 100, 200, and 500. In each job subset (20,50,100), they have 5, 10, and 20 machines to process the jobs, respectively. The job subset (200) has 10 and 20 machines, and the job subset (500) has only 20 machines to process the jobs. Ten different matrices of processing times were generated for each of the 12 sizes. For each of those matrices, four scenarios were built. The scenarios represent different configurations for the due dates [32]. Each instance is independently executed ten replications, and in each replication the relative percentage deviation (RPD) is computed as follows:

$$RPD(A) = \frac{(Sol^A - Sol^{Min}) * 100}{Sol^{Min}} \quad (6)$$

Where, Sol^A defines the value of the objective function reached by the BABC algorithm; and Sol^{Min} defines the minimum objective value obtained among all the compared algorithms. We can see that the smaller the RPD is the better result the algorithm produces. The BABC algorithm is coded in Visual C++ and run on an Intel Pentium IV 2.4 GHz PC with 512 MB of memory.

On the basis of a set of preliminary experiments, best results were achieved using the following parameters:

ISSN:2356-5608

TABLE I
PARAMETER VALUES USED FOR THE BLOCKING ALGORITHMS AFTER CALIBRATION

Parameter	MCN	PS	SN	Limit
Value	100	50	50	5

A. *Comparison of Taillard's benchmarks*

Table II summarizes the results of the comparison of the algorithms under the makespan criterion. The effectiveness of the reported method against the [22], DPSO [37], DABC [8], IG [27], HDDE [34], and the RAIS [19] was measured by listing the ARPDs values, where Sol^{Min} defines the best-known solution provided by [27]. Note that in the DABC paper [8], the author did not report the results on the larger instances.

TABLE II
ARPD ON TAILLARD INSTANCES FOR EACH ALGORITHM UNDER THE MAKESPAN CRITERION

Inst	RAIS	MA	IG	DPSO	HDDE	DABC	BABC
20*5	0,000%	0,000%	0,000%	0,000%	0,000%	0,000%	0,000%
20*10	0,000%	0,000%	0,000%	0,011%	0,000%	0,000%	0,000%
20*20	0,000%	0,000%	0,000%	0,000%	0,000%	0,000%	0,000%
50*5	-0,192%	-0,083%	0,000%	1,075%	0,779%	-0,202%	-0,226%
50*10	-0,197%	-0,202%	0,000%	1,255%	0,532%	-0,335%	-0,329%
50*20	-0,004%	-0,220%	0,000%	1,294%	0,351%	-0,278%	-0,211%
100*5	-0,818%	-0,360%	0,000%	1,348%	1,778%	-0,051%	-0,842%
100*10	-0,882%	-0,699%	0,000%	1,522%	1,013%	-0,877%	-0,930%
100*20	-0,712%	-0,709%	0,000%	1,843%	0,858%	-1,142%	-0,883%
200*10	-0,329%	-0,565%	0,000%	2,942%	2,494%	0,000%	-0,595%
200*20	-0,629%	-1,488%	0,000%	1,830%	1,240%	0,000%	-1,136%
500*20	-0,015%	-2,388%	0,000%	2,726%	1,538%	0,000%	-2,694%
Average	-0,315%	-0,559%	0,000%	1,321%	0,882%	-0,240%	-0,654%

The above table reveals that MA, DPSO, DABC, IG, HDDE, and RAIS are significantly outperformed by our algorithm for middle or large-scale instance sizes. For the small-scale instance sizes, the proposed algorithm obtains similar results with the compared ones. The total ARPD for all the 120 instances concerning the BABC algorithm is -0,654%. Whereas, for MA, DPSO, DABC, IG, HDDE, and RAIS the values are -0,559%, 1,321%, -0,240%, 0,000%, 0,882%, and -0,315%, respectively. DPSO is the worst algorithm with the largest overall RPD value. The negative average RPD values indicate an enhancement to the best-known solution reported in the literature: these improvements occur in all Taillard's instances from (50×5) to (500×20) test sets.

Besides, Table III summarizes the results of the comparison of the algorithms under the total flowtime criterion. The effectiveness of the reported method against the hmghs [35], hDABC [14], and the IG [17] was measured by listing again the ARPDs values, where Sol^{Min} defines the best-known solution reported by [35].

TABLE III
ARPD ON TAILLARD INSTANCES FOR EACH ALGORITHM UNDER THE TOTAL FLOWTIME CRITERION

Inst	hmghs	IG	Hdabc	BABC
20*5	0,00%	0,00%	0,00%	0,00%
20*10	0,00%	0,00%	0,00%	0,00%
20*20	0,00%	0,00%	0,01%	-0,13%
50*5	0,00%	-0,99%	-0,20%	-1,58%
50*10	0,00%	-0,59%	-0,26%	-1,19%
50*20	0,00%	-0,27%	-0,11%	-1,22%
100*5	0,00%	-2,97%	-0,69%	-4,26%
100*10	0,00%	-1,81%	-0,30%	-4,06%
100*20	0,00%	-1,46%	-0,44%	-3,54%
200*10	0,00%	-2,50%	-0,79%	-4,33%
200*20	0,00%	-1,48%	-0,62%	-4,13%
500*20	0,00%	-1,98%	-1,26%	-2,84%
Average	0,00%	-1,17%	-0,39%	-2,27%

As we can see, the above table reveals that hmgHS, hDABC, and IG are significantly outperformed by the proposed algorithm. The total ARPD for all the 120 instances concerning the BABC is -2,274%. Whereas, for hmgHS, hDABC, and IG the values are: 0,000%, -0,389%, and -1,171%, respectively.

B. *Comparison of Ronconi and Henriques's benchmarks*

Table IV summarizes the results of the comparison of the algorithm under the total tardiness criterion. The effectiveness of the reported method against the GRASP metaheuristic [32] and the GA based on the path relinking technique GA_PR [7] was measured by listing again the ARPDs values, where Sol^{Min} defines the total tardiness value obtained by GRASP algorithm.

Based on the above table, the BABC algorithm obtained the same results as the GRASP in 53 problems. These performances are much better than those obtained by GA_{PR} (124 problems). We report the average of improvement percentage of each class with 10 problems. The BABC achieved the highest improvement for test instances with N*M =200×10 (-22,833%) and N*M =500×20 (-29,064%). The lowest values of improvement of the algorithms were for problems with N*M =20×20 (-1,144%). An overall average improvement of -9,563% was achieved.

These conclusions clearly show that the proposed algorithm provide better results than the compared algorithm for all problem sizes.

TABLE IV
ARPD ON RONCONI AND HENRIQUE'S INSTANCES FOR EACH ALGORITHM UNDER THE TOTAL FLOWTIME CRITERION

ISSN:2356-5608

Inst	Scenario 1		Scenario 2		Scenario 3		Scenario 4		All Scenario	
	GA_PR	BABC	GA_PR	BABC	GA_PR	BABC	GA_PR	BABC	GA_PR	BABC
20*5	-3,774\%	-6,293\%	-1,166\%	-1,781\%	-1,969\%	-2,688\%	-1,625\%	-2,179\%	-2,134\%	-3,236\%
20*10	-1,175\%	-2,257\%	-2,404\%	-1,92\%	-1,436\%	-1,871\%	-1,347\%	-1,843\%	-1,590\%	-2,294\%
20*20	-1,576\%	-1,979\%	-0,544\%	-0,884\%	-0,486\%	-0,652\%	-0,74\%	-0,994\%	-0,837\%	-1,127\%
50*5	-9,437\%	-13,291\%	0\%	-2,09\%	-4,014\%	-5,045\%	-3,138\%	-3,962\%	-4,147\%	-6,097\%
50*10	-8,372\%	-13,133\%	-10,517\%	-16,966\%	-3,169\%	-4,507\%	-1,701\%	-4,194\%	-5,940\%	-9,700\%
50*20	-2,958\%	-7,003\%	-2,347\%	-4,992\%	-2,128\%	-3,151\%	-1,531\%	-2,414\%	-2,241\%	-4,390\%
100*5	-11,033\%	-15,113\%	-4,449\%	-4,776\%	-0,573\%	-2,075\%	-0,628\%	-4,409\%	-4,171\%	-6,593\%
100*10	-6,606\%	-10,407\%	25,378\%	6,854\%	-2,268\%	-3,349\%	-3,45\%	-4,738\%	3,263\%	-2,910\%
100*20	-4,477\%	-11,632\%	-11,545\%	-25,299\%	-1,144\%	-3,135\%	-0,727\%	-2,572\%	-4,548\%	-10,660\%
200*10	-5,616\%	-15,284\%	-57,059\%	-58,357\%	-1,154\%	-7,768\%	-1,516\%	-9,914\%	-16,336\%	-22,831\%
200*20	-4,964\%	-15,918\%	-10,747\%	-29,556\%	-2,383\%	-8,29\%	-2,034\%	-7,308\%	-5,032\%	-15,268\%
500*20	-6,255\%	-20,189\%	-30,471\%	-54,02\%	-2,46\%	-16,132\%	-3,594\%	-25,91\%	-10,695\%	-29,063\%
Average	-5,52\%	-11,042\%	-8,823\%	-16,255\%	-1,957\%	-4,889\%	-1,836\%	-5,870\%	-4,534\%	-9,514\%

V. Conclusion

In this paper we have proposed an ABC algorithm (BABC) for the blocking permutation flow shop scheduling problem under regular objectives. Hybridized with local search technique, we sketch new schemes for the employed, onlooker, and scout bee phases. Computational results attest that BABC algorithm is very competitive when compared with leading algorithms. Improvements occur in all Taillard's instances from (50×5) to (500×20) test sets under makespan and total flow time criteria. An overall average improvement of -9,563% was achieved for the blocking problem under tardiness criterion.

References

[1] I.N.K. Abadi, N.G. Hall, and C. Sriskandarajah, Minimizing cycle time in a blocking flowshop, Operations Research 1 (2000), 177–80.

[2] V.A. Armentano and D.P. Ronconi, Tabu search for total tardiness minimization in flowshop scheduling problems, Computers and Operations Research 26 3 (1999), 219–235.

[3] V.A Armentano and D.P. Ronconi, Minimizac¸ ˆao do tempo total de atrasono problema de flowshop com buffer zero atrav´es de busca tabu, Gestao and Produc¸ao 7 3 (2000), 352–363.

[4] K.R. Baker and J.W.M. Bertrand, An investigation of due date assignment rules with constrained tightness, J. Oper. Manage 3 (1981), 109–120.

[5] V. Caraffa, S. Ianes, TP. Bagchi, and C. Sriskandarajah, Minimizing makespan in a flowshop using genetic algorithms, International Journal of Production Economics 2 (2001), 101–15.

[6] D. Davendra, M. Bialic-Davendra, R. Senkerik, and M. Pluhacek, Scheduling the flow shop with blocking problem with the chaos-induced discrete self organising migrating algorithm, Proceedings 27th European Conference on Modelling and Simulation (2013).

[7] Tiago de O. Januario, Jos Elias C. Arroyo, and Mayron Csar O. Moreira, Nature inspired cooperative strategies for optimization (nicso 2008), Springer Berlin Heidelberg, 2009.

[8] G. Deng, Z. XU, and X. Gu, A discrete artificial bee colony algorithm for minimizing the total flow time in the blocking flow shop scheduling, Chinese Journal of Chemical Engineering 20 (2012), 1067–1073.

[9] P.C. Gilmore and R.E. Gomory, Sequencing a one state variable machine: a solvable case of the traveling salesman problem, Operations Research 5 (1964), 655–79.

[10] F. Glover and M. Laguna, Tabu search, Kluwer Academic Publishers, Boston (1997).

[11] J. Grabowski. and J. Pempera, The permutation flowshop problem with blocking. a tabu search approach, Omega 3 (2007), 302–11.

[12] R.L. Graham, E.L. Lawler, J.K. Lenstra, and Kan.A.H.G. Rinnooy, Optimization and approximation in deterministic sequencing and scheduling: a survey, Annals of Discrete Mathematics 5 (1979), 287–362.

[13] N.G. Hall and C. Sriskandarajah, A survey of machine scheduling problems with blocking and no-wait in process, Operations Research 44 (1996), 510–25.

[14] Yu-Yan Han, J.J. Liang, Quan-Ke Pan, Jun-Qing Li, Hong-Yan Sang, and N.N. Cao, Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem, Int J Adv Manuf Technol 67 (2013), 397–414.

[15] S.M. Johnson, Optimal two- and three-stage production schedules with setup time included, Naval Research Logistics Quarterly 1 (1954), 61–8.

[16] D. Karaboga, An idea based on honeybee swarm for numerical optimization, Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Department (2005).

[17] D. Khorasanian and G. Moslehi, An iterated greedy algorithm for solving the blocking flow shop scheduling problem with total flow time criteria, International Journal of Industrial Engineering and Production Research 23 (2012), 301–308.

[18] C. Koulamas, The total tardiness problem: Review and extensions, Operations Research 42 (1994), 1025–1041.

[19] S.W. Lin and K.C. Ying, Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm, Omega 41 (2013), 383–389.

[20] S.T. McCormick, M.L. Pinedo, S. Shenker, and B. Wolf, Sequencing in an assembly line with blocking to minimize cycle time, Operations Research 37 (1989), 925–935.

[21] M. Nawaz, Jr.E.E. Enscore, and I. Ham, A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem, Omega 11 (1983), 91–95.

[22] Q. Pan, L. Wang, H. Sang, J. Li, and M. Liu, A high performing memetic algorithm for the flowshop scheduling problem with blocking, IEEE Transactions on Automation Science and Engineering 10 (2013), 741–756.

[23] Q.K. Pan and L. Wang, Effective heuristics for the blocking flowshop scheduling problem with makespan minimization, Omega 2 (2012), 218–29.

[24] M. Pinedo, Scheduling: theory, algorithms, and systems, Pretice Hall. U.A.S (2008).

[25] B. Qian, L. Wang, D.X. Huang, W.L. Wang, and X. Wang, An effective hybrid de-based algorithm for multi-objective flowshop scheduling with limited buffers, Computers and Operations Research 1 (2009), 209–3.

[26] S.S. Reddi and C.V. Ramamoorthy, On the flow-shop sequencing problem with no wait in process, Operational Research Quarterly 3 (1972), 323–31.

[27] I. Ribas, R. Companys, and X. Tort-Martorell, An iterated greedy algorithm for the flowhsop scheduling problem with blocking, Omega 3 (2011), 293–301.

[28] Imma Ribas, Ramon Companys, and Xavier Tort-Martorell, An efficient iterated local search algorithm for the total tardiness blocking flow shop problem, International Journal of Production Research 51 (2013), 5238–5252.

[29] H. Rock, Some new results in flow shop scheduling, Zeitschrift fur Operations Research 28 (1984), 1–16.

[30] D.P. Ronconi, A note on constructive heuristics for the flowshop problem with blocking, International Journal of Production Economics 87 (2004), 39–48.

[31] D.P Ronconi, A branch-and-bound algorithm to minimize the makespan in a flowshop problem with blocking, Annals of Operations Research 1 (2005), 53–65.

ISSN:2356-5608

[32] D.P. Ronconi and L.R.S. Henriques, Some heuristic algorithms for total tardiness minimization in a flowshop with blocking, Omega 2 (2009), 272–81.

[33] M.F. Tasgetiren, Q.K. Pan, P.N. Suganthan, and A.H.L. Chen, A discrete artificial bee colony algorithm for the total flow time minimization in permutation flow shops, Inform. Sci. 16 (2011), 3459–3475.

[34] L. Wang, Q.K. Pan, P.N. Suganthan, W.H. Wang, and Y.M. Wang, A novel hybrid discrete differential evolution algorithm for blocking flowshop scheduling problems, Computers and Operational Research 3 (2010), 509–20.

[35] L. Wang, Q.K. Pan, and M.F. Tasgetiren, Minimizing the total flow time in a flowshop with blocking by using hybrid harmony search algorithms, Expert Syst. Appl 12 (2010), 7929–7936.

[36] L. Wang., Q.K. Pan, and M.F. Tasgetiren, A hybrid harmony search algorithm for the blocking permutation flowshop scheduling problem, Comput. Ind. Eng. 1 (2011), 76–83.

[37] X. Wang and L. Tang, A discrete particle swarm optimization algorithm with self-adaptive diversity control for the permutation flowshop problem with blocking, Applied Soft Computing 12 (2012), 652–662.

ISSN:2356-5608