An Efficient Discrete Artificial Bee Colony Algorithm for the Blocking Flow Shop Problem with Total Flowtime Minimization

Imma Ribas[1,a], Ramon Companys[b], Xavier Tort-Martorell[c]

a, Departament d'Organització d'Empreses, DOE – ETSEIB - Universitat Politècnica de Catalunya. BarcelonaTech, Avda. Diagonal,647, 7th Floor, 08028 Barcelona, Spain. E-mail:imma.ribas@upc.edu

b, CDE - EPSEB - Universitat Politècnica de Catalunya. BarcelonaTech, Gregorio Marañón 44-50, 3rd Floor, 08028 Barcelona, Spain. E-mail:ramón.companys@upc.edu

c, Departament de Estadística e investigación Operativa- ETSEIB - Universitat Politècnica de Catalunya. BarcelonaTech, Avda. Diagonal,647, 6th Floor, 08028 Barcelona, Spain. E-mail: xavier.tort@upc.edu

Abstract

This paper presents a high performing Discrete Artificial Bee Colony algorithm for the blocking flow shop problem with flow time criterion. To develop the proposed algorithm, we considered four strategies for the food source phase and two strategies for each of the three remaining phases (employed bees, onlookers and scouts). One of the strategies tested in the food source phase and one implemented in the employed bees phase are new. Both have been proved to be very effective for the problem at hand. The initialization scheme named HPF2(λ,μ) in particular, which is used to construct the initial food sources, is shown in the computational evaluation to be one of the main procedures that allow the DABC_RCT to obtain good solutions for this problem. To find the best configuration of the algorithm, we used Design of Experiments (DOE). This technique has been used extensively in the literature to calibrate the parameters of the algorithms but not to select its configuration. Comparing it with other algorithms proposed for this problem in the literature demonstrates the effectiveness and superiority of the DABC_RCT.

Keywords: Scheduling; flow shop; blocking; total flowtime; heuristics

[1] Corresponding author.
E-mail address: imma.ribas@upc.edu
Fax: +34 93 401 60 54

1 Introduction

The blocking flow shop scheduling problem allows many productive systems to be modeled when there are no buffers between consecutive machines. Some industrial examples can be found in the production of concrete blocks, where storage is not allowed in some stages of the manufacturing process (Grabowski & Pempera, 2000); in the iron and steel industry (Gong, Tang, & Duin, 2010); in the treatment of industrial waste and the manufacture of metallic parts (Martinez, Dauzère-Pérès, Guéret, Mati, & Sauer, 2006); or in a robotic cell, where a job may block a machine while waiting for the robot to pick it up and move it to the next stage (Sethi, Sriskandarajah, Sorger, Blazewicz, & Kubiak, 1992). In general, it is useful for those systems that have a production line without a drag system that forces a job to be transferred between two consecutive stations at pre-established times. In this type of production configuration, a machine can be blocked by the job it has processed if the next machine is not available. Hence, accurate scheduling is necessary to minimize machine blocking and idle time, which allows increasing the productivity level.

Although the blocking flow shop scheduling problem has not been as extensively studied as the permutation flow shop problem, several types of metaheuristics have been proposed to solve the former in order to minimize makespan: a genetic algorithm (GA) (Caraffa, Ianes, Bagchi, & Sriskandarajah, 2001); two tabu search (TS) algorithms (Grabowski & Pempera, 2007); a hybrid genetic algorithm (HGA) (Wang et al., 2006); a particle swarm optimization algorithm (HPSO) (Liu et al., 2008); a differential evolution (DE) algorithm (Qian et al., 2009); a hybrid discrete differential evolution algorithm (Wang et al., 2010); a hybrid harmony search (Wang, Pan, & Tasgetiren, 2011); an iterated greedy algorithm (Ribas, Companys, & Tort-Martorell, 2011); a simulated annealing algorithm with a local search (Wang, Song, Gupta, & Wu, 2012); a discrete self-organizing migrating algorithm (Davendra & Bialic-Davendra, 2013); a variable neighborhood search (Ribas, Companys, & Tort-Martorell, 2013); a Memetic algorithm (Pan, Wang, Sang, Li, & Liu, 2013); an artificial immune system (Lin & Ying, 2013); and a discrete artificial bee colony (Han, Gong, & Sun, 2014).

However, little research has been done to solve the blocking flow shop scheduling problem in ways that include other interesting criteria for the industry, such as total tardiness or total flowtime. For the former, Armentano and Ronconi (2000) proposed a Tabu Search procedure, Ronconi and Henriques (2009) a new NEH-based method and a GRASP algorithm and Ribas, Companys and Tort-Martorell (2013) proposed an iterated

local search method. For the latter, Wang, Pan, and Fatih Tasgetiren (2010) proposed a hybrid Harmony Search (HS) algorithm, Deng, Xu, and Gu (2012) a Discrete Artificial Bee Colony (DABC) algorithm, and Moslehi and Khorasanian (2013) a branch and bound algorithm that can be used in small instances. The criterion of minimizing total flowtime has been found to be an important real-life objective in industries, since it results in the even utilization of resources, even turn-over of finished jobs and reduced in-process inventory. Thus, it is considered to be more relevant and meaningful in today's dynamic production environment (Liu & Reeves, 2001). Therefore, it is interesting to expand on the existing research in order to have efficient scheduling procedures available for sequencing jobs in productive environments that can be modeled as the blocking flow shop problem with total flowtime criterion.

One of the recent swarm metaheuristics that has successfully been applied to several optimization problems is the Artificial Bee Colony (ABC) algorithm proposed by Karaboga (2005). Although the ABC algorithm was described for solving numerical problems, discrete versions have been introduced to solve several combinatorial problems. A complete review of papers published up to 2012 about the ABC algorithm and its applications can be found in (Karaboga, Gorkemli, Ozturk, & Karaboga, 2014). In particular, some Discrete Artificial Bee Colony (DABC) algorithms have been proposed in the field of scheduling to solve several scheduling problems under different constraints and/or objective functions. Nasiri (2015) presents a DABC algorithm for the stage shop problem, which is a special case of the general shop scheduling problem. Pan, Wang, Li, and Duan (2014) present it for the hybrid flow shop scheduling problem to minimize the makespan and Li and Pan (2014) for the hybrid flow shop scheduling problem with limited buffers. Wang, Zhou, Xu, Wang, and Liu (2012) applied a DABC algorithm to the flexible job-shop scheduling problem; Zhang, Song, and Wu (2013) to the job-shop scheduling problem for minimizing the total weighted tardiness; Li, Pan, and Gao (2011) and Wang, Zhou, Xu, Wang, and Liu (2011) proposed a multi-objective DABC algorithm for the flexible job-shop scheduling problem; and Lei (2012) proposed it for the interval job-shop scheduling problem with non-resumable jobs and flexible maintenance. Finally, for the permutation flow shop scheduling problem: Liu and Liu (2013) present a DABC procedure for makespan minimization; and Tasgetiren, Pan, Suganthan, and Chen (2011a) for flowtime minimization. Deng et al. (2012) and Han et al. (2012) considered the blocking constraint and the total flowtime criterion, whereas

Tasgetiren, Pan, Suganthan, and Oner (2013) considered the no-idle constraint for total tardiness minimization.

The blocking flow shop problem, denoted as $Fm\,|\,block\,|\,\sum C_i$, according to the notation proposed by Graham et al. (1979), can be defined as follows. A set of n jobs have to be processed by m machines in the same order, implying that a job sequence determined for machine 1 is kept throughout the system. Each job i, $i\epsilon\{1, 2, ..., n\}$ requires a fixed positive processing time $p_{j,i}$ on every machine j, $j\epsilon\{1, 2, ..., m\}$. Jobs and machines are available from time zero onwards. Our objective is to find a job processing sequence that minimizes the total flowtime. $Fm|block|\Sigma C_i$ can be modeled with the following equations, where $[k]$ is the index of the job in the k-th position in the permutation, $e_{j,k}$ denotes the time at which job $[k]$ begins to be processed by machine j, and $c_{j,k}$ is the departure time of job $[k]$ from machine j. Note that if job $[k]$ can leave machine j when it is completed, which depends on the availability of machine j+1, then $c_{j,k}$ is not only the departure time but also the completion time of job $[k]$ on machine j:

$$e_{j,k} + p_{j,[k]} \leq c_{j,k} \quad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (1)$$

$$e_{j,k} \geq c_{j,k-1} \qquad j=1, 2, ..., m \quad k=1, 2 ,..., n \qquad (2)$$

$$e_{j,k} \geq c_{j-1,k} \qquad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (3)$$

$$c_{j,k} \geq c_{j+1,k-1} \qquad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (4)$$

$$TF = \sum_{k=1}^{n} c_{m,k} \qquad (5)$$

with $c_{j,0} = 0 \quad \forall j \quad , \; c_{0,k} = 0, \quad c_{m+1,k} = 0 \quad \forall k$ being the initial conditions.

If equations (2) and (3) are summarized as (6) and equation (1) and (4) as (7), the schedule obtained is semi-active, which is interesting because an optimal solution can be found in the subset of the semi-active set of solutions.

$$e_{j,k} = \max\{c_{j,k-1}; c_{j-1,k}\} \qquad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (6)$$

$$c_{j,k} = \max\{e_{j,k} + p_{j,[k]}, c_{j+1,k-1}\} \qquad j=1, 2, ..., m \quad k=1, 2, ..., n \qquad (7)$$

The aim of this paper is to propose an efficient DABC algorithm, named DABC_RCT, for the blocking flow shop problem with total flowtime criterion. To develop the proposed algorithm, we considered four strategies for the food source phase and two strategies for each of the three remaining phases (employed bees, onlookers and scouts). One of the strategies tested in the food source phase and one implemented in the

employed bees phase are new. Both have been proved to be very effective. The initialization scheme named HPF2(λ,μ) in particular was used to construct the initial food sources, which the computational evaluation has shown to be one of the main procedures that allow the DABC_RCT to obtain good solutions for this problem. To find the best configuration of the algorithm, we used Design of Experiments (DOE). This technique has been extensively used in the literature to calibrate the parameters of the algorithms but not to select its configuration. Comparing it with other algorithms proposed in the literature for this problem demonstrates the effectiveness and superiority of the DABC_RCT.

The rest of the paper is organized as follows. Section 2 describes the different strategies tested in each phase of the algorithm; section 3 shows the design of experiments done to choose the best combination of strategies; section 4 shows the computational evaluation of the algorithms; and, finally, section 5 is devoted to conclusions and future work.

2 Proposed alternatives for a Discrete Artificial Bee Colony Algorithm

The ABC algorithm is a swarm intelligence technique inspired by the intelligent foraging behavior of honey bees. This algorithm has three essential components: food sources, which are the set of current solutions; the employed bees that are associated with a particular food source to be exploited; and unemployed bees. The unemployed bees are made up of two types: onlookers, who wait in the nest and establish a food source through the information shared by the employed; and scouts, who search for new food sources in the area surrounding the hive. There are several strategies to implement in each part of the algorithm, and each combination can lead to a different Discrete Artificial Bee Colony algorithm. The point is to know which strategy and which combination among them has to be used in order to enhance the performance of the algorithm for the problem at hand. The final configuration of the algorithm was set by means of a design of experiments, which are explained in section 4.

In the first phase (generation of food sources), we implemented four strategies in order to guarantee a diversification of solutions by testing the convenience of starting the algorithm with either good solutions or random solutions. On the remaining steps for employees, onlooker and scout bees (i.e., the components that allow the algorithm to intensify or to diversify the search of solutions), two alternative strategies were also tested. All these methods are explained in the following sections.

2.1 Initialization

The algorithm starts with the generation of N initial solutions. These solutions characterize the initial food sources that will be explored by the employed bees. Each food source is represented as a job permutation, and the total flowtime evaluation of this sequence gives the quality of the source. Some authors (Han et al., 2012; Wu, Qian, Ni, & Fan, 2012; Karaboga & Ozturk, 2011) propose random generation of the food sources (solutions) to guarantee diversification of solutions. Some others propose generating at least one of the solutions by a heuristic procedure in order to obtain one food source of a certain quality (Tasgetiren, Pan, Suganthan, & Chen, 2011b). However, in Liu and Liu (2013), a GRASP based on an NEH algorithm (Nawaz, Enscore, & Ham, 1983) is used to generate all food sources in order to guarantee an initial swarm of quality and diversity.

To investigate whether or not it is better to initialize the algorithm with good solutions, we divided this phase into two parts. The first part generates the set of food sources according to two schemes. One of them provides better solutions than the other. The second part is devoted to analyzing whether or not it is useful to improve these solutions with a variable neighborhood search. The application of the VNS allows improving the solutions at the expense of losing diversity. The final configuration of the two parts will permit knowing the right balance between good solutions and diversity.

2.1.1 The First Part of Food Source Generation

In the first part, two strategies were tested to evaluate whether it is better to start the algorithm with a set of good solutions or by generating one good solution and the others randomly in order to guarantee an initial diversified swarm. Both strategies use a constructive procedure to create a solution, which we named HPF2(λ,μ); but they differ in their generation of the remaining food sources, as will be explained later. HPF2(λ,μ) is a constructive procedure that creates a sequence in two steps: selecting the first job (step 1); and constructing the remaining sequences in order to minimize both the timeout of machines and the total flowtime (step 2).

The first step selects a job that minimizes a bicriteria index (R(i)), which considers its contribution to the completion time (minimum sum of its processing times, P_i) and the generated front delay.

The measurement of the front delay (in grey, Figure 1) can be calculated according to equation (8).

(Please, insert near here figure 1)

Figure 1. Grey area indicates the front delay of job J1

$$\sum_{j=1}^{m}(m-j)\cdot p_{j,i} \tag{8}$$

Since this term had a different magnitude than the sum of the processing time of a job when evaluating index R(i) (see equation (9)), this first term was scaled by multiplying it by 2/(*m*-1). Observe that – with the correction introduced in the first term – if the processing time in all stages is 1, both terms are equal to *m*, which demonstrates that both have the same magnitude.

$$R(i)=\lambda\cdot\left(2\cdot\sum_{j=1}^{m}(m-j)\cdot p_{j,i}\Big/(m-1)\right)+(1-\lambda)\cdot\sum_{j=1}^{m}p_{j,i} \tag{9}$$

Notice that if λ=0, the job selected is the one with the minimum sum of processing time; whereas if λ=1, the selected job is the one that generates the minimum front delay.

The second step builds the remaining sequence to minimize the timeout of machines and the total flowtime, which is carried out with index *ind1*. The timeout is measured with the first term of equation (10), which is similar to the index used in the Profile Fitting procedure (McCormick, Pinedo, Shenker, & Wolf, 1989). However the total flow time is measured with the second term that evaluates the contribution of the considered job *i* to the total flowtime of the partial sequence.

$$ind1(i,k)=\mu\cdot(\sum_{j=1}^{m}(c_{j,k+1}(\sigma*i)-c_{j,k}(\sigma)-p_{j,i}))+(1-\mu)\cdot(C_i-C_{[k-1]}) \tag{10}$$

Hence, HPF2(λ,μ) can be described as follows:

- Step 1: selection of the first job of the sequence. Select the job with minimum R(*i*) and put it in the first position of sequence σ. Set *k*=1. In case of ties, select the job with minimum $p_{1,i}$.
- Step 2: construction of the remaining sequence. While *k*<*n*, calculate index *ind1* as in equation (10) for each unscheduled job *i*. Select the job with minimum *ind1*. In case of ties, select the job which leads to the partial sequence with minimum total flowtime.

Parameters λ and μ were selected by measuring the performance of the algorithm, which itself was done by combining several λ and μ values. For this test, we used 140

randomly generated instances that were grouped into 28 sets of size n x m, where n= {20, 50, 80, 110, 140, 170, 200} and m = {5, 10, 15, 20}. The evaluated values were λ={0.55, 0.6, 0.65, 0.7, 0.75} and μ={0.65, 0.70, 0.75,0.80, 0.85}. The performance was measured by the Relative Percentage Deviation (RPD) from the best solution (minimum total flowtime), which was obtained during the experiment using all combination of values. Therefore, RPD is calculated as in (11):

$$RPD = \frac{TF_k - TFref_k}{TFref_k} \cdot 100 \qquad (11)$$

where TF_k is the total flowtime obtained in instance k and $Tref_k$ is the minimum flowtime obtained in this instance by any combination of values.

The Average Relative Percentage Deviation (ARPD) of all RPDs obtained per each instance and combination of λ and μ values is shown in table 1. As can be seen, the best solutions were obtained when λ =0.65 and μ=0.75.

λ / μ	0.65	0.70	0.75	0.80	0.85
0.55	0.860	0.773	0.656	0.713	0.775
0.60	0.814	0.738	0.628	0.694	0.724
0.65	0.759	0.713	**0.565**	0.627	0.643
0.70	0.767	0.724	0.569	0.608	0.662
0.75	0.775	0.732	0.575	0.611	0.666

Table 1. ARPD of total flowtime values obtained by HPF2 per each λ and μ combination

Therefore, a food source was generated according to these parameter values. The creation of the remaining food sources depends on the strategy used. For the first strategy (STR1), we fixed parameter λ=0.65, and μ was selected randomly from a given range interval [μ_{min}, μ_{max}]; whereas the remaining solutions in the second strategy (STR2) were generated randomly. Selecting μ in a given interval that depends on n is explained by the compromise between the diversity of the solutions and their quality. For small values of n, a narrow interval could lead to very similar solutions. On the other hand, a narrower interval is required for higher values of n, because a huge interval could result in worse solutions in terms of total flowtime. Therefore, we set the interval depending on n according to the values in Table 2.

n	μ_{min}	μ_{max}
$0 < n < 75$	0	1
$75 \leq n < 150$	.5	1
$150 \leq n$	.6	.9

Table 2. Values of μ_{min} and μ_{max} for each range of n

The flowtime calculation in an n-job, m-machine flow shop for a given sequence is of complexity O(nm). Therefore, since k flowtimes in k jobs and m machines must be calculated in step 2, we can conclude that the complexity of this procedure is O(n^2m).

2.1.2 Second Part of food source generation

In our aim to investigate the convenience of initiating the algorithm with good food sources (solutions), a variable local search (named LS and based on swap and insert neighborhood structures) was implemented in this part. The procedures for exploring them were named LS1 and LS2, respectively.

In LS1, neighbors are generated for each job in the sequence by swapping one job with all jobs that follow it in the sequence. If the best neighbor (σ') is better than the current solution (σ), it becomes the new current solution σ, and the process continues until all jobs have been considered. To avoid constantly exploring neighborhoods in the same order, jobs are selected randomly.

In LS2, neighbors are generated for each job in the sequence by removing the job from its position and inserting it into all other possible positions. If the best neighbor (σ') is better than the current solution (σ), it becomes the new current solution σ, and the process continues until all jobs have been considered. As in LS1, jobs are selected randomly.

The implemented variable local search (Figure 2) uses both structures at each iteration, one after the other. The first neighborhood to be explored is selected randomly with a probability of 50%. After exploring the solutions that neighbor the current solution σ, the local optimum σ' is compared with σ. If the solution has improved, σ' replaces σ and the search continues throughout the other neighborhoods. This process goes on until the current solution is no longer improved. Next, the local optimum σ' is compared with the best solution σ* in terms of quality. If TF(σ') is less than TF(σ*), then σ' replaces σ*.

(please, insert figure 2 near here)

Figure 2. Pseudocode of the LS

Finally, the scheme for generating the initial food sources is shown in Figure 3.

(please, insert figure 3 near here)

Figure 3. Implemented strategies for generating initial food sources

2.2 Employed bees

In this phase, the employed bees are sent to the food source to evaluate their surroundings. In our implementation, two employed bees' were sent: the best one and another selected randomly. To enhance the exploration and be able to access a good food source, we tested two methods.

The first method (DC) applies the deconstruction and construction procedures proposed in Ruiz and Stützle (2007). The deconstruction procedure randomly extracts d jobs from the current sequence, and the construction procedure re-inserts them one at a time using the insertion procedure of NEH heuristic, starting with the first job that was removed until reaching the last one. According to the results obtained in a previous test, we set d=8. Next, the LS tries to improve the obtained solution and compares it with the original. The new one is kept only if it is better than the original.

In the second method a new scheme named *Three Neighborhood Operators* (TNO) is presented. This scheme consists of applying three operators to the two selected solutions. These operators were proposed by Della Groce, Narayan, and Tadei (1996) for the two-machine total completion time flow shop problem to generate neighboring solutions. The operators are defined as follows:

- PI (Pairwise Interchange): Given a sequence, σ, and two positions, $k1$ and $k2$, swap the jobs that are in these positions, i.e.: σ = (5,3,1,2,4), $k1$ = 1 and $k2$ = 4; the resulting sequence is σ_0 = (2,3,1,5,4).
- EFSR (Extraction and Forward Shifted Reinsertion): Given a sequence (σ) and two positions ($k1$, $k2$), with $k2$ later in the sequence than $k1$, extract the job at position $k2$ and reinsert it in position $k1$, i.e.: σ = (5,3,1,2,4), $k1$=1 and $k2$ = 4; the resulting sequence is σ_0 = (2,5,3,1,4).
- EBSR (Extraction and Backward Shifted Reinsertion): Given a sequence (σ) and two positions ($k1$, $k2$), with position $k1$ before $k2$ in the sequence, extract the job at position $k1$ and re-insert it in position $k2$, i.e.: σ = (5,3,1,2,4), $k1$ = 1 and $k2$ = 4; the resulting sequence is σ_0 = (3,1,2,5,4).

The TNO starts by randomly selecting $k1$ and $k2$ ($k1 < k2$). Next, the three operators are applied to the selected sequence (σ), and the best solution among the three new sequences is chosen (σ'). This process is done t times. In our implementation, t was set

to 2 in accordance with the results obtained in a previous test. Next, the LS procedure tries to improve the obtained solution and then compares it with the original (σ). The new one (σ'') is kept only if it is better than the original. The TNO scheme is described in Figure 4.

(please, insert figure 4 near here)

Figure 4. Pseudocode of the employed bee phase

2.3 Onlooker bees

The onlookers look out for a food source to exploit. They wait in the nest and establish a food source through the information shared by employed bees. In this phase, we tested two strategies: path relinking and the single-point crossover operation.

Path relinking is a search technique originally proposed by Glover and Laguna (1998) to explore the path between two sets of good solutions. In our implementation, two solutions are selected: the best one and another selected randomly from the food source set. The best solution is the destination, and the other solution is the path origin. The path is built by interchanging movements in order to convert the original solution into the destination solution. Therefore, the final solution is the reference and the other one is continuously changed with each movement. The process starts by comparing both solutions and detecting the jobs that occupy different positions in both solutions. Next, the first job (according to its number) is in a different position from the original solution and is interchanged with the one that occupies that position. The new solution is evaluated and replaces the original one only if it is better. The process continues until the original solution is equal to the destination solution. Notice that if there are k jobs in different positions, a maximum of k-1 movements are necessary because the last one leads the permutation to the reference one. Hence, path relinking is carried out only if more than two jobs can be swapped. For example, if σ=(5,4,1,2,3) and σ*=(2,3,1,5,4), jobs 2, 3, 4 and 5 are in different positions. The first movement in σ is a swap between 2 and 5, which leads to σ1=(2,4,1,5,3). If the total flowtime of σ1 is lower than σ, σ1 replaces σ. Now, σ1 and σ are compared and, as σ1 has only two jobs in positions that are different than σ, the process is stopped because the swap movement converts σ1 into σ*.

The single-crossover operator is typically used in genetic algorithms because it allows creating a new solution from two others. In our implementation, one of them was the

best solution (σ*), and the other one (σ) was selected randomly from the set of food sources. The process starts with randomly generating a cut point on σ. Next, the first part of σ is copied to offspring 1, and the remaining positions are filled with the jobs not included in the first part, in the relative order that they have in σ*. The second offspring is created by copying the second part of σ and filling the remaining positions according to the relative order that they have in σ*. The two offspring are evaluated and the best one replaces σ only if it is better. An example is shown in figure 5.

(please, insert figure 5 near here)

Figure 5. Example of the crossover operator

2.4 Scout bees

The scouts seek new food sources. In our implementation, a new solution is created according to the strategy followed in the two steps of the food source phase, i.e., with HPF2 (0.65; μ) or randomness in step 1 and with or without applying the variable local search in step 2. As in the other phases, two strategies were tested. The first strategy consists of replacing the worst solution in the food source set with the new solution; whereas the worst solution is replaced by the new one in the second strategy only if the latter is better.

3 Design of experiment for the DABC configuration

To identify the best configuration of the DABC algorithm, we used Design of Experiments (DOE) techniques (Box, Hunter, & Hunter, 2009). Given the nature of the factors (the 4 steps of the algorithm) and the strategies for studying each of them, we decided to use a two-level factorial design. This type of design is a very useful experimentation methodology; it allows estimating the size and assessing the significance of factor changes (in our case changes in the algorithm steps) in the response that interests us (in our case the RPD). A very interesting characteristic of factorial designs is that – on top of studying the effect of each factor by itself (known as the main effects) – they allow us to study their interactions. In other words, we can evaluate if the effect of one of the factors in the response depends on the level of the other factor. As will be seen later, this is what happens between factors P1 and P2 (Figure 6). Naturally, this fact makes this type of design especially suited to determining

the “best” algorithm. Table 3 shows the factors and levels considered. Notice that the algorithm’s first phase, how to get food sources, has been subdivided into two factors that we named initialization 1 and initialization 2.

Factors (Algorithm steps)	**Levels** 1	2
P1: initialization 1	STR1	STR2
P2: initialization 2	With LS	Without LS
P3: Employee bees	DC	TNO
P4: Onlooker bees	Path relinking	Crossover
P5: Scout bees	Always replace the worst solution	Replace the worst solution only if the new one is best

Table 3. Factors and levels considered in the factorial design

A two-level full factorial design with 5 factors (a 2^5 design) requires 32 runs. Such a design allows estimating 31 effects: 5 main effects, 10 two-factor interactions, 10 three–factor interactions, 5 four-factor interactions and one five-factor interaction. Since the effect of three and higher order interactions can be considered negligible (Box et al., 2009), it was decided to conduct a half fraction of the full design, a 2^{5-1} fractional factorial design (Table 4 presents the design matrix). This is a resolution V design that allows us to estimate all the main effects and two-factor interactions without any confounding among them. They are confounded with higher order interactions that, as commented above, can be considered negligible. Furthermore, if the analysis of results suggests that one of them may be important, it is always possible to conduct 16 additional runs that will form the full factorial together with the 16 initially conducted runs.

Run	**Source food 1**	**Source food2**	**Employee bees**	**Onlooker bees**	**Scout bees**
1	STR1	With LS	TNO	Path relinking	Replace if best
2	STR2	With LS	TNO	Path relinking	Replace always
3	STR1	Without LS	TNO	Path relinking	Replace always
4	STR2	Without LS	TNO	Path relinking	Replace if best
5	STR1	With LS	DC	Path relinking	Replace always
6	STR2	With LS	DC	Path relinking	Replace if best
7	STR1	Without LS	DC	Path relinking	Replace if best
8	STR2	Without LS	DC	Path relinking	Replace always
9	STR1	With LS	TNO	Crossover	Replace always
10	STR2	With LS	TNO	Crossover	Replace if best
11	STR1	Without LS	TNO	Crossover	Replace if best

12	STR2	Without LS	TNO	Crossover	Replace always
13	STR1	With LS	DC	Crossover	Replace if best
14	STR2	With LS	DC	Crossover	Replace always
15	STR1	Without LS	DC	Crossover	Replace always
16	STR2	Without LS	DC	Crossover	Replace if best

Table 4. Design matrix of the half fraction of the full design.

The resulting 16 algorithms from the combinations of the alternative procedures in each step were tested on a test-bed that was created *ad hoc* to separate the calibration benchmark from the final testing benchmark. Each algorithm was tested on a 2 GHz Intel Core 2 Duo E8400 CPU with 2 GB of RAM, with 140 randomly generated instances grouped into 28 sets of size *n* x *m*, where *n*= {20, 50, 80, 110, 140, 170, 200} and *m* = {5, 10, 15, 20} with 5 instances per group. So, we can say that the final design is a $7*4*2^{5-1}$ design that requires 448 runs. On top of that, each experimental condition was replicated five times; thus the final number of runs conducted was 448*5=2240.

The resulting algorithms performance was measured by the Relative Percentage Deviation (RPD), as in equation (11). In this case TF_k was the average total flowtime of the 5 runs at instance *k*, and $TFref_k$ was the minimum total flowtime obtained at instance *k* by any of the 16 algorithms in any of the 5 runs.

One important issue to take into account when analyzing the results is that, even though RPD is supposed to level out the differences due to the distinct level of difficulty presented by instances, it does not (Ribas et al., 2013). The usual way to remove this variability so that it does not make it difficult to identify significant factors (algorithm steps, in our case) is to consider the 140 instances as a blocking variable. Then, it is possible to compare the 16 algorithm variations (resulting from the 2^{5-1} design) without interferences from differences in the instances. The procedure is equivalent to analyzing the residuals of a linear regression between RPD as the independent variable and the instances as the dependent variable. We call this new variable RPD_Blck.

Source	Degrees of freedom	Sum of Squares	Adjusted Mean Square	F-Statistic	p-value	Significance
n	6	153.14	25.52	500.57	0.00	**
m	3	1.79	0.59	11.71	0.00	**
P1	1	35.32	35.32	692.83	0.00	**
P2	1	17.05	17.05	334.43	0.00	**
P3	1	0.17	0.17	3.41	0.06	*
P4	1	0.002	0.003	0.05	0.81	
P5	1	0.003	0.003	0.06	0.81	
n*m	18	12.95	0.71	14.11	0.00	**

***n**P1**	6	9.85	1.64	32.22	0.00	**
***n**P2**	6	4.79	0.79	15.68	0.00	**
***n**P3**	6	0.32	0.05	1.06	0.38	
***n**P4**	6	0.03	0.01	0.11	0.99	
***n**P5**	6	0.01	0.002	0.04	1.00	
***m**P1**	3	0.40	0.13	2.65	0.04	
***m**P2**	3	0.28	0.09	1.84	0.13	
***m**P3**	3	0.04	0.01	0.27	0.84	
***m**P4**	3	0.02	0.01	0.16	0.92	
***m**P5**	3	0.01	0.004	0.09	0.96	
P1*P2	1	11.45	11.45	224.64	0.00	**
P1*P3	1	0.01	0.01	0.14	0.70	
P1*P4	1	0.04	0.03	0.73	0.39	
P1*P5	1	0.001	0.001	0.03	0.85	
P2*P3	1	0.01	0.01	0.19	0.66	
P2*P4	1	0.001	0.001	0.02	0.89	
P2*P5	1	0.005	0.01	0.11	0.73	
P3*P4	1	0.0001	0.0001	0.00	0.97	
P3*P5	1	0.0002	0.0002	0.00	0.95	
P4*P5	1	0	0	0.00	0.97	
Error	2152	109.73	0.051			
Total	2239	357.49				

Table 5. ANOVA of RPD_Blck

By using RPD_Blck as the response of interest and considering the main effects and two-factor interactions, the analysis of the experiment yields the ANOVA presented in table 5, where significant effects at the 0.05 level are marked with two asterisks and those at the 0.1 level with one.

The residual analysis does not present any violation of the Analysis of Variance assumptions and, thus, the results can be readily interpreted.

There are nine significant effects that can be classified into three groups:

- Three effects that are the natural consequence of the differences in the difficulties of the problem: *n*, *m* and the *n***m* interaction. They were expected and, in fact, are of no interest
- Four effects of the algorithm steps: P1, P2, the interaction P1*P2 and, to a lesser degree, P3. P1 and P2 (initialization 1 and 2) represent the effects of food source generation. Since P1 and P2 interact, their effects have to be analyzed together and a plot is an excellent way to do it. Figure 6 shows these effects: it is clear that, for P1, STR1 is always better than STR2 and that P2 has little effect when STR1 is used. This fact means that it is better to generate the whole population with the HPF2(0.65,μ) procedure. Furthermore the quality and diversity of these solutions

means that the application of LS is not necessary. However, when STR2 is used, P2 is always better when the level *with LS* is chosen. That is, if most of the solutions have been generated randomly, the population is poor and needs to be improved with LS. In spite of that, the effects of P1 and P2 have to be analyzed together, as mentioned previously. In Figure 7, we show the main effects of P1, P2 and P3 together, so that that it can be seen that the effect of P3 (the strategy used by the employed bees) is very small in comparison. The best level is TNO.

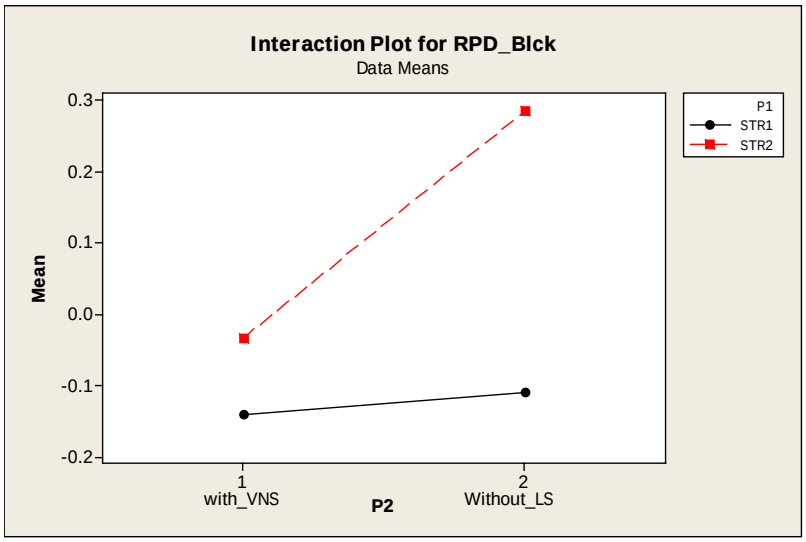
- 

Figure 6. Interaction plot of P1 and P2

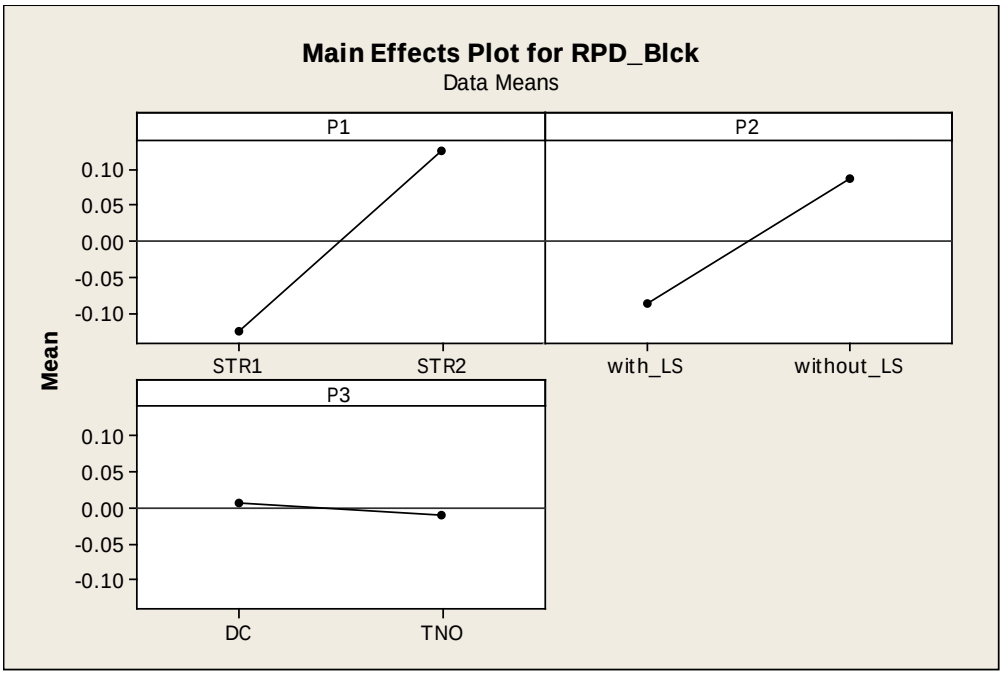


Figure 7 Main effects plot of P1, P2 and P3

- Two interactions between the number of jobs (*n*) and the algorithm steps: *n**P1 and *n**P2. These interactions reflect the fact that, when the number of jobs is small, all strategies behave very well. As an example, Figure 8 shows the *n**P1 interaction. It

is clear that for n=20 and n=50 both levels of P1 provide similar results; while the difference is evident between STR1 and STR2 when n is bigger. There is also an interaction between the number of machines (m) and the algorithm's first step; of course, this has the same explanation as for the n*P1 interaction commented on above.

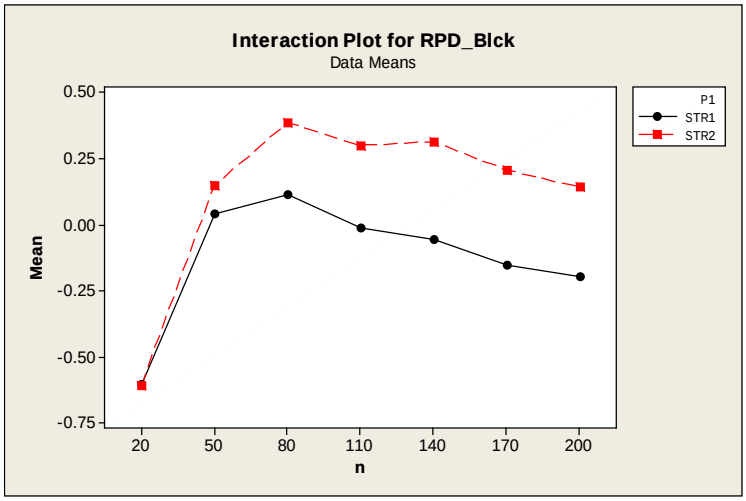


Figure 8. n*P1 interaction

Finally, as a result of this analysis, we concluded that the configuration of the proposed DABC algorithm was formed as indicated in table 6. Its outline can be seen in Figure 9.

P1: Initialization 1	STR1
P2: Initialization 2	Without LS
P3: Employee bees	TNO
P4: Onlooker bees	Path relinking
P5: Scout bees	Replace the worst solution only if the new one is best

Table 6. Final configuration of the proposed DABC

Figure 9. Outline of the DABC algorithm

3.1 Experimental adjustment of DABC parameters

A golden rule of experimental design is to not try to learn everything at once from a first experiment (Box et al., 2009). The idea is to use what is called a sequential strategy: run an experiment, learn from it and use it to design a follow-up experiment. This is what we have done. After selecting the basic structure of the algorithm, we adjusted (recalibrated) the main parameters: number of sources (*fs*) and the number of times that

```
procedure DABC
   Set parameter fs
    Food sources:      Generate the initial population with STR1;
                       σ_best =the best solution in the population;
                       σ_worst=the worst solution in the population;

  While (stopping criterion not met)
      Employed bees:  for σ_best and another σ of the population randomly selected:
                            for j=1 to t
                             apply the TNO procedure
                            endfor
                      endfor
      Onlooker bees:  Select a solution σ of the population randomly;
                            apply the path relinking procedure from σ to σ_best;
      Scout bees:     Generate a new food source (σ_new) with HPF2(0.65,μ)
                      if σ_new < σ_worst  then
                            replace the worst solution for the new one;
                      endif
   endwhile;
end
```

the neighborhood operators are applied (*t*).The best levels found for these two parameters in the first experiment were 6 and 2, respectively. Now, in a new experiment, we move these values around a bit to see if we can further improve the RPD index. The selected levels were:

fs: 5,6,7

t: 1,2,3

Calibration was done on the same test-bed used for the configuration of the algorithm, which, as said before, is different than the one used in the final testing. The alternatives were compared using the RPD index.

The ANOVA results are shown in table 7, where one can see that the only significant parameters that are of no interest (aside from *n*, *m* and *n***m*) are *t* (*p*-value=0.001) and the interaction *n***t* (*p*-value 0.012). Figure 10 shows the *n***t* interaction, where it can be seen that the interaction is weak and it does not affect the conclusion that can be reached

from Figure 11: that *t* can be set to either 1 or 2 because there is no difference between them. Therefore, this test confirms the previous parameters.

Source	Degrees of freedom	Sum of Squares	Mean Square	F-Statistic	p-value
n	6	239.1052	39.851	914.16	0.000
m	3	1.994	0.664	15.25	0.000
fs	2	0.086	0.043	1.00	0.370
t	2	0.592	0.296	6.8	0.001
n*m	18	13.240	0.735	16.87	0.000
n*fs	12	0.148	0.012	0.28	0.992
n*t	12	1.124	0.093	2.15	0.012
m*fs	6	0.146	0.024	0.56	0.763
m*t	6	0.177	0.029	0.68	0.666
fs*t	4	0.105	0.026	0.61	0.658
Error	3078	161.643	0.043		
Total	3779	418.364			

Table 7. Analysis of Variance for RPD versus *n*, *m*, *fs* and *t*

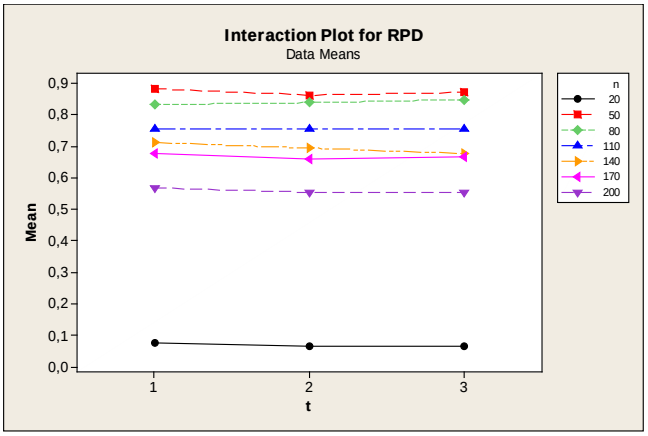


Figure 10. Interaction plot of *n*t*

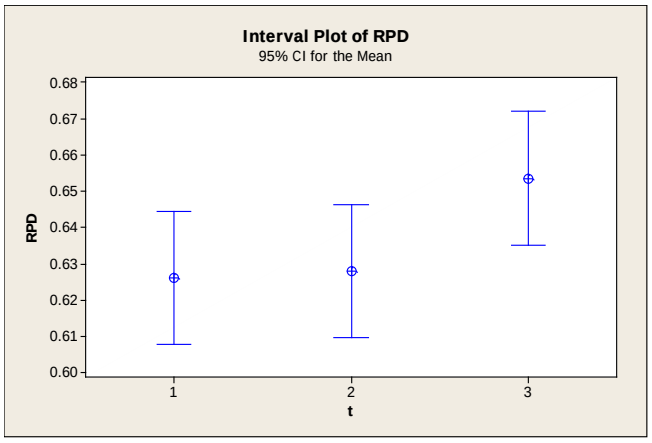


Figure 11. Interval plot of *t*

4 Computational evaluation

In this section, the performance of the proposed DABC, named DABC_RCT, is compared against the algorithms proposed in the literature for the problem at hand: the Harmony Search (HS) algorithm (Wang et al., 2010), denoted as HS_WPT; the Discrete Artificial Bee Colony (DABC) algorithm (Deng et al., 2012), denoted as DABC_DXG; and an IG algorithm proposed by Khorasanian and Moslehi (2012), denoted as IG_KM. Moreover, in order to show its performance, we included the HPF2 procedure (0.65, 0.75) in the comparison, which is used to find the first food source in the proposed DABC.

All algorithms were coded in the same language (QuickBASIC) and tested on the same computer, a 3 GHz Intel Core 2 Duo E8400 CPU with 2 GB of RAM. To make a fair comparison, all algorithms adopted the CPU time limit as a stopping criterion, which was fixed at $k \cdot n^2 \cdot m \cdot 10^{-5}$ seconds, with k set to 15 and 30 in order to analyze the performance of these algorithms for two levels of CPU time. In each test, five runs were carried out by each algorithm for all 150 instances.

The test was done using Taillard's benchmark (Taillard, 1993) for the blocking flow shop scheduling problem and using the total flowtime criterion, as is done in Wang et al. (2010), Khorasanian and Moslehi (2012) and Deng et al. (2012). However, the latter authors use only the first 90 instances. Taillard's test-bed is composed of 120 instances (12 sets of 10 instances each), from 20 jobs and 5 machines to 500 jobs and 20 machines, where $n \in \{20, 50, 100, 200, 500\}$ and $m \in \{5, 10, 20\}$, although not all combinations of n and m are available. In particular, sets 200x5, 500x5 and 500x10 are missing, but they were added as in Pan and Ruiz, (2012) in order to maintain the orthogonality of the experiment.

As in the other tests, the performance of each algorithm was measured by the Relative Percentage Deviation (RDP) index, as in (11). In this case, $TF_{k.}$ was the average total flowtime obtained at instance k in the 5 runs, and $TFref_k$ was the best known solution for this instance. The best known solutions are reported in Table 12 at the end of this section.

The results are shown in Table 8-9, where we have averaged the RPD values (ARPD) of the 10 instances of each nxm group, for k=15 and k=30 in the stopping criterion, respectively. We can see that the ranking between algorithms is the same in both cases, and their convergence is similar. Notice that the effect of duplicating the CPU time is noteworthy in the performance of the small instances, but it diminishes when n increases. This fact indicates that it is necessary to increase the factor n in the CPU time limit even more, i.e., applying n^3m instead of n^2m.

nxm	HPF2 (0.65,0.75)	DABC_DXG	HS_WPT	IGA_KM	DABC_RCT
20x5	4.038	0.077	0.518	0.181	0.109
20x10	3.156	0.066	0.302	0.148	0.072
20x20	3.989	0.036	0.114	0.083	0.042
50x5	3.923	2.153	5.480	2.484	1.439
50x10	3.665	2.022	4.642	1.846	1.292
50x20	5.036	1.368	3.301	1.214	0.886
100x5	3.967	3.635	7.931	4.434	1.867
100x10	4.094	3.897	6.563	3.353	1.949
100x20	5.554	3.008	5.138	2.263	1.856
200x10	2.243	3.308	6.394	3.814	1.267
200x20	2.779	2.752	4.650	2.545	1.129

500x20	1.692	3.704	5.424	3.711	0.550
200x5	2.394	3.389	8.886	5.194	1.211
500x5	1.128	4.383	10.304	6.627	0.537
500x10	1.512	5.039	8.208	5.566	0.649
All	3.278	2.589	5.190	2.898	**0.990**

Table 8. ARPD for each *n*x*m* set and algorithm when *k*=15

*n*x*m*	HPF2 (0.65,0.75)	DABC _DXG	HS_WPT	IGA_KM	DABC_RCT
20x5	4.038	0.029	0.228	0.122	0.049
20x10	3.156	0.031	0.148	0.072	0.062
20x20	3.989	0.005	0.058	0.066	0.022
50x5	3.923	1.735	5.021	2.317	1.242
50x10	3.665	1.552	4.311	1.596	1.147
50x20	5.036	0.967	2.999	0.947	0.715
100x5	3.967	3.107	7.860	4.007	1.674
100x10	4.094	3.462	6.570	3.095	1.722
100x20	5.554	2.559	5.026	2.008	1.660
200x10	2.243	3.353	6.400	3.638	1.142
200x20	2.779	2.720	4.673	2.326	0.979
500x20	1.692	3.164	5.411	3.564	0.513
200x5	2.394	3.453	8.784	4.968	1.119
500x5	1.128	2.953	10.340	6.452	0.515
500x10	1.512	3.803	8.215	5.348	0.623
All	3.278	2.193	5.070	2.702	**0.879**

Table 9. APRD for each *n*x*m* set and algorithm when *k*=30

Regarding the performance of the algorithms, we can see that the DABC_RCT performs substantially better than the other algorithms at these two CPU time levels. However, DABC_DXG shows better performance than DABC_RCT when n= 20, but the proposed algorithm is considerably better for the set of instances where n>20. It is worth noting that the HPF2(0.65,μ) procedure (which is proposed for generating the initial food sources) performs considerably better than HS_WPT in the set of instances with n>20, and better than IGA_KM and DABC_DXG in the set of instances with more than 100 jobs. This fact is one explanation for why the proposed DABC_RCT has a better performance than the DABC_DXG. DABC_DXG generates the initial population (food sources) similarly to our STR2 strategy; i.e., one solution is generated by a heuristic, in this case a modified NEH algorithm, and the other solutions are generated randomly. Next, the solutions are improved by an insertion-based local search to improve the quality of the population. But our experiment discarded these strategies. One explanation is because the improvement of solutions through local search consumes much time, which diminishes the algorithm's capacity to perform more

iterations in order to find new food sources to explore. Instead, HPF2(λ,μ) gives good solutions in little time, which allows the algorithm to find better solutions. This fact allows us to say that it is recommendable to start the DABC algorithms with an efficient heuristic. To confirm this observation, we have modified the DABC_DXG by changing its procedure so that it will generate the initial food sources through the HPF2(λ,μ) procedure. The results for *k*=30 are shown in table 10, where the column DABC_DXG2 shows the results obtained by the modified DABC_DXG algorithm. It is worth noting the improvement of DABC_DXG2 when compared with DABC_DXG, which confirms our hypothesis. Now, the differences between DABC_RCT and DABC_DXG2 have diminished, but the DABC_RCT remains better.

nxm	DABC_RCT	DABC_DXG	DABC_DXG2
20×5	0.049	0.029	0.029
20×10	0.062	0.031	0.028
20×20	0.022	0.005	0.024
50×5	1.242	1.735	1.350
50×10	1.147	1.552	1.230
50×20	0.715	0.967	0.836
100×5	1.674	3.107	2.011
100×10	1.722	3.462	2.11
100×20	1.660	2.559	2.035
200×10	1.142	3.353	1.198
200×20	0.979	2.720	1.123
500×20	0.513	3.164	0.567
200×5	1.119	3.453	1.157
500×5	0.515	2.953	0.528
500×10	0.623	3.803	0.732
All	**0.879**	2.193	0.997

Table 10. APRD for each *nxm* set and algorithm when *k*=30

Another difference was also significant during the design of experiments to configure the proposed DABC, and that is the strategy used to find a neighboring solution for the food source. In this part of the algorithm, DABC_RCT uses three neighbor operators that use swap and insert movements, and DABC_DXG uses strategies to generate neighbors that only use insert movements. Therefore, from the obtained results, we recommend combining swap and inserting movements to generate neighboring solutions that allow diversifying the search.

To check whether the observed differences from Table 9 are indeed statistically significant, we carried out an analysis of variance (ANOVA) where the heuristics type (*algorithm*) is considered a factor. Thanks to the addition of test beds to the Taillard collection, the ANOVA is balanced and has the advantage of a higher detection power.

The ANOVA hypotheses were tested by a residual analysis, which showed small departures from normality; fortunately, the ANOVA method is robust to violations of this assumption. Therefore, the very clear results that were obtained in the ANOVA validate the conclusions and make a deeper analysis unnecessary. The ANOVA table (Table 11) shows that factors: *algorithm*, *n* and *m* and their interaction are highly significant.

Source	Degrees of freedom	Sum of Squares	Mean Square	F-Statistic	p-value
n	4	920.110	230.027	422.59	0.000
m	2	96.805	48.403	88.92	0.000
algorithm	4	1416.775	354.194	650.70	0.000
*n*m*	8	37.377	4.672	8.58	0.000
*n*algorithm*	16	1161.787	72.612	133.4	0.000
*m*algorithm*	8	212.352	26.544	48.76	0.000
Error	707	384.842	0.544		
Total	749	4230.048			

Table 11. ANOVA test for the comparison of algorithms

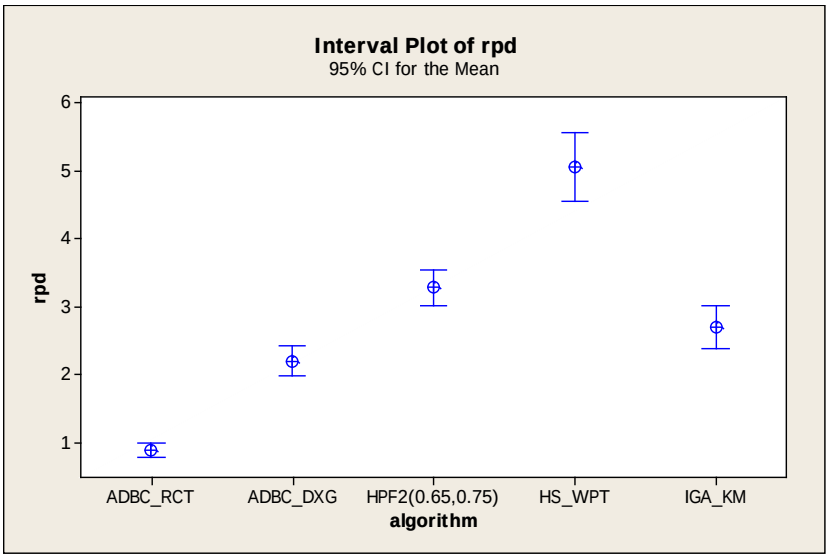


Figure 12. Interval plot of RPD by algorithm when *k*=30

Figure 12 reports the means and 95% confidence intervals. Note that there are no overlaps between intervals, which means that the differences observed from Table 9 are significant at the 95% confidence level. Additionally, in Figures 13-14 we can observe the interaction between *m* and *n* with the algorithms. Notice in Figure 13 that DABC_DXG and DABC_RCT are the algorithms which are less influenced by *m*.

However, the behavior of HS_WPT and IGA_KM is significantly different in respect to the *m* values. Both perform better when *m* increases, the opposite of the behavior of HPF2(0.65,0.75).

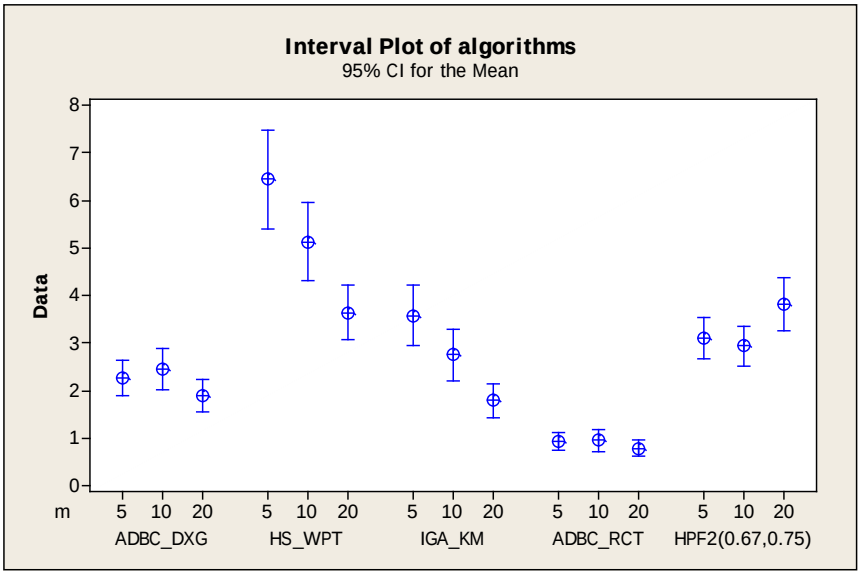


Figure 13. Interval plot of RPD by algorithm and *m* when *k*=30

In Figure 14 we can observe that HS_WPT and IGA_KM perform worse when the number of jobs increases. However, in DABC_RCT and HPF2(0.65,0.75), we can see a progressive reduction in the performance for *n*=50 and 100, but after *n*=100 we can observe an increase in the efficiency with *n*. One way to increase the efficiency of DABC_RCT even more, for example, would be to try to increase the efficiency of the HPF2(0.65, 0.75) for *n*≤100 by applying the insertion phase of NEH to the obtained solution with HPF2(0.65, 0.75) in those instances when *n*≤100.

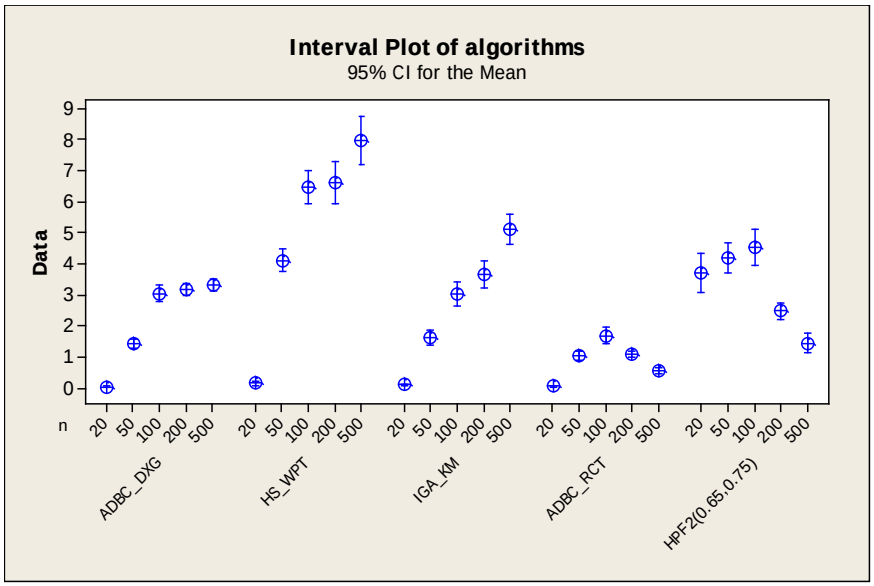


Figure 14. Interval plot of RPD by algorithm and *n* when *k*=30

Finally, we reported the new best solutions found during this research (Table 12) for most of the Taillard instances used in the BFSP, which could serve as a basis for comparison in future research.

Set	Best	Set	Best	Set	Best
20×5		20×10		20×20	
1	14953	11	22358	21	34683
2	16343	12	23881	22	32855
3	14297	13	20873	23	34825
4	16483	14	19916	24	33006
5	14212	15	20196	25	35328
6	14624	16	20126	26	33720
7	14936	17	19471	27	33992
8	15193	18	21330	28	33388
9	15544	19	21585	29	34798
10	14392	20	22582	30	33174
50×5		50×10		50×20	
31	72672	41	99674	51	136865
32	78140	42	95608	52	129958
33	72913	43	91791	53	127617
34	77399	44	98454	54	131889
35	78353	45	98164	55	130967
36	75402	46	97246	56	131760
37	73842	47	99953	57	134217
38	73442	48	98027	58	132990
39	70871	49	96708	59	132599
40	78729	50	98019	60	135710
100x5		100x10		100×20	
61	288332	71	354083	81	425224
62	280491	72	333379	82	435289
63	276228	73	343957	83	430634
64	259596	74	359259	84	432314
65	273086	75	338537	85	426405
66	267381	76	327254	86	430308
67	274744	77	335366	87	436642
68	269689	78	343174	88	440930
69	284816	79	344563	89	432876
70	282005	80	347845	90	437286
200×10		200×20		500x20	
91	1281633	101	1499623	111	8719682
92	1283164	102	1541253	112	8849228
93	1277933	103	1546279	113	8789777
94	1271502	104	1540822	114	8828454
95	1275901	105	1514600	115	8796337
96	1251213	106	1528885	116	8837577
97	1304158	107	1532090	117	8729909
98	1298900	108	1543229	118	8800506
99	1277801	109	1524293	119	8782791
100	1273794	110	1535329	120	8849551
200×5		500x5		500x10	
121	1071652	131	6389122	141	7552404
122	1026640	132	6415066	142	7665025
123	1059120	133	6460745	143	7626599
124	1044074	134	6334201	144	7626405

125	1064274	135	6373873	145	7479900
126	1021482	136	6282522	146	7537299
127	1082018	137	6244926	147	7510712
128	1043921	138	6352627	148	7562013
129	1057482	139	6328390	149	7550242
130	1037496	140	6309180	150	7549596

Table 12. Best solutions for the blocking flow shop with flowtime criterion

5 Conclusions

In this paper, we have presented an efficient Discrete Artificial Bee Colony algorithm, named DABC_RCT, for sequencing jobs in a blocking flow shop with the objective of minimizing the total flowtime of jobs. To configure the proposed algorithm, we considered four strategies for the food source phase and two strategies for each of the other phases (employed bees, onlookers and scouts). The final composition of the algorithm was decided by means of a Design of Experiments (DOE) that allowed us to estimate not only the effect of each part of the algorithm but also their interaction, which makes this type of design especially suited to determining the best algorithm. The experiment allowed us to prove that the employed initialization scheme has great influence on the performance of the algorithm. In our algorithm, we implemented a new method named HPF2(λ,μ), which allowed us to generate initial food sources that were diversified and of good quality. Another significant part of the algorithm was the employed bees phase, which is where the algorithm diversifies the search. In this phase, the selected strategy was a new scheme that combined insert and swap movements. The comparison of DABC_RCT with other algorithms proposed in the literature for this problem demonstrates its effectiveness and superiority.

In our experiment, the onlookers and scout phases have not been significant in the performance of the algorithm. This means that the strategies used do not contribute to increasing its efficiency. Therefore, it should be necessary to implement other schemes in order to find good procedures that would allow complementing the other parts.

One future research direction is to adapt the DABC_RCT to other objectives, such as tardiness, where the research is scarce, or to multi-objective functions. One of the main points in adjusting the algorithm is to have efficient heuristics available for creating a good food source. Therefore, research in this direction may also be necessary. Another interesting line of research would be to consider other restrictions, such as setup times, since this constraint is found in most manufacturing environments. Finally, it would be

very interesting to adapt the algorithm to solve the scheduling problem in a distributed flow shop, because this configuration allows us to represent situations that arise in the supply chain.

References

Armentano, V. A., & Ronconi, D. P. (2000). Minimização do Tempo Total de Atraso no Problema de Flowshop com Buffer Zero através de Busca Tabu. Gestao & Produçao, 7(3), 352.

Box, G., Hunter, J. S., & Hunter, W. (2009). Statistics Experimenters. Book, 1–655.

Caraffa, V., Ianes, S., Bagchi, T., & Sriskandarajah, C. (2001). Minimizing makespan in a blocking flowshop using genetic algorithms. International Journal of Production Economics, 70(2), 101–115.

Della Groce, F., Narayan, V., & Tadei, R. (1996). The two-machine total completion time flow shop problem. European Journal of Operational Research, 90, 227–237.

Deng, G., Xu, Z., & Gu, X. (2012). A Discrete Artificial Bee Colony Algorithm for Minimizing the Total Flow Time in the Blocking Flow Shop Scheduling. Chinese Journal of Chemical Engineering, 20(6), 1067–1073.

Glover, F. W., & Laguna, M. (1998). Tabu Search, Volumen 1.

Gong, H., Tang, L., & Duin, C. W. (2010). A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. Disruption Management, 37(5), 960–969.

Grabowski, J., & Pempera, J. (2000). Sequencing of jobs in some production system. European Journal of Operational Research, 125(3), 535–550.

Graham, R. L., Lawler, E. L., Lenstra, J. K., & Rinnooy Kan, A. H. G. (1979). Optimization and approximation in deterministic sequencing and scheduling: A survey. Annals of Discrete Mathematics, 5, 287–326.

Han, Y.-Y., Gong, D., & Sun, X. (2014). A discrete artificial bee colony algorithm incorporating differential evolution for the flow-shop scheduling problem with blocking. Engineering Optimization, 1–20.

Han, Y.-Y., Liang, J. J., Pan, Q.-K., Li, J.-Q., Sang, H.-Y., & Cao, N. N. (2012). Effective hybrid discrete artificial bee colony algorithms for the total flowtime minimization in the blocking flowshop problem. The International Journal of Advanced Manufacturing Technology, 67(1-4), 397–414.

Karaboga, D. (2005). An idea based on honey bee swarm for numerical optimization. Technical Report-TR06, Erciyes University, Engineering Faculty, Computer Engineering Department, 10.

Karaboga, D., Gorkemli, B., Ozturk, C., & Karaboga, N. (2014). A comprehensive survey: artificial bee colony (ABC) algorithm and applications. Artificial Intelligence Review, 42(1), 21–57.

Karaboga, D., & Ozturk, C. (2011). A novel clustering approach: Artificial Bee Colony (ABC) algorithm. Applied Soft Computing, 11(1), 652–657.

Khorasanian, D., & Moslehi, G. (2012). An Iterated Greedy Algorithm for solving the blocking flow shop scheduling problem with total flow time criteria. International Journal of Industrial Engineering & Production Research, 23(4), 301–308.

Lei, D. (2012). Multi-objective artificial bee colony for interval job shop scheduling with flexible maintenance. The International Journal of Advanced Manufacturing Technology, 66(9-12), 1835–1843.

Li, J., & Pan, Q. (2014). Solving the large-scale hybrid flow shop scheduling problem with limited buffers by a hybrid artificial bee colony algorithm. Information Sciences.

Li, J.-Q., Pan, Q.-K., & Gao, K.-Z. (2011). Pareto-based discrete artificial bee colony algorithm for multi-objective flexible job shop scheduling problems. The International Journal of Advanced Manufacturing Technology, 55(9-12), 1159–1169.

Lin, S.-W., & Ying, K.-C. (2013). Minimizing makespan in a blocking flowshop using a revised artificial immune system algorithm. Omega, 41(2), 383–389.

Liu, J., & Reeves, C. R. (2001). Constructive and composite heuristic solutions to the P//∑Ci scheduling problem. Data Envelopment Analysis, 132(2), 439–452.

Liu, Y.-F., & Liu, S.-Y. (2013). A hybrid discrete artificial bee colony algorithm for permutation flowshop scheduling problem. Applied Soft Computing, 13(3), 1459–1463.

Martinez, S., Dauzère-Pérès, S., Guéret, C., Mati, Y., & Sauer, N. (2006). Complexity of flowshop scheduling problems with a new blocking constraint. European Journal of Operational Research, 169(3), 855–864.

McCormick, S. T., Pinedo, M. L., Shenker, S., & Wolf, B. (1989). Sequencing in an Assembly Line with Blocking to Minimize Cycle Time. Operations Research, 37, 925–936.

Moslehi, G., & Khorasanian, D. (2013). Optimizing blocking flow shop scheduling problem with total completion time criterion. Computers & Operations Research, 40(7), 1874–1883.

Nasiri, M. M. (2015). A modified ABC algorithm for the stage shop scheduling problem. Applied Soft Computing, 28, 81–89.

Nawaz, M., Enscore Jr, E. E., & Ham, I. (1983). A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega, 11(1), 91–95.

Pan, Q., Wang, L., Sang, H., Li, J., & Liu, M. (2013). A High Performing Memetic Algorithm for the Flowshop Scheduling Problem With Blocking. IEEE Transactions on Automation Science and Engineering, 10(3), 741–756.

Pan, Q.-K., Wang, L., Li, J.-Q., & Duan, J.-H. (2014). A novel discrete artificial bee colony algorithm for the hybrid flowshop scheduling problem with makespan minimisation. Omega, 45, 42–56.

Ribas, I., Companys, R., & Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. Omega, 39(3), 293–301.

Ribas, I., Companys, R., & Tort-Martorell, X. (2013). A competitive variable neighbourhood search algorithm for the blocking flowshop problem. European J. of Industrial Engineering, 7(6), 729–754.

Ribas, I., Companys, R., & Tort-Martorell, X. (2013). An efficient iterated local search algorithm for the total tardiness blocking flow shop problem. International Journal of Production Research, 51(17), 5238–5252.

Ronconi, D. P., & Henriques, L. R. S. (2009). Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. Omega, 37(2), 272–281.

Ruiz, R., & Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research, 177(3), 2033–2049.

Sethi, S. P., Sriskandarajah, C., Sorger, G., Blazewicz, J., & Kubiak, W. (1992). Sequencing of parts and robot moves in a robotic cell. International Journal of Flexible Manufacturing Systems, 4, 331–358.

Taillard, E. (1993). Benchmarks for basic scheduling problems. European Journal of Operational Research, 64(2), 278–285.

Tasgetiren, M. F., Pan, Q. K., Suganthan, P. N., & Chen, A. H. L. (2011a). A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. Information Sciences, 181, 3459–3475.

Tasgetiren, M. F., Pan, Q.-K., Suganthan, P. N., & Chen, A. H.-L. (2011b). A discrete artificial bee colony algorithm for the total flowtime minimization in permutation flow shops. Information Sciences, 181(16), 3459–3475.

Tasgetiren, M. F., Pan, Q.-K., Suganthan, P. N., & Oner, A. (2013). A discrete artificial bee colony algorithm for the no-idle permutation flowshop scheduling problem with the total tardiness criterion. Applied Mathematical Modelling, 37(10-11), 6758–6779.

Wang, C., Song, S., Gupta, J. N. D., & Wu, C. (2012). A three-phase algorithm for flowshop scheduling with blocking to minimize makespan. Computers and Operations Research, 39, 2880–2887.

Wang, L., Pan, Q.-K., & Fatih Tasgetiren, M. (2010). Minimizing the total flow time in a flow shop with blocking by using hybrid harmony search algorithms. Expert Systems with Applications, 37(12), 7929–7936.

Wang, L., Pan, Q.-K., & Tasgetiren, M. F. (2011). A hybrid harmony search algorithm for the blocking permutation flow shop scheduling problem. Computers & Industrial Engineering.

Wang, L., Zhou, G., Xu, Y., Wang, S., & Liu, M. (2011). An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 60(1-4), 303–315.

Wang, L., Zhou, G., Xu, Y., Wang, S., & Liu, M. (2012). An effective artificial bee colony algorithm for the flexible job-shop scheduling problem. The International Journal of Advanced Manufacturing Technology, 60(1-4), 303–315.

Wu, B., Qian, C., Ni, W., & Fan, S. (2012). Hybrid harmony search and artificial bee colony algorithm for global optimization problems. Computers & Mathematics with Applications.

Zhang, R., Song, S., & Wu, C. (2013). A hybrid artificial bee colony algorithm for the job shop scheduling problem. International Journal of Production Economics, 141(1), 167–178.

Figure 1. Grey area indicates the front delay of job J1

Figure 2. Pseudocode of the LS

Figure 3. Implemented strategies for generating initial food sources

Figure 4. Pseudocode of the employed bee phase

Figure 5. Example of the crossover operator