Omega 39 (2011) 293–301

Contents lists available at ScienceDirect

Omega

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

An iterated greedy algorithm for the flowshop scheduling problem with blocking

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

[a] Laboratori d'Organització Industrial, DOE – ETSEIB – Universitat Politècnica de Catalunya, Avda. Diagonal, 647, planta 7, 08028 Barcelona, Spain
[b] Departament de Estadística e Investigación Operativa – ETSEIB – Universitat Politècnica de Catalunya, Avda. Diagonal, 647, planta 6, 08028 Barcelona, Spain

A R T I C L E I N F O

Article history:
Received 21 July 2009
Accepted 23 July 2010
Available online 30 July 2010

Keywords:
Flowshop
Blocking
Makespan
Iterated local search

A B S T R A C T

This paper proposes an iterated greedy algorithm for solving the blocking flowshop scheduling problem for makespan minimization. Moreover, it presents an improved NEH-based heuristic, which is used as the initial solution procedure for the iterated greedy algorithm. The effectiveness of both procedures was tested on some of Taillard's benchmark instances that are considered to be blocking flowshop instances. The experimental evaluation showed the efficiency of the proposed algorithm, in spite of its simple structure, in comparison with a state-of-the-art algorithm. In addition, new best solutions for Taillard's instances are reported for this problem, which can be used as a basis of comparison in future studies.

1. Introduction

In the traditional flowshop model, it is assumed that there are buffers of infinite capacity between two consecutive machines where jobs can be stored until they can be processed in the next stage. In practice, there are many productive environments where the buffer capacity is limited or zero for technical reasons or due to the process characteristics [1]. In such situations, a job completed on one machine may block that machine until the next downstream machine is free. Blocking scheduling problems can be found in a variety of industrial systems. Grabowski and Pempera [2] describe an example in the production of concrete blocks where storage is not allowed in some stages of the manufacturing process. Gong et al. [3] describe a scheduling problem and the blocking constraint which arises in the iron and steel industry. Martinez et al. [4] consider a new blocking constraint, found in the treatment of industrial waste and the manufacturing of metallic parts, where a machine remains blocked by a job until its operation on the next machine is finished and it leaves the machine. Another flowshop environment without intermediate buffers is the no-wait flowshop, where the operations of a job have to be processed continuously from start to finish without interruptions. Therefore, no buffer storage is needed. One example to which the no-wait flowshop problem applies can be found in Oulamara et al. [5]. A detailed review on flowshop with blocking and no wait in process can be found in Hall and Sriskandarajah [1].

In this paper, we consider the permutation flowshop problem without buffers between two consecutive machines, which is known as the blocking flowshop problem (BFSP). In the BFSP, a set of n jobs must be processed in m machines, in the same order, from machine 1 to machine m. Each job i, $i \in \{1,2,\ldots,n\}$, requires a fixed non-negative processing time $p_{j,i}$ on every machine j, $j \in \{1,2,\ldots,m\}$. The setup times are considered to be included in the processing time. The objective is to find a sequence for processing the jobs such that the makespan is minimized. According to the notation proposed by Graham et al. [6], this problem is denoted as $Fm|block|C_{max}$.

With regards to complexity, Reddi and Ramamoorthy [7] show that $F2|block|C_{max}$ can be reduced to a special case of the travelling salesman problem (TSP) with $n+1$ towns (0, 1, 2, …, n) that can be solved in polynomial time using Gilmore and Gomory's algorithm [8]. Hall and Sriskandarajah [1] showed, using results obtained by Papadimitriou and Kanellakis [9], that the $Fm|block|C_{max}$ problem for $m \geq 3$ machines is strongly NP-hard.

Since the problem is NP-hard, it is more practical to use heuristic procedures when the problem to be solved has a large number of jobs, as is usual in industry. The simplest heuristics are constructive procedures, which use rules to assign a priority index to each job, in each step, to build a sequence. Some examples of constructive heuristics have been developed to solve the BFSP for makespan minimization, such as profile fitting (PF) [10], but some attempts have also been made to study the efficiency of adapting constructive heuristics of the permutation flowshop problem without buffer constraints (PFSP) to the BFSP. Leisten [11] adapted some procedures proposed for the PFSP and concluded that the NEH heuristic [12] was the best one. In NEH, as is well

* Corresponding author. Tel.: +34 93 401 65 87; fax: +34 93 401 60 54.
E-mail addresses: imma.ribas@upc.edu, iribasv@uoc.edu (I. Ribas).

0305-0483/$ - see front matter © 2010 Elsevier Ltd. All rights reserved.
doi:10.1016/j.omega.2010.07.007

known, jobs are first ordered according to the longest processing time (LPT) rule and are then iteratively inserted in a partial sequence in accordance with the initial order obtained in the first step. In the literature on the PFSP, this procedure has been widely studied and some modifications have been proposed in order to render it even more efficient. These modifications can be viewed as new ordering rules to be used in the first step or as tie-breaking methods to be used in the insertion phase when two positions lead to the same makespan [13–15]. The literature on BFSP has explored these ideas less extensively, even though the NEH procedure also gives a good performance for this case. Only Ronconi [16] proposes two different ordering procedures. We did not find any papers that apply tie-breaking ideas to the blocking case. Moreover, we analysed whether the reversibility property of the BFSP can be used to improve the solution, as proposed in [15].

Metaheuristics are more sophisticated heuristic procedures. Recently, different types of metaheuristics have been developed to solve the problem considered. Caraffa et al. [17] proposed a genetic algorithm (GA). Grabowski and Pempera [18] presented two tabu search (TS) algorithms and evaluated them with Taillard's benchmarks, using as a reference the solutions obtained and reported by Ronconi [19]. Wang et al. [20] proposed a hybrid genetic algorithm (HGA), Liu et al. [21] proposed an algorithm based on particle swarm optimization (HPSO) and Qian et al. [22] proposed an algorithm based on differential evolution (DE) that was later adapted to multi-criteria case [23] and, very recently, Wang et al. [24] proposed a hybrid discrete differential evolution algorithm which they claim outperforms the TS proposed in [18]. All of these algorithms are very sophisticated and sometimes difficult to implement without contacting the authors to obtain detailed information. Therefore, it is highly desirable that a heuristic be simple—that is, easily understood and adapted to specific constraints of industrial environments. One very simple and effective metaheuristic is the iterated greedy (IG) algorithm, which iteratively applies constructive heuristics to an incumbent solution and uses an acceptance criterion to decide whether the newly constructed solution should replace the current one. This type of algorithm has been successfully applied to the PFSP by Ruiz and Stützle [25]. However, to the best of our knowledge, it has not been used to solve the problem considered here.

The aim of this paper is to provide an improved NEH-based heuristic and a very competitive IG algorithm for the BFSP with the makespan criterion.

The paper is organized as follows. After this brief introduction, we describe the IG algorithm proposed and the adjustment of the parameters. In Section 3, the experimental results are analysed. Finally, Section 4 presents some concluding remarks.

2. Iterated greedy algorithm for the BFSP

The IG algorithm is a simple stochastic local search method, which generates a sequence of solutions by iterating over a greedy construction heuristic using destruction and construction phases. The destruction phase removes some jobs from the incumbent solution. In the construction phase, a new candidate solution is created by reconstructing a complete solution using a greedy constructive heuristic. Once the candidate solution has been completed, an acceptance criterion is applied to decide whether the constructed solution should replace the current solution. The process iterates between these two phases until a defined stopping criterion is met.

The IG is closely related to iterated local search (ILS). The main difference between the two is that ILS applies local search to perturbations of the current search point to extend the search space and to escape from deep local optima, whereas in IG the perturbation of the current solution is stronger because it is done by means of the destruction and reconstruction of the solution with a greedy constructive heuristic. Therefore, the IG is better suited than ILS to escape from strong local optima.

```
Procedure Iterated greedy
      Generate initial solution ($\pi_0$)
      $\pi = \pi_0$
      $\pi_{best} = \pi_0$
   repeat
      $\pi'$=Local Search ($\pi$)
      if Cmax($\pi'$) < Cmax($\pi_{best}$) then $\pi_{best} = \pi'$  endif
      $\pi$= Acceptance criterion ($\pi_{best}$, $\pi'$)
      $\pi'$=deconstruction ($\pi$)
      $\pi$= construction ($\pi'$)
      if Cmax($\pi$) < Cmax($\pi_{best}$) then $\pi_{best} = \pi$  endif
   until stopping criteria is met
End
```

Fig. 1. Pseudocode of the iterated greedy algorithm.

Fig. 1 shows the main structure of the proposed IG algorithm. In the next section, we describe each of its components.

3. Initial solution

The NEH heuristic is frequently used in many metaheuristics as an initial solution procedure due to its effectiveness in obtaining quality solutions for both the PFSP and the BFSP. Therefore, we considered using NEH as the constructive greedy algorithm for the application of the IG to the BFSP, but first we tried to improve its performance by applying adaptations of some ideas proposed in the literature on the PFSP.

The first idea is to use a different ordering rule to sequence the jobs. Ronconi [16] proposed using the *MinMax* (MM) or PF procedure instead of the LPT to improve the solutions obtained in the BFSP.

Another alternative is to test some tie-breaking strategies to use in the ordering phase (step 1) when two jobs have the same processing time, and in the insertion phase (step 2) when two positions result in the same makespan. It has been shown, in the PFSP, that the performance of NEH improves if ties in the insertion phase are treated with an appropriate method. Since ties exist in both steps, it could be interesting to consider tie-breaking methods not only for step 2 but also for step 1. To the best of our knowledge, no tie-breaking for step 1 has been proposed either for the PFSP or for the BFSP, whereas some efforts have been made to find tie-breaking methods for step 2 in the PFSP. In this study, we analysed the performance of the NEH heuristic with tie-breaking strategies in steps 1 and 2.

For the ordering phase, we tested the following methods, which make use of the two indices $S_{1i} = \sum_{j=1}^{m}(m-j)p_{j,i}$ and $S_{2i} = \sum_{j=1}^{m}(j-1)p_{j,i}$ of the Trapezium procedure [26], being $P_i = \sum_{j=1}^{m} p_{j,i}$

- Method S1: If jobs i and j have the same P_i and $S_{1i} > S_{1j}$, then i goes before j.
- Method S2: If jobs i and j have the same P_i and $S_{2i} > S_{2j}$, then j goes before i.
- Method S3: Let $S_{3i}=S_{1i}-S_{2i}$. If jobs i and j have the same P_i and $S_{3i} > S_{3j}$, then i goes before j.
- Method S4K: This criterion is equivalent to the ordering rule proposed by Kalczynski and Kamburowski [13] in the NEHKK1 heuristic. Let $S_{4Ki}=\min\{S_{1i}, S_{2i}\}$. If jobs i and j have the same P_i and $S_{4i} > S_{4j}$, then i goes before j.

- Method S4: Let $S_4 = \max\{S_{1i}, S_{2i}\}$. If jobs i and j have the same P_i and $S_{4i} > S_{4j}$, then i goes before j.

For the insertion phase, we tested the behaviour of the tie-breaking strategies proposed by Kalczynski and Kamburowski [13], Dong et al. [14] and Ribas et al. [15] for the PFSP.

Finally, according to [15], improved solutions can also be obtained if the reversibility property of the problem is used. This property can be articulated as follows:

Given an instance I, which we call the direct instance, with process times $p_{j,i}$, the reverse instance I' can be determined, with process times $p'_{j,i}$ calculated according to (1):

$$p'_{j,i} = p_{m-j+1,i} \quad j=1,2,\ldots,m \quad i=1,2,\ldots,n \tag{1}$$

For a permutation π, the value of $C_{\max}$ in I is the same as that obtained in I' for the reverse permutation π'. This means that a heuristic applied to I and its reverse I' yields different approximate solutions; thus, we can choose the better of the two solutions.

These three approaches were evaluated in order to define the specifications of the initial solution procedure. The computational evaluation and conclusions are presented in Section 4.

3.1. Local search

The performance of the IG improves when a local search is applied on the reconstructed solution before the acceptance criterion is used to decide from which solution the algorithm shall continue. The implemented local search is a variant of the non-exhaustive descent algorithm (NEDA). NEDA tries to improve the solution by swapping any two positions in the sequence. If, during the process, a new permutation improves the value of the objective function ($C_{\max}(\pi)$), it becomes the new current solution and the process continues until all of the positions have been permuted and no further improvement is possible. However, when the domain of the solution defined by the adopted neighbourhood contains plateaus, great improvements can sometimes be obtained if, during the search, some solutions with the same value as the current best solution (i.e. ties) are accepted. In the BFSP, we observed that the number of ties is much lower than in the PFSP. Even so, it could be interesting to find out if better solutions can be obtained if ties are accepted with a certain probability α. However, to avoid being trapped in the local search, we set a maximum number of ties (β) that can be accepted during the search. Therefore, the local search finishes when the number of ties reaches β or there is no change in the incumbent solution.

In addition, to avoid always exploring the neighbourhood in the same order, which can lead to one area being explored more intensively than others, we incorporated a tool which we call *revolver*. The *revolver* is a pointer vector whose components are initialized with the different positions that a job can have in the sequence. The components are randomly mixed and used to codify the searching positions in the neighbourhood of the solution. Given two pointers to positions i, j in the job sequence, their equivalents i_{rev} and j_{rev} are searched for in the revolver vector, rev, being $i_{rev} = rev(i)$ and$j_{rev} = rev(j)$. These new positions are used when the non-exhaustive descent search is applied. The suitability of this procedure is analysed in Section 4.2.

3.2. Acceptance criterion

The acceptance criterion is used to decide whether the solution obtained in the local search, π', should be accepted as the current solution, π, for the next iteration. In our implementation, if the improved solution is worse than the best solution found, π_{best}, is accepted with a probability of 50%.

3.3. Destruction and construction phases

The destruction phase consists in extracting d randomly chosen jobs from the current solution π and re-inserting them, one at a time, using the insertion procedure (step 2) of the NEH, as done in [25].

3.4. Experimental parameter adjustment of the algorithm

The proposed IG algorithm has only three parameters to be adjusted: the probability of accepting a tie solution (α), the maximum number of ties accepted (β) and the number of jobs extracted from the current solution (d). Not all levels of α and β are compatible, because when β is 0, α can only take the value 0. Therefore, we converted α and β into a single parameter (γ) that takes the values shown in Table 1. Notice that the levels are chosen in a way that will allow for the estimation of the effect of α if the value of β is not zero.

Consequently, γ has 7 levels whereas the number of levels of d, n and m have been fixed at 3, 7 and 4, respectively. To find the best level of each one, we conducted a $7 \times 3 \times 4 \times 7$ full factorial experiment with γ and d as controllable factors and m and n as non-controllable factors. Therefore, the final levels chosen for the four factors are:

γ : 1,2,3,4,5,6,7
d : 5,6,7
n : 20,50,80,110,140,170,200
m : 5,10,15,20

Due to the randomness of the improvement procedure, we generated five instances for each of the 28 factor combinations and we performed 5 runs per instance. This means that we processed 140 instances (28 sets of 5) for each combination of parameters with a computation time limit set at $10 \cdot n^2 m$ ms. The experiments were carried out on a 2.8 GHz Pentium IV with 512 MB of RAM.

To analyse the experimental results, we used the relative percentage deviation (RPD) calculated as (2)

$$RPD = \frac{Heur_{hs} - Best_s}{Best_s} 100 \tag{2}$$

where $Heur_{hs}$ was the average of the makespan values obtained by the heuristic h, in instance s, and $Best_s$ the lowest makespan known for this instance.

The results were analysed by a multiway analysis of variance (ANOVA) where n and m were non-controllable factors. To check the ANOVA model hypothesis (normality, homoscedasticity and independence), the standardized residuals were analysed and no major departure from the assumption was found. The only minor problem is a slight skewness (departure from normality) that

Table 1
Combination of α and β values for each of γ levels.

γ	α	β
1	0.25	$\sqrt{n}$
2	0.25	$\frac{\sqrt{n}}{2}$
3	0.5	$\sqrt{n}$
4	0.5	$\frac{\sqrt{n}}{2}$
5	0.75	$\sqrt{n}$
6	0.75	$\frac{\sqrt{n}}{2}$
7	0	0

cannot be corrected by any Box–Cox transformation [27]; in any case, as is well known [28], ANOVA is robust with respect to the normality assumption.

Table 2 shows the results of the analysis. The results indicate that the only significant controllable factor (p-value < 0.05) is γ. Of course, n and m are also significant, which is not a surprise given their nature; they were included in the ANOVA in order to take away their effect.

Interestingly, both factor interactions between γ and the non-controllable factors n and m are significant. This is due to the fact that $\gamma=7$, on average the best option, has a very similar performance to other options when the number of ties found increases. Fig. 2 shows the number of ties for the different values of n and m. For $n=80$, 110 and 200 and $m=5$, the number of ties is clearly greater than it is for larger m values.

According to Figs. 3 and 4, which show the $\gamma \times n$ and $\gamma \times m$ interactions for the abovementioned values of n and m, the performance of the algorithm is very similar regardless of the γ value.

A first exploration of the significant effect of γ by a main-effect plot (Fig. 5) shows that the main difference is between level 7 and the rest.

A more rigorous analysis of the differences in γ levels, using Dunnett's simultaneous confidence intervals (Table 3), shows clearly that the only significant difference is between level 7 and the others. Among the other levels there are no significant differences.

Table 2
ANOVA table on ARPD.

Source	DF	Seq SS	Adj SS	Adj MS	F	p
γ	6	6.3648	6.3648	1.0608	11.69	0.000
$\gamma * d$	12	0.3905	0.3905	0.0325	0.36	0.977
$\gamma * n$	36	6.9572	6.9572	0.1933	2.13	0.000
$\gamma * m$	18	4.2566	4.2566	0.2365	2.61	0.000
D	2	0.0365	0.0365	0.0182	0.2	0.818
N	6	240.3698	240.3698	40.0616	441.37	0.000
M	3	20.1877	20.1877	6.7292	74.14	0.000
$d * n$	12	0.6053	0.6053	0.0504	0.56	0.878
$d * m$	6	0.4309	0.4309	0.0718	0.79	0.577
$n * m$	18	57.8615	57.8615	3.2145	35.42	0.000
Error	2820	255.9597	255.9597	0.0908		
Total	2939	593.4205				

As a result of this test, α and β were set to 0 and d was set to 6 but, as it is not a significant factor, any of the two other values could have been chosen.

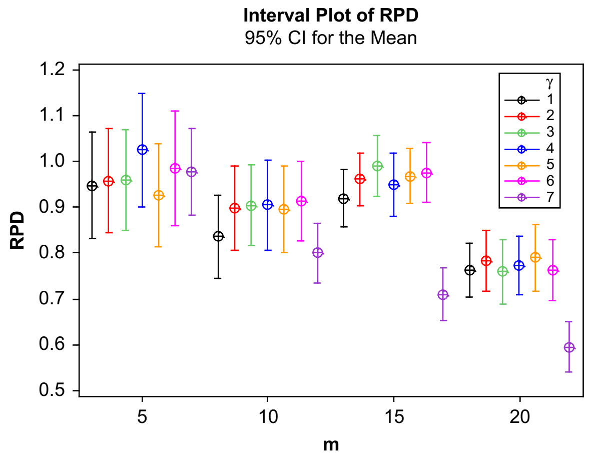

Fig. 3. Interaction between γ and m.

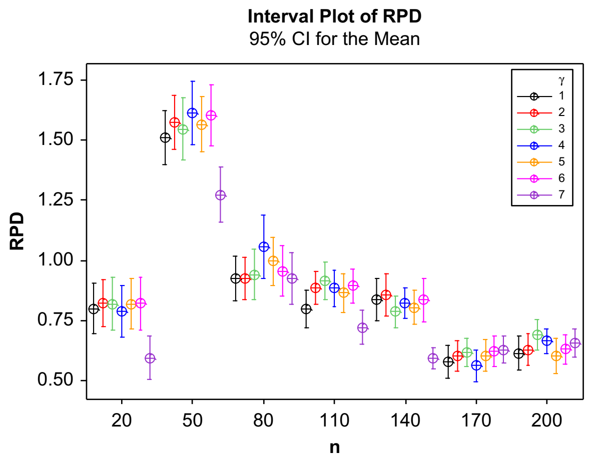

Fig. 4. Interaction between γ and n.

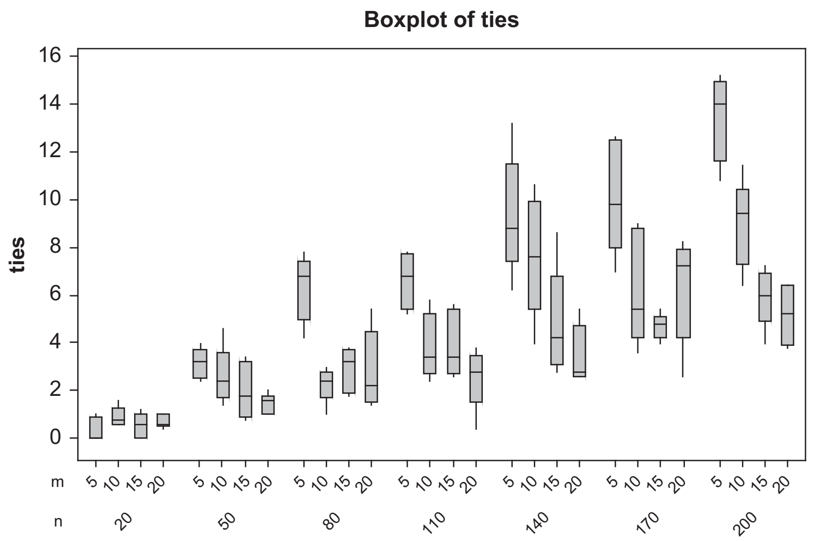

Fig. 2. Number of ties found in the instances used to calibrate the IG algorithm.

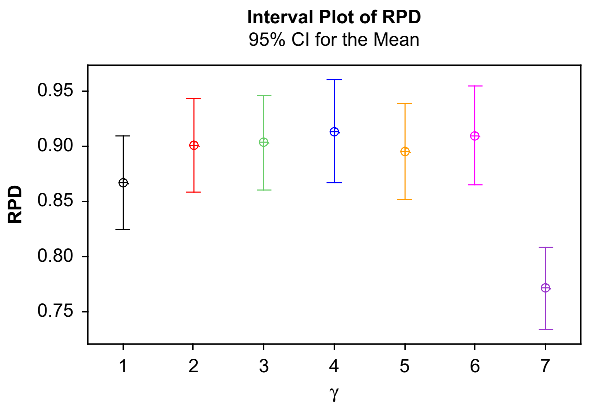

Fig. 5. Main-effect plot of γ.

Table 3
Dunnett's simultaneous tests. ARPD comparisons with control level $\gamma=7$.

γ	Difference of means	SE of difference	*T*-value	Adjusted *p*-value
1	0.09562	0.02079	4.6	0
2	0.12910	0.02079	6.21	0
3	0.13198	0.02079	6.35	0
4	0.14203	0.02079	6.83	0
5	0.12376	0.02079	5.95	0
6	0.13823	0.02079	6.65	0

4. Computational analysis

This section evaluates the performance of the variants of the NEH procedure at improving the initial solutions, as well as that of the proposed IG algorithm. Both tests were done using Taillard's benchmark [29]. Note that we used a different dataset to calibrate the IG parameters in order to avoid overfitting in the results. The Taillard benchmark is composed of 120 instances grouped in 12 sets of different sizes, ranging from 20 jobs and 5 machines to 500 jobs and 20 machines.

All programs were coded in the same language (QuickBASIC) and were tested on the same computer, a 3 GHz Intel Core 2 Duo E8400 CPU with 2 GB of RAM. To analyse the experimental results, we measured the RPD over the best-known solution, as in (2).

In this study, new best solutions were found in most of the Taillard instances. The best solutions for calculating the RPD are shown in Table 4, where the column *Dataset* indicates the size of the instances denoted as $n \times m$; *Best* indicates the best solution found; and *Source* indicates, by means of a number, the study that reported the method used: "1" indicates the hybrid discrete differential evolution (HDDE) algorithm proposed by Wang et al. [24] and "2" indicates the IG algorithm described here. The permutations associated with each solution can be found in [30]. The optimal solutions are marked with an asterisk. The optimality of these solutions was proved by the LOMPEN algorithm [31].

4.1. *Analysis of results for the variants of the NEH procedure*

The first step was to analyse the efficiency of the tie-breaking methods proposed for steps 1 and 2. Firstly, we implemented the NEH procedure with each of the tie-breaking methods to be used when two jobs have the same processing time (step 1). Each of these resulting procedures was run on the direct and reverse instances. Table 5 shows the results. The average relative percentage deviation (ARPD) corresponding to each set of Taillard's instances is shown in the dNEH rows and rNEH rows, respectively, and the better of the two values is shown in the

Table 4
Best solutions for Taillard's benchmark.

Dataset	Best	Source	Dataset	Best	Source	Dataset	Best	Source
20 × 5			50 × 5			100 × 5		
1	1374*	1,2	31	3002	2	61	6151	2
2	1408*	1,2	32	3201	2	62	6022	2
3	1280*	1,2	33	3011	2	63	5927	2
4	1448*	1,2	34	3128	2	64	5772	2
5	1341*	1,2	35	3166	2	65	5960	2
6	1363*	1,2	36	3169	2	66	5852	2
7	1381*	1,2	37	3013	2	67	6004	2
8	1379*	1,2	38	3073	2	68	5915	2
9	1373*	1,2	39	2908	2	69	6123	2
10	1283*	1,2	40	3120	2	70	6159	2
20 × 10			50 × 10			100 × 10		
11	1698*	1,2	41	3638	2	71	7042	2
12	1833	1,2	42	3507	2	72	6791	2
13	1659	1,2	43	3488	2	73	6936	2
14	1535	1,2	44	3656	2	74	7187	2
15	1617	1,2	45	3629	2	75	6810	2
16	1590	1,2	46	3621	2	76	6666	2
17	1622	1,2	47	3696	2	77	6801	2
18	1731	1,2	48	3572	2	78	6874	2
19	1747	1,2	49	3532	2	79	7055	2
20	1782	1,2	50	3624	2	80	6965	2
20 × 20			50 × 20			100 × 20		
21	2436	1,2	51	4500	2	81	7844	2
22	2234	1,2	52	4276	2	82	7894	2
23	2479	1,2	53	4289	2	83	7794	2
24	2348	1,2	54	4377	2	84	7899	2
25	2435	1,2	55	4268	2	85	7901	2
26	2383	1,2	56	4280	2	86	7888	2
27	2390	1,2	57	4308	2	87	7930	2
28	2328	1,2	58	4326	2	88	8022	2
29	2363	2	59	4316	2	89	7969	2
30	2323	1,2	60	4428	2	90	7993	2
200 × 10			200 × 20			500 × 20		
91	13,406	2	101	14,912	2	111	36,609	2
92	13,313	2	102	15,002	2	112	36,927	2
93	13,416	2	103	15,186	2	113	36,646	2
94	13,344	2	104	15,082	2	114	36,641	2
95	13,360	2	105	14,970	2	115	36,583	2
96	13,192	2	106	15,101	2	116	36,917	2
97	13,598	2	107	15,099	2	117	36,518	2
98	13,504	2	108	15,141	2	118	36,837	2
99	13,310	2	109	15,034	2	119	36,641	2
100	13,439	2	110	15,122	2	120	36,866	2

NEH2 rows. The tie-breaking method used in each case is indicated with its name in brackets after dNEH, rNEH or NEH2. The numeral 0 indicates that no specific method has been used, i.e. if two jobs have the same processing time, the first job is chosen. A comparison of the dNEH rows or rNEH rows separately indicates that the results achieved by any of these methods are a little better than those obtained by NEH without any tie-breaking. Note that the significant improvement is achieved by using the reversibility property of the problem, the NEH2 rows. The NEH2 rows show that there are considerable differences between the results achieved by each tie-breaking method. In particular, it stands out that the best-performing method is S4.

Since S4 turned out to be efficient at increasing the performance of NEH, next we analysed the results obtained by NEH with each of the tie-breaking methods proposed for the insertion phase (step 2) with and without S4 in step 1. Table 6 shows the results obtained by NEH with each tie-breaking method for step 2 and S4 for step 1. Table 7 shows the results obtained when no tie-breaking method is used in step 1. The first row, in both tables, indicates, by means of a number, the tie-breaking method implemented in the NEH procedure: "0" when

Table 5
ARPD obtained with each tie-breaking method used in step 1, on Taillard's instances.

n	20	20	20	50	50	50	100	100	100	200	200	500	
m	5	10	20	5	10	20	5	10	20	10	20	20	All
dNEH(0)	5.58	5.33	3.46	8.59	7.79	7.34	8.34	8.04	5.62	7.83	5.65	4.37	6.49
rNEH(0)	5.25	5.52	3.48	8.60	7.84	7.22	8.20	7.66	6.18	7.82	5.49	4.70	6.50
NEH2(0)	4.89	5.22	3.37	7.75	7.52	6.85	7.92	7.52	5.52	7.61	5.24	4.32	6.14
dNEH(S1)	5.63	5.35	3.39	8.80	7.73	7.42	8.48	7.72	5.47	7.71	5.16	4.56	6.45
rNEH(S1)	5.22	5.52	3.48	8.99	8.16	7.25	8.11	7.37	6.12	7.60	5.59	4.50	6.49
NEH2(S1)	4.92	5.22	3.29	8.20	7.58	6.89	7.66	7.22	5.43	7.42	5.01	4.38	6.10
dNEH(S2)	5.29	5.33	3.46	8.30	8.11	7.40	8.35	7.69	5.93	7.91	5.44	4.44	6.47
rNEH(S2)	5.34	5.54	3.42	8.51	7.68	7.36	8.19	7.81	6.24	7.71	5.48	4.57	6.49
NEH2(S2)	4.77	5.22	3.34	7.84	7.54	7.01	7.78	7.27	5.77	7.58	5.33	4.35	6.15
dNEH(S3)	5.63	5.35	3.39	8.80	7.73	7.42	8.48	7.72	5.47	7.71	5.16	4.56	6.45
rNEH(S3)	5.22	5.52	3.48	8.99	8.16	7.25	8.11	7.37	6.12	7.60	5.59	4.50	6.49
NEH2(S3)	4.92	5.22	3.29	8.20	7.58	6.89	7.66	7.22	5.43	7.42	5.01	4.38	6.10
dNEH(S4K)	5.46	5.35	3.35	8.74	7.83	7.22	8.31	7.61	5.69	7.99	5.49	4.47	6.47
rNEH(S4K)	5.34	5.54	3.46	8.75	8.08	7.25	8.73	7.41	6.24	7.78	5.40	4.49	6.49
NEH2(S4K)	4.98	5.24	3.25	8.06	7.68	6.82	7.97	7.18	5.68	7.74	5.19	4.31	6.15
dNEH(S4)	5.46	5.33	3.51	8.56	7.69	7.27	7.97	7.75	5.76	7.72	5.26	4.55	6.40
rNEH(S4)	5.23	5.52	3.44	8.31	7.73	7.19	8.32	7.61	6.09	7.84	5.48	4.59	6.45
NEH2(S4)	4.82	5.22	3.37	7.75	7.38	6.82	7.71	7.39	5.29	7.43	5.07	4.44	**6.06**

Table 6
ARPD obtained with each tie-breaking method for step 2 and with S4 for step 1, on Taillard's instances.

Method	**0**	**0**	**0**	**1**	**1**	**1**	**2**	**2**	**2**	**3**	**3**	**3**
n × ***m***	**NEH**	**rNEH**	**NEH2**	**dNEH**	**rNEH**	**NEH2**	**dNEH**	**rNEH**	**NEH2**	**dNEH**	**rNEH**	**NEH2**
20 × 5	5.46	5.23	4.82	5.37	5.36	4.93	5.12	5.12	5.12	5.29	5.34	4.77
20 × 10	5.33	5.52	5.22	5.49	5.61	5.26	5.52	5.52	5.52	5.33	5.54	5.22
20 × 20	3.51	3.44	3.37	3.43	3.45	3.43	3.44	3.44	3.44	3.46	3.42	3.34
50 × 5	8.56	8.31	7.75	8.66	8.08	7.80	8.34	8.34	8.34	8.30	8.51	7.84
50 × 10	7.69	7.73	7.38	7.72	7.68	7.27	7.83	7.83	7.83	8.11	7.68	7.54
50 × 20	7.27	7.19	6.82	7.36	7.25	7.18	7.03	7.03	7.03	7.40	7.36	7.01
100 × 5	7.97	8.32	7.71	8.17	8.46	7.94	8.64	8.82	8.64	8.35	8.19	7.78
100 × 10	7.75	7.61	7.39	7.47	7.70	7.21	7.53	7.53	7.53	7.69	7.81	7.27
100 × 20	5.76	6.09	5.29	6.18	5.79	5.57	6.12	6.12	6.12	5.93	6.24	5.77
200 × 10	7.72	7.84	7.43	7.70	7.67	7.34	7.76	7.76	7.76	7.91	7.71	7.58
200 × 20	5.26	5.48	5.07	5.43	5.45	5.23	5.43	5.43	5.43	5.44	5.48	5.33
500 × 20	4.55	4.59	4.44	4.39	4.46	4.26	4.50	4.58	4.50	4.44	4.57	4.35
All	6.40	6.45	**6.06**	6.45	6.42	6.12	6.44	6.46	6.44	6.47	6.49	6.15

no tie-breaking method is used (i.e. if two positions lead to the same makespan, the first position is taken); "1" when the Ribas et al. [15] method is implemented; "2" for the Kalczynski and Kamburowski [13] strategy; and "3" for the method proposed by Dong et al. [14]. Each of these procedures was run on the direct and reverse instances. The corresponding ARPD of each set is shown in the dNEH columns and rNEH columns, respectively, and the better of the two values is retained in the NEH2 columns. In general, as shown in Tables 6 and 7, tie-breaking behaviour is not the same for direct and reverse instances. Only the method proposed by Kalczynski and Kamburowski [13] gives similar results for either instance. Moreover, as noted above, the improvement achieved by using the reversibility property is greater than the improvement achieved by any of these methods.

A comparison of the values shown in Tables 6 and 7 shows that, on average, the best results are obtained when S4 is used in step 1. In particular, the overall ARPD indicated in the NEH2 columns shows that the best-performing procedure is to use NEH2 with S4 to break ties in step 1 and not to use any of these tie-breaking procedures for step 2.

Next, a second test was carried out in order to analyse the effect of the different initial ordering procedures (LPT, MM and PF) in the solution obtained by the NEH-based algorithms (NEH, MME and PFE, respectively). Since the PF ordering rule does not define the first job to be considered, we chose the job with the smallest processing time. In light of the results obtained in the previous analysis, we implemented NEH with the tie-breaking method S4 used in step 1. Table 8 shows the ARPD for each set of problem and procedure. In line with the notation used above, columns with "d" before the name indicate that the procedure was applied on the direct instance, columns with "r" before the name indicate that the procedure was applied on the reverse instance, and columns with a "2" after the name retain the better of the two solutions.

Table 8 shows that the reversibility property improves the obtained results by 5–7%. In most instances, the best results are obtained when the initial order of jobs is determined with the MM procedure. Only in the set of instances with 20 jobs does the PF procedure lead to better results. MME2 is, on average, 18% better than NEH2.

Finally, although MME2 has been shown to be the best of these procedures, it is advisable to analyse the computational time required by each one. Table 9 shows that all procedures require a similar CPU time. Therefore, because on average MME2 obtains

Table 7
ARPD obtained with each tie-breaking method for step 2 and without any tie-breaking method for step 1, on Taillard's instances.

Method	0	0	0	1	1	1	2	2	2	3	3	3
$n \times m$	NEH	rNEH	NEH2	dNEH	rNEH	NEH2	dNEH	rNEH	NEH2	dNEH	rNEH	NEH2
20 × 5	5.58	5.25	4.89	5.60	5.59	5.16	5.24	5.24	5.24	5.32	5.14	4.99
20 × 10	5.33	5.52	5.22	5.49	5.61	5.26	5.52	5.52	5.52	5.28	5.29	5.19
20 × 20	3.46	3.48	3.37	3.39	3.40	3.39	3.39	3.39	3.39	3.37	3.34	3.29
50 × 5	8.59	8.60	7.75	8.34	8.59	7.95	8.14	8.14	8.14	8.58	8.51	8.21
50 × 10	7.79	7.84	7.52	7.55	7.73	7.29	7.37	7.37	7.37	7.88	7.70	7.56
50 × 20	7.34	7.22	6.85	7.44	7.23	7.14	7.17	7.17	7.17	7.25	7.28	7.09
100 × 5	8.34	8.20	7.92	8.43	8.20	7.85	8.60	8.60	8.60	8.41	8.29	8.00
100 × 10	8.04	7.66	7.52	7.58	7.47	7.27	7.81	7.81	7.81	7.81	7.56	7.34
100 × 20	5.62	6.18	5.52	6.13	5.83	5.66	6.08	6.08	6.08	6.00	6.08	5.85
200 × 10	7.83	7.82	7.61	7.61	7.78	7.31	7.47	7.49	7.47	8.23	7.59	7.52
200 × 20	5.65	5.49	5.24	5.14	5.49	5.05	5.20	5.20	5.20	5.31	5.32	5.15
500 × 20	4.37	4.70	4.32	4.58	4.35	4.30	4.46	4.46	4.46	4.61	4.61	4.43
All	6.49	6.50	**6.14**	6.44	6.44	**6.14**	6.37	6.37	6.37	6.50	6.39	6.22

Table 8
ARPD obtained by each initial ordering procedure on Taillard's benchmark.

$n \times m$	d NEH	rNEH	NEH2	d MME	r MME	MME2	d PFE	r PFE	PFE2
20 × 5	5.46	5.23	4.82	5.66	5.99	5.14	5.82	5.23	4.90
20 × 10	5.33	5.52	5.22	5.49	5.04	4.69	5.06	5.70	4.57
20 × 20	3.51	3.44	3.37	3.35	4.39	3.34	3.29	4.05	3.00
50 × 5	8.56	8.31	7.75	6.10	6.96	5.81	8.51	8.45	7.84
50 × 10	7.69	7.73	7.38	6.71	6.95	6.30	7.21	7.09	6.52
50 × 20	7.27	7.19	6.82	5.39	5.82	5.06	5.67	5.67	5.29
100 × 5	7.97	8.32	7.71	6.74	6.82	6.53	7.69	7.96	7.46
100 × 10	7.75	7.61	7.39	5.86	6.21	5.81	6.86	6.95	6.44
100 × 20	5.76	6.09	5.29	4.50	4.83	4.34	4.84	5.03	4.67
200 × 10	7.72	7.84	7.43	6.07	6.00	5.85	7.41	7.38	7.10
200 × 20	5.26	5.48	5.07	4.10	3.95	3.80	4.69	4.65	4.42
500 × 20	4.55	4.59	4.44	3.04	3.05	2.94	3.92	4.19	3.88
All	6.40	6.45	6.06	5.25	5.50	**4.97**	5.91	6.03	5.51

Table 9
Average CPU time, in seconds.

$n \times m$	NEH2	MME2	PFE2
20 × 5	0.00	0.00	0.00
20 × 10	0.00	0.00	0.01
20 × 20	0.00	0.01	0.01
50 × 5	0.01	0.01	0.01
50 × 10	0.01	0.01	0.02
50 × 20	0.02	0.03	0.03
100 × 5	0.03	0.03	0.03
100 × 10	0.04	0.05	0.07
100 × 20	0.09	0.09	0.13
200 × 10	0.18	0.20	0.27
200 × 20	0.36	0.40	0.51
500 × 20	2.29	2.34	3.16

Table 10
ARPD on Taillard instances for each algorithm.

$n \times m$	IG1	IG2	HDDE
20 × 5	0.39	0.46	1.49
20 × 10	0.48	0.62	1.53
20 × 20	0.31	0.32	1.23
50 × 5	2.71	2.99	5.69
50 × 10	3.24	3.23	5.63
50 × 20	2.88	2.54	5.04
100 × 5	3.82	3.56	7.22
100 × 10	3.34	3.48	6.67
100 × 50	3.03	2.82	4.41
200 × 10	3.85	3.63	6.91
200 × 20	2.31	2.20	4.34
500 × 20	1.32	1.33	3.93
Overall mean	**2.31**	**2.26**	**4.51**

Table 11
Two-way ANOVA test for the comparison of algorithms.

Source	DF	Seq SS	Adj SS	Adj MS	F	p
algorithm	2	473.571	473.571	236.786	891.72	0
dataset	11	862.859	862.859	78.442	295.41	0
*algorithm*dataset*	22	89.705	89.705	4.077	15.36	0
Error	324	86.034	86.034	0.266		
Total	359	1512.169				

better results than the other two procedures, we propose applying MME to the direct and reverse instances and retaining the better of the two solutions as the initial solution procedure for the BFSP.

4.2. Experimental evaluation of the IG algorithm

In this section, we analyse the performance of two variants of the proposed IG algorithm, denoted as IG1 and IG2. The only difference between these two variants is that IG2 uses the revolver pointer in the local search, which makes it possible to examine the usefulness of this tool in the BFSP. Furthermore, we compare the performance of these algorithms with the performance of the HDDE algorithm proposed by Wang et al. [24], which is, to the best of our knowledge, the best-performing algorithm proposed for this problem so far. To compare the algorithms in the same conditions, we re-implemented the HDDE algorithm. All of the algorithms were coded in the same language (QuickBASIC) and run on 120 Taillard instances using the computational time limit, set at $30 \cdot n^2m$, as a stopping criterion in all cases. Due to the randomness of the algorithms, five runs were done for each one. As a performance criterion, we measured the RPD for each instance, as in (2).

Table 10 shows that, on average, IG1 and IG2 perform better than the HDDE algorithms. The statistical analysis of these results was done by means of a two-way ANOVA. The hypotheses were tested by a residual analysis, which showed small departures from normality, mainly due to a low level of skewness and three borderline outliers. However, as noted above, the ANOVA method is robust to violations of this assumption; this fact, together with the clarity of the results, validates the conclusions and makes a deeper analysis unnecessary.

The ANOVA table (Table 11) shows that the algorithm, the dataset and their interaction are highly significant. Fig. 6 clearly shows that the significant difference for algorithms is due to the

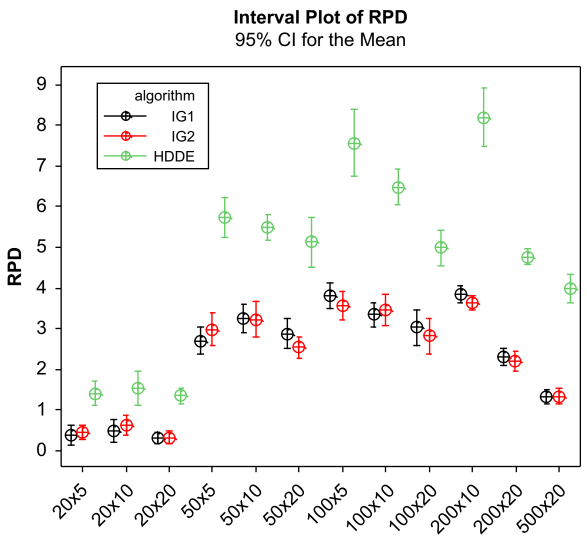

Fig. 6. Interaction plot among algorithms and datasets.

differences between algorithms IG1 and IG2 (which are essentially equal) on the one hand, and HDDE, on the other; this is confirmed by Tukey's multiple comparisons, which give significant *p*-values and are dependent on each explanation given. Therefore, we conclude that the proposed IG algorithm is very competitive.

Moreover, IG1 and IG2 behave in a very similar way. This indicates that, for this problem, randomizing the search path does not necessarily lead to better solutions. The explanation for this can be found in the small number of ties found during the search. If the number of ties were bigger, randomizing the search would mean not always being trapped in the same local minimum; however, since there are so few ties, this tool is not efficient.

5. Conclusions

In this paper, we presented an effective IG algorithm for solving the flowshop scheduling problem with blocking to minimize makespan. The IG procedure proposed makes use of the insertion method of the NEH heuristic, in the construction phase, to reinsert the jobs extracted in the destruction phase, as proposed in [25]. Our goal was to improve the performance of NEH. To do this, first we tested some tie-breaking methods to be used in each of the two phases, namely, the ordering of jobs in accordance with LPT and the insertion phase. In light of the results obtained, we concluded that the use of the reversibility property of the problem is a powerful tool for improving the solutions. Therefore, for constructive procedures, we recommend running the procedure on direct and reverse instances and retaining the better of the two solutions. The performance of NEH, for the blocking problem, was shown to improve when method S4 is used to break ties in the first step and none of the methods proposed for the PFSP are used to break ties in the second step.

Next, making use of the reversibility property, we analysed the effectiveness of using two different procedures, MM and PF, to order the jobs instead of using the LPT rule originally proposed in NEH. The resulting procedures were named MME2 and PFE2, respectively. The computational results showed that MME2 is superior to NEH2. Therefore, we recommend using MME2 as an effective constructive heuristic for the BFSP.

Finally, the proposed IG procedure, which uses MME2 as the initial solution procedure, was tested against the HDDE algorithm proposed by Wang et al. [24]. The comparison revealed that the IG algorithm, despite its simplicity, is very competitive. We reported the new best known solutions found with the proposed IG method for most of the Taillard instances used in the BFSP, which could serve as a basis for comparison in future research.

Future research could focus on developing tie-breaking methods to be used in the MME2 procedure in order to further increase its efficiency. Moreover, due to the simplicity of the proposed procedure, it could be interesting to analyse its performance using other objective criteria and in other problem settings, such as jobshop or hybrid flowshop.

Acknowledgements

The authors wish to thank the referees for their insightful comments on an earlier version of the paper. The research conducted by the authors was partly funded by the Spanish Ministry of Science and Innovation under Grant DPI2007-61371.

References

[1] Hall NG, Sriskandarajah C. A survey of machine scheduling problems with blocking and no wait in process. Operations Research 1996;44(3):510–25.
[2] Grabowski J, Pempera J. Sequencing of jobs in some production system. European Journal of Operational Research 2000;125(3):535–50.
[3] Gong H, Tang L, Duin CW. A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. Computers and Operations Research 2010;37(5):960–9.
[4] Martinez S, Dauzère-Pérès S, Guéret C, Mati Y, Sauer N. Complexity of flowshop scheduling problems with a new blocking constraint. European Journal of Operational Research 2006;169(3):855–64.
[5] Oulamara A, Kovalyov MY, Finke G. Scheduling a no-wait flow shop containing unbounded batching machines. IIE Transactions 2005;37(8):685–96.
[6] Graham RL, Lawler EL, Lenstra JK, Rinnooy Kan AHG. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics 1979;5:287–326.
[7] Reddi SS, Ramamoorthy CV. On the flow-shop sequencing problem with no wait in process. Operations Research Quarterly 1972;23(3):323–31.
[8] Gilmore PC, Lawler EL, Shmoys DB. Well-solved special cases. In: Lawler EL, Lenstra KL, Rinooy Kan AHG, Shmoys DB, editors. The traveling salesman problem: A Guided Tour of Combinatorial Optimization: Wiley; The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization, p. 87–143.
[9] Papadimitriou CH, Kanellakis PC. Flowshop scheduling with limited temporary storage. Journal of the ACM 1980;27(3):533–49.
[10] McCormick ST, Pinedo ML, Shenker S, Wolf B. Sequencing in an assembly line with blocking to minimize cycle time. Operations Research 1989;37:925–36.
[11] Leisten R. Flowshop sequencing problems with limited buffer storage. International Journal of Production Research 1990;28(11):2085–100.
[12] Nawaz M, Enscore Jr EE, Ham I. A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega 1983;11(1):91–5.
[13] Kalczynski PJ, Kamburowski J. An improved NEH heuristic to minimize makespan in permutation flow shops. Computers and Operations Research 2008;35(9):3001–8.
[14] Dong X, Huang H, Chen P. An improved NEH-based heuristic for the permutation flowshop problem. Computers and Operations Research 2008;35(12):3962–8.
[15] Ribas I, Companys R, Tort-Martorell X. Comparing three-step heuristics for the permutation flow shop problem. Computers and Operations Research 2010;37(12):2062–70.
[16] Ronconi DP. A note on constructive heuristics for the flowshop problem with blocking. International Journal of Production Economics 2004;87(1):39–48.
[17] Caraffa V, Ianes S, Bagchi TP, Sriskandarajah C. Minimizing makespan in a blocking flowshop using genetic algorithms. International Journal of Production Economics 2001;70(2):101–15.
[18] Grabowski J, Pempera J. The permutation flow shop problem with blocking. A tabu search approach. Omega 2007;35(3):302–11.
[19] Ronconi DP. A branch-and-bound algorithm to minimize the makespan in a flowshop with blocking. Annals of Operations Research 2005;138(1):53–65.
[20] Wang L, Zhang L, Zheng D. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. Computers and Operations Research 2006;33(10):2960–71.

[21] Liu B, Wang L, Jin Y. An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. Computers and Operations Research 2008;35(9):2791–806.
[22] Qian B, Wang L, Huang DX, Wang X. An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers. International Journal of Production Research 2009;47(1):1–24.
[23] Qian B, Wang L, Huang D, Wang W, Wang X. An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers. Computers and Operations Research 2009;36(1):209–33.
[24] Wang L, Pan Q, Suganthan PN, Wang W, Wang Y. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. Computers and Operations Research 2010;37(3):509–20.
[25] Ruiz R, Stützle T. A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. European Journal of Operational Research 2007;177(3):2033–49.
[26] Companys R. Métodos heurísticos en la resolución del problema del taller mecánico. Estudios Empresariales 1966;5(2):7–18.
[27] Box GEP, Cox DR. An analysis of transformations (with discussions). Journal of the Royal Statistical Society 1964;Series B(26):211–56.
[28] Box GEP, Andersen SL. Permutation theory in derivation of robust criteria and the study of departures from assumptions. Journal of the Royal Statistical Society 1955;Series B(17):1–26.
[29] Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research 1993;64(2):278–85.
[30] Companys R, Ribas I. New insights on the blocking flow shop problem. 2010, p. 1–28, ⟨http://upcommons.upc.edu/e-prints/bitstream/2117/6985/1/DIT block-2.pdf⟩.
[31] Companys R, Mateo M. Different behaviour of a double branch-and-bound algorithm on Fm|prmu|Cmax and Fm|block|Cmax problems. Computers and Operations Research 2007;34(4):938–53.