Hybrid metaheuristic algorithms for the tardiness blocking flow shop problem

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

[a, b] Laboratori d'Organització Industrial, DOE – ETSEIB - Universitat Politècnica de Catalunya, Avda. Diagonal,647, 7th Floor, 08028 Barcelona, Spain

[c] Departament de Estadística E investigación Operativa- ETSEIB - Universitat Politècnica de Catalunya, Avda. Diagonal,647, 6th Floor, 08028 Barcelona, Spain

Abstract

This paper proposes an Iterated Local Search (ILS) procedure and an Iterated Greedy (IG) algorithm, which are both combined with a variable neighbourhood search (VNS), for dealing with the flow shop problem with blocking, in order to minimize the total tardiness of jobs. The structure of both algorithms is very similar, but they differ in the way that the search is diversified in the space of solutions. In the ILS algorithm, the diversification is performed by a perturbation mechanism that takes into account some characteristics of the problem; whereas the perturbation in the IG is performed through a deconstruction and construction phase proposed in the literature that has been proven to be very effective in dealing also with the makespan criterion. Moreover, the algorithms have been tested with three initial solution procedures. The computation of these algorithms when evaluated against an algorithm from the literature has shown their good performance.

Keywords: blocking flow shop, tardiness, iterated greedy algorithm, heuristics

1. Introduction

One of the most studied problems in combinatorial optimization is the permutation flow shop scheduling problem. In a flow shop, there are *n* jobs that have to be processed in *m* machines. All jobs follow the same route in the machines. The processing time of job *i*, $i \in \{1,2,..., n\}$ on machine *j*, $j \in \{1,2,..., m\}$, is $p_{j,i} > 0$. In the traditional version of the problem, it is assumed that there are buffers of infinite capacity between consecutive machines, where jobs, after being processed by the previous machine, can wait until the subsequent machine is available. However, in many industrial systems this supposition cannot be made, since the capacity of buffers is zero, due to the characteristics of the process [1]. Some examples can be found in the production of concrete blocks

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

where storage is not allowed in some stages of the manufacturing process [2]; in the iron and steel industry [3]; in the treatment of industrial waste and the manufacture of metallic parts [4]; 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 [5].

The literature regarding flow shop with blocking is not very extensive. However, in recent years there has been an increase in the number of published papers which deal with the blocking flow shop problem for makespan minimization. For instance, Grabowski and Pempera [6] propose two tabu search (TS) algorithms. Wang et al. [7] propose a hybrid genetic algorithm (HGA), Liu et al [8] an algorithm based on particle swarm optimization (HPSO) and Qian et al. [9] one that is based on differential evolution (DE) that is later adapted to the multicriteria case [10]. Wang et al. [11] propose a hybrid discrete differential evolution (HDDE) and Ribas et al. [12] an iterated greedy (IG) algorithm. Most of these procedures use some variant of the NEH heuristic [13] to generate an initial solution, because this structure has been proven very effective in minimizing makespan for the permutation and blocking flow shop problem. As is well known, this procedure consists of two steps. The first step creates a sequence of jobs according to LPT rule, which is improved upon in the second phase by an insertion procedure. Most of the proposed variants consist of priority rules adapted to the problem's characteristics; they substitute the LPT rule in the first step. For this type of procedure, Ronconi [14] suggests two methods for ordering the jobs: the Profile fitting heuristic [15] and the Min-Max rule. Alternatively, Pan et al [16] suggest some PF variants to combine with the enumeration scheme.

This paper deals with the permutation flow shop problem without buffers between two consecutive machines in order to minimize total tardiness. This problem can be denoted as $Fm \mid block \mid \sum T$, according to the notation proposed by Graham et al. [17]. The Fm| block |$\sum$T can be formulated with the following equations, where $e_{j,k}$ denote the time in which the job [k] starts to be processed on machine j and $c_{j,k}$ is the departure time of job [k] in machine j:

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

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

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

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

$$TT = \sum_{i=1}^{n} \max(c_{m,i} - d_i, 0) \qquad (5)$$

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

If equations (2) and (3) are summarized as (6) and equation (1) and (5) 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.

$$e_{jk}=\max\{c_{j,k-1}; c_{j-1,k}\}. \qquad (6)$$

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

The tardiness criterion has been studied less than the makespan or flowtime criteria, despite the fact that scheduling according to this performance measure helps companies offer a high service level to their customers, which is essential for survival in the market. In particular, to the best of our knowledge, only [18] and [19] dealt with the blocking flow shop problem for total tardiness minimization. Armentano and Ronconi [18] propose a Tabu Search procedure that uses the LBNEH method proposed in [20] to obtain the initial solution. Alternatively, in Ronconi and Henriques [19], a new NEH-based method (FPDNEH) and a GRASP procedure is proposed for this problem.

In this paper we suggest two heuristic procedures: 1) Iterated Greedy (IG) and 2) Iterated Local Search (ILS). Both of them are combined with a Variable Neighbourhood Search (VNS) to minimize the tardiness of scheduled jobs in a flow shop environment with blocking. We have compared these algorithms to a Greedy Randomized Adaptive Search (GRASP) procedure from the literature and to this same GRASP when combined with VNS. The obtained results show that ILS and IG, both with VNS, are efficient procedures for dealing with this problem.

The paper is organized as follows: after this brief introduction, the ILS and IG algorithms are presented in section 2. Section 3 shows computational experiments with both approaches and section 4 summarizes the conclusions.

2. Hybrid metaheuristic approaches

In this paper we propose two metaheuristic approaches for Fm| block |∑T: iterated local search (ILS) and iterated greedy local search (IG). Both have been combined with a variable neighbourhood search (VNS). These two approaches are very close to each other but they differ in the way that they diversify the search. Both procedures can be decomposed into four basic components: the initial

solution procedure, the local search, the perturbation mechanism and the acceptance criterion. The initial solution procedures, the local search and the acceptance criterion are the same in both cases, whereas the perturbation mechanism, which is specific for each algorithm, is presented in their respective sections (2.3 and 2.4).

2.1 Initial Solution

In order to obtain a good initial solution, we have tested some NEH-based heuristics which have been adapted to the specific problem. As has been mentioned before, the structure of the NEH has proven very effective in minimizing the makespan for the permutation and blocking flow shop problem, but the scheme is also effective for the tardiness criterion. In this way, Armentano and Ronconi [20] proposed ordering the jobs with a tardiness lower bound (LB) rule, whereas Ronconi and Henriques [19] showed that better solutions are obtained if jobs are ordered with a *fitting processing times and due dates* (FPD) rule.

We have tested the performance of three ordering rules, which are especially designed for the tardiness criterion. They are the *earliest due date* (EDD), the *slack* (SL) and the FPD rule proposed in [19] and are to be used in ILS and IG to generate an initial solution. These procedures have been named N_{EDD}, N_{SL} and N_{FPD}, respectively.

2.2 Neighbourhood structures

The local search implemented in both algorithms consists of a variable neighbourhood search (VNS). The VNS method has proven very effective in escaping from a local optimum whose main strategy is the systematic change of neighbourhood structure during the search [21,22]. In our procedures, two neighbourhood structures have been considered: swap and insertion. The procedures for exploring each one have been named LS1 and LS2, respectively.

LS1 is based on the swap neighbourhood structure, which considers any two positions $j, k \in \{1, .., n\}$ being $j \neq k$, in a sequence, where the job of position j is exchanged for the job of position k. This neighbourhood can potentially generate $n \cdot (n\text{-}1)/2$ neighbouring solutions for each solution. LS1 is defined as follows: for each job in the sequence, neighbours are generated by swapping a job with all jobs that follow it in the sequence. If the best neighbour (x') is better than the current solution (x), it becomes the new current solution x and the process continues until all jobs have been considered. To

prevent the neighbourhoods from always being explored in the same order, the jobs are selected randomly.

LS2 is based on the insertion neighbourhood structure, where the job at position j is removed from its position and inserted at position $k \in \{1, ..., n\}$ being $k \neq j$, in a sequence. This neighbourhood can potentially generate $n \cdot (n-1)$ neighbouring solutions for each solution. We defined LS2 as follows: for each job in the sequence, neighbours are generated by removing the job from its position and inserting it in all other possible positions. If the best neighbour (x') is better than the current solution (x), it becomes the new current solution x and the process continues until all jobs have been considered. As in LS1, jobs are selected randomly.

```
Procedure VNS
   TT*= TT(x);  x* = x;
     nml1=0
  if random < β then
   indmet = 0
  else indmet = 1
  endif
   do
       nml1=nml1+1;
      TT0 = TT(x)
     if indmet =0 then
       LS1
      else
       LS2
      endif
      if TT(x) < TT0 or nml1=1 then
         indmet = 1 – indmet
       else exit do
      endif
  loop
 end
```

Figure 1. Pseudocode of the variable neighbourhood search implemented.

In our implementation (Figure 1), the percentage of times that one neighbourhood structure is selected first is controlled by parameter β. After exploring the neighbouring solutions of current solution x, the local optimum x' is compared with x. If the solution has improved, x' replaces x and the search continues in the other neighbourhood. This process continues until the current solution is no longer improved. Next, the local optimum x' is compared with the best solution x* in terms of the quality of the solution. If TT(x') is less than TT(x*), then x' replaces x*. If the solution has been improved, there are two possible options: to leave (*shallow* local search) or to restart the current local

search (*in depth* local seach). These options are a two-level factor which are adjusted when tuning the algorithm.

2.3 Iterated Local Search

The Iterated Local Search (ILS) is a metaheuristic procedure that applies a local search to perturbations in the current search point in order to diversify the search. The perturbation has to be enough to escape from a local optimal but should not completely destroy the characteristics of the obtained solution in order to avoid random restarts. The implemented perturbation is specific to the problem being dealt with and it tries to improve the current solution by moving some delayed jobs to advanced positions.

ILs procedure
x:=*Generate Initial Solution*
x*:=x; TT*=TT(x)
repeat
x':= VNS (x)
if TT(x')< TT^* **then** TT^* = TT(x'); x^* = x'; **endif**
if TT^* < TT(x') **and** *random* < α **then**
x' = x^*;
endif
x =*swap_perturbation* (x')
until stopping condition met
end
Figure 2. Pseudocode of ILS algorithm

The pseudocode of ILS is shown in figure 2. Firstly, an initial solution is generated and the process goes to the main body of the algorithm, an operation which is repeated until the stopping condition is met. In this implementation, this condition has been fixed to a CPU time limit. At each iteration, improvements are attempted on the current solution *x* in the VNS module. Next, if the improved solution x' is worse than the best solution found x*, then the current sequence x' is set to the best sequence found x* with a probability 1-α. Next, the solution x' is perturbed by moving some delayed jobs to advanced positions. To select the delayed jobs, the lateness of each job is calculated and kept in an array. This array is ordered in decreasing order of lateness and the first 2·d jobs in the array, i.e. those jobs that are more delayed, are selected as candidates to be moved forward. Among these 2d jobs, *d* are randomly selected and each of these jobs are swapped with another job, also selected randomly, which are in previous positions. The magnitude of the perturbation is fixed by parameter *d*.

The *swap_perturbation* is built from the following steps:

step 1: calculate the lateness of jobs :in array **lat**(*x*')

step 2: order **lat**(*x*') in non-decreasing order of lateness

step 3: set **v**=select the last *d* positions in **lat**(*x*')

step 4: set **w**=select randomly *d*/2 positions of **v**

step 5: set **w'**=for each position in **w** select, randomly, an advanced position

step 6: swap those jobs in position **w** with those in position **w'** in *x* '

2.4 Iterated Greedy Algorithm

The Iterated Greedy Algorithm (IG) is closely related to ILS, the main difference being in the type of perturbation used. The IG generates a sequence of solutions through iterations over a greedy construction heuristic using destruction and construction phases. The destruction phase removes some jobs from the incumbent solution. The construction phase creates a new candidate solution, reconstructing a complete solution by applying a greedy constructive heuristic. Therefore, IG uses a stronger perturbation than ILS.

The pseudocode of the proposed IG is shown in Figure 3. Firstly, an initial solution is generated and the process goes to the main body, which is repeated until the CPU time limit is reached. At each iteration, improvements are attempted on the current solution *x*. If the improved solution *x*' is worse than the best solution found *x**, then the current sequence *x*' is set to the best sequence found *x** with a probability 1-*α*. Next the solution is perturbed by removing *d* jobs, which are randomly chosen, from the current solution *x*. They are then reinserted, one at a time, using the insertion procedure of the NEH, as is done in [23].

```
procedure Iterated Greedy
  x:=Generate Initial Solution
  x*:=x;
  repeat
       x':= VNS (x)
      if TT< TT* then  TT* = TT; x* = x';  endif
      if  TT* < TT and random < α then
        x' = x*;
      endif
       x=decon_perturbation (x')
  until stopping condition met
end

procedure decon_perturbation (x')
      x'' := Ø;
     for i :=1 to d do
```

```
    remove one job of x' randomly and insert it in position i of x'';
  endfor
  for i :=1 to d do
    insert jobs of x'' in x according to the insertion procedure of NEH;
  end for
end
```

Figure 3. pseudocode of the Iterated Greedy algorithm

2.5 Experimental parameter adjustment of the algorithm

Both algorithms have some parameters to be adjusted: *d*, the number of jobs to be considered for swapping in ILS or the number of jobs to be extracted in the perturbation phase in IG; α, the rejection threshold for accepting a worse solution; β, the percentage that the local search begins with LS1; and ML, the strategy used during the search.

The levels chosen for these parameters were:

α: 0.5, 0.75

β: 0.25, 0.5, 0.75

ML: *shallow*, *in depth*

d (ILS)*:* 2,3,4

d (IG)*:* 4, 6, 8

For this test, 480 instances were generated *ad hoc*, 10 instances for each combination of n={20, 50, 100, 200} and m={5, 10, 20} and 4 ranges of due dates, which are named scenarios from now on. The due dates of jobs were uniformly distributed between LB(1-T-R/2) and LB(1-T+R/2) as in [24], where T and R are the tardiness factor of jobs and the dispersion range of due dates, respectively. LB is a lower bound of the C_{max} with unlimited buffer in the flow shop [25]. Therefore, each of the scenarios correspond to a combination of R={0.6, 1.2} and T={ 0.2, 0.4}. Due to the randomness of the improvement procedure, we performed 5 runs per instance. The computation time limit was set to $10 \cdot n^2 \cdot m \cdot 10^{-5}$ seconds. The experiments were carried out on an Intel Core 2 Duo E8400 CPU, with 3GHz and 2GB RAM memory.

To analyze the experimental results obtained, we measured the relative deviation index (RDI), calculated as (8) for each procedure:

where $Heur_{hs}$ is the average of tardiness values obtained by heuristic *h* in 5 runs, in instance *s*, and $Best_s$ and $Worst_s$ are the best and worst solutions obtained for this instance, in any run, among all the combinations of parameters.

The EDA (Exploratory Data Analysis) of the results from the parameter adjustment shows that there are two idiosyncrasies to take into account before proceeding to a formal analysis. The first one is that the obtained RDI values show a non-normal distribution. This is due to the existence of a high concentration of zero values. This has been solved by removing from the analysis all the instances in which RDI values were zero for all combinations of parameters. It is obvious that these instances do not contribute to differentiating between them. For the IG algorithm, 59 instances were removed, leaving 421 to be analyzed. For the ILS, the number is a bit higher: 71 instances were removed, leaving 409 to be analyzed.

The second idiosyncrasy to take into account before proceeding is that, even though RDI is supposed to level out the differences due to the distinct level of difficulty presented by instances, it does not. Figure 4 shows boxplots of RDI (left for ILS and right for IG), stratified by *n* and *m*. It is easy to see that for low values of *n* and *m*, "easy" instances, RDIs are lower than for high values.

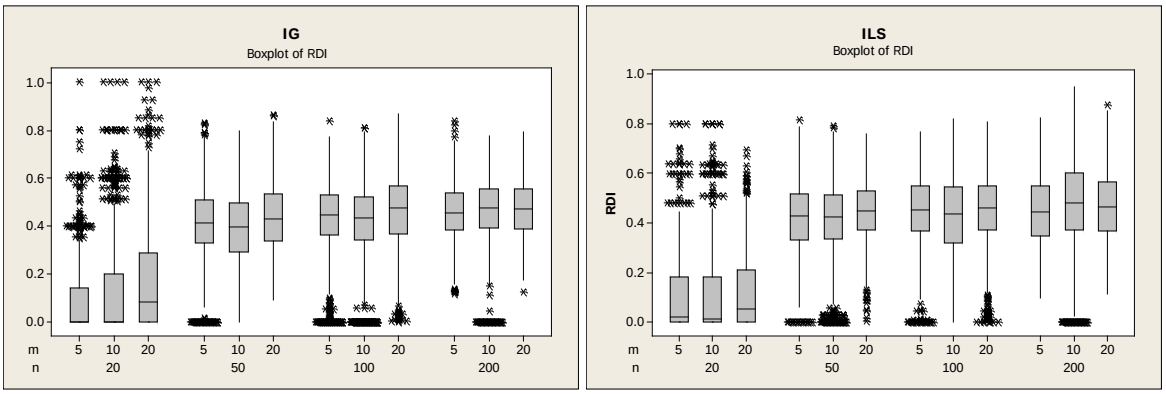


Figure 4. Boxplot of RDI stratified by *n* and *m*

The usual way to remove this variability, so that it doesn't hinder the identification of significant algorithm parameters, is to consider the 480 instances as a blocking variable in the Analysis of Variance. The effect is to compare the 36 algorithm variations (resulting from the combination of all parameters) without interferences from instances differences. The procedure is equivalent to analyze a new variable, let's call it RDI_Blck, generated by subtracting to each RDI the average of the 480 RDI values obtained for each instance by the 36 different algorithm variations. Figure 5 is analogous

to Figure 4 but using RDI_Blck instead of RDI. It is clear that differences due to instances have disappeared.

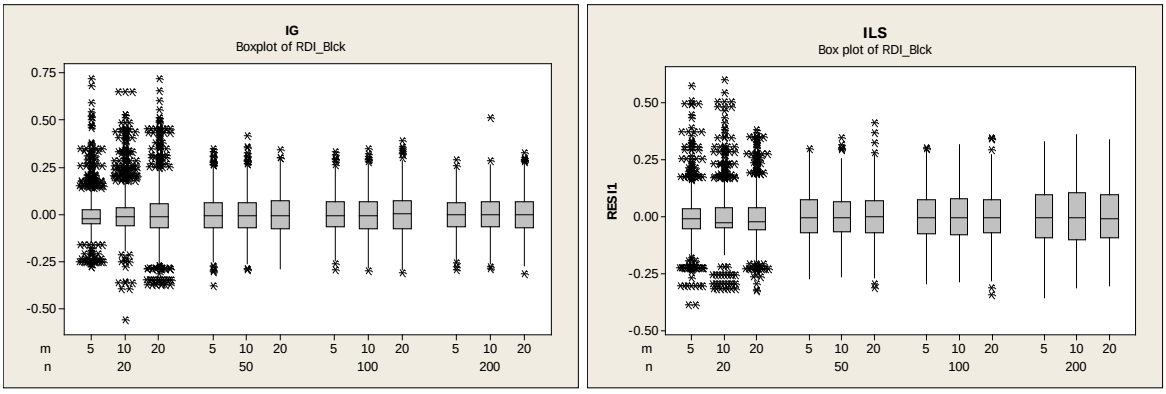


Figure 5. Boxplot of RDI_Blck stratified by *n* and *m*

In consequence, an Analysis of Variance for each algorithm was conducted, including as factors the four algorithm parameters (α, β, d and ML) and their two factor interactions plus the blocking variable (instances). The analysis of residuals shows no relevant violations of the ANOVA assumptions. The *p*-values showing the significance level for each factor are summarized for the two algorithms in Table 1. The blocking factor (instances), not included in the table, is obviously highly significant in both cases, as was to be expected from the discussion in the paragraph above.

Factors	**IG**	**ILS**
β	0.000	0.000
α	0.016	0.561
d	0.000	0.000
ML	0.000	0.000
$\beta*\alpha$	0.400	0.095
$\beta*d$	0.592	0.158
$\beta*ML$	0.998	0.490
$\alpha*d$	0.108	0.033
$\alpha*ML$	0.136	0.039
$d*ML$	0.000	0.778

Table 1. *p*-values from the Analysis of Variance for factors of IG and ILS

Regarding the algorithm parameters, β, d and ML are all highly significant for both algorithms, and α is slightly significant in both cases. This is a primary effect in IG and results from the interactions with d and ML in ILS. The interaction between d and ML is also significant in IG. A graphical analysis of these three interactions clearly shows that the size of their effect is negligible in light of the size of the main effects; in other words, it is not necessary to take them into account in order to find the best levels for β, α, d and ML. To determine the best level for each parameter, we have computed 95% confidence intervals for each parameter level after removing all other influences; that is, by using the residuals from the model that include the remaining parameters plus the blocking

factor. The results are shown in Figure 6 for ILS and Figure 7 for IG, and the best levels for the two algorithms are summarized in Table 2

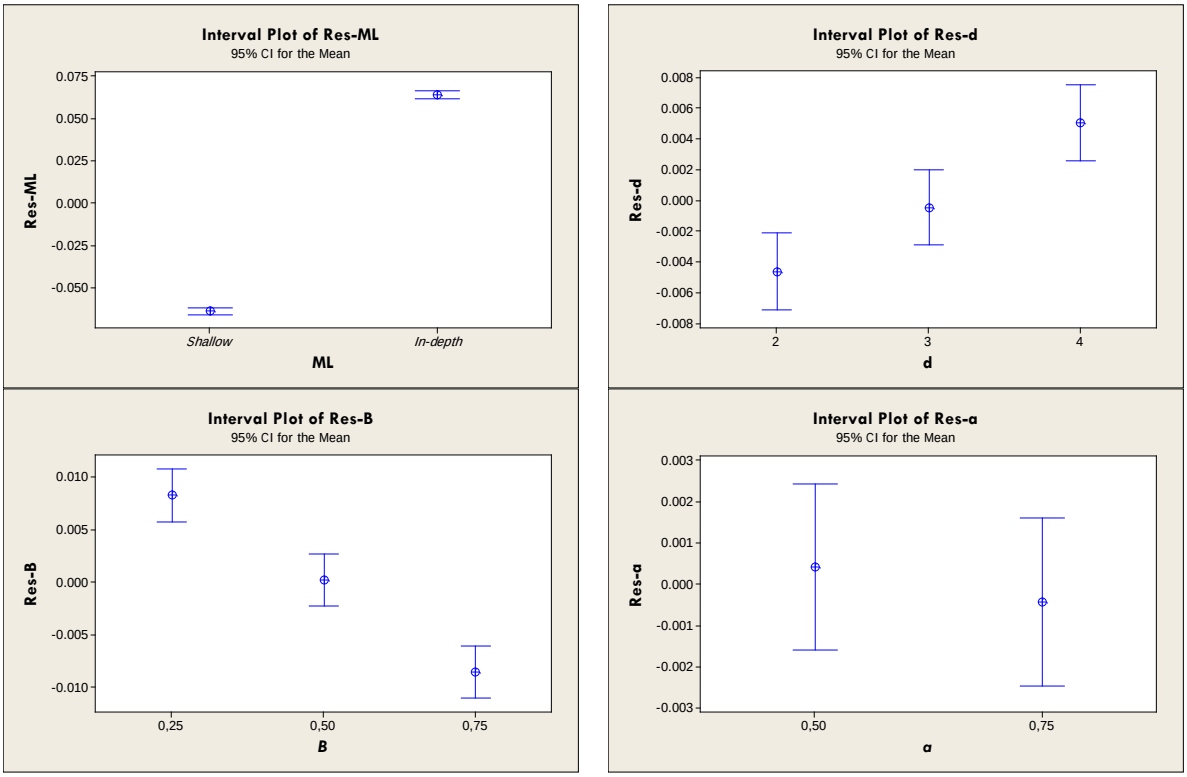
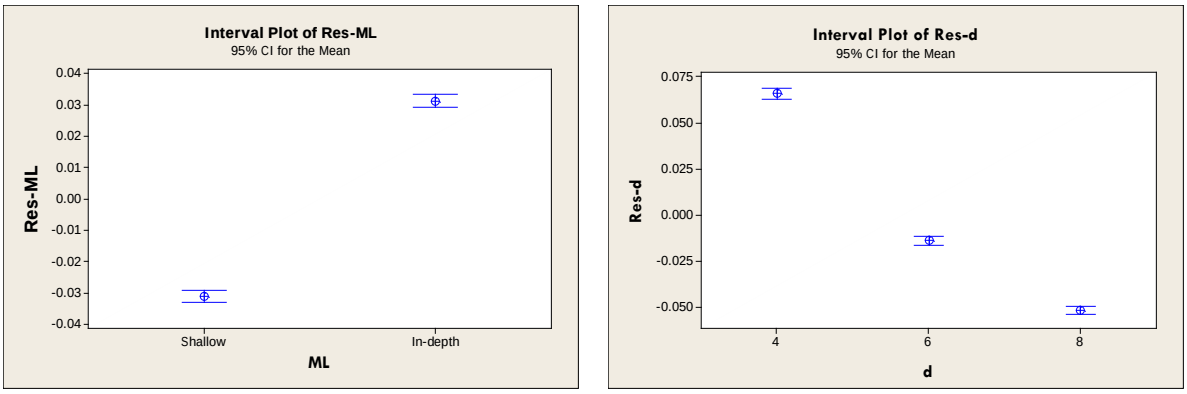


Figure 6. ILS Algorithm. 95% confidence intervals for each parameter once the influence of the others and the instances have been taken out.



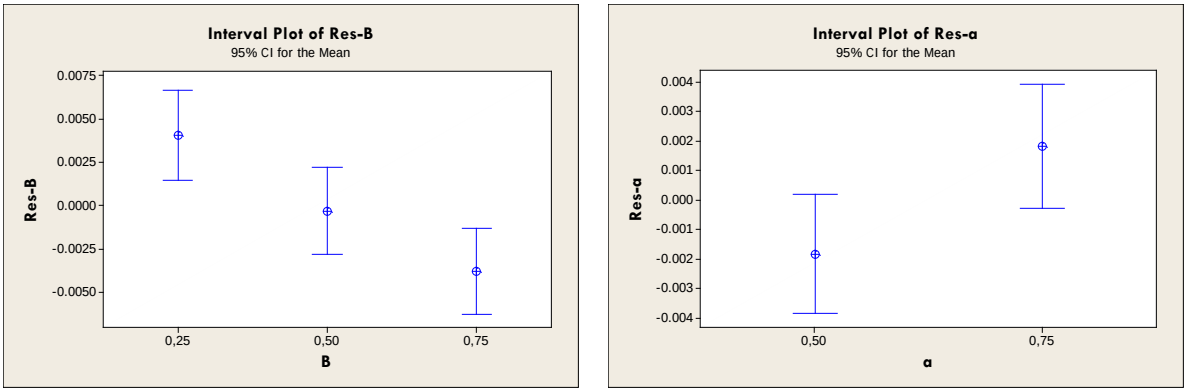


Figure 7. IG Algorithm. 95% confidence intervals for each parameter once the influence of the others and the instances have been taken out.

Parameter	**IG**	**ILS**
α	0.5	Indifferent
β	0.75	0.75
ML	*shallow*	*shallow*
d	8	2

Table 2. Best level of algorithm parameters

Notice that for the ILS algorithm, the main effect of α is not significant because the three slightly significant interactions compensate each other.

3. Computational evaluation

In this section the performance of the proposed procedures has been tested against three test beds, two of them from the literature and the other one has been created *ad hoc*. The first test bed taken from the literature was the first 110 instances of Ronconi [19] (11 sets of 10 instances each), generated according to Taillard [25]. The due dates of the jobs were generated as in [24]. For each instance, four scenarios were created in relation to the following T and R values shown in columns 1 and 2 of Table 3:

Scenario	**Ronconi test bed**		**Taillard test bed**		**RCT test bed**	
	T	**R**	**T**	**R**	**T**	**R**
1	0.2	0.6	0.36	0.48	0.2	0.6
2	0.2	1.2	0.36	0.96	0.2	1.2
3	0.4	0.6	0.52	0.48	0.4	0.6
4	0.4	1.2	0.52	0.96	0.4	1.2

Table 3. Definition of T and R value for each scenario and test bed

The second test bed was the 11 first sets of Taillard's well-known instances [25] that, even though they were created for the permutation flow shop to minimize makespan, they are also used for the blocking flow shop to minimize the maximum completion time of jobs as well as to minimize the tardiness criterion. As the original instances do not have due dates, we have generated them according to [24]. But, in this case, the lower bound of C_{max} has been substituted by an upper bound, which is the best known solution for each instance available in [26], because these values are tighter to the optimal solution than that calculated by the lower bound procedure. As these solutions are about 20% tighter than the lower bound, we have applied an impact equivalent to this percentage to the *T* and *R* values of each scenario (see column 3 and 4 of table 1).

Finally, because we observed different behaviours of the algorithms in these two test beds, even though they were created in the same way, we decided to create a new set of instances that were also generated according to [25], which we named RCT. This test bed contains 11 sets of 10 instances. The size of each test (*n* x *m*) is the same as in the other test beds. The processing times of jobs were uniformly distributed between [1, 99] and due dates were created according to [24]. The *T* and *R* values for each scenario are the same as in [19] (see columns 5 and 6 of table 1).

The comparison between procedures is made with the RDI index defined as (8); but in this case, the $Best_s$ and $Worst_s$ solutions are the best and worst solutions obtained from among all methods included in the comparison. Since this index takes values between 0 and 1, an index that is close to 0 indicates that the procedure is a good method for minimizing the tardiness criterion, whereas an index close to 1 indicates that some other procedure among those being compared is better. Notice that if the best and worst solutions are the same, it means that all these procedures have obtained the same solution; therefore the index associated to all procedures is set to 0. This measure is a good index for comparing several methods between them, but its value depends on the procedures being considered in the comparison. Therefore, the value of the index associated to each procedure has to be recalculated each time that a new procedure is included in the comparison, because the best and worst solutions of the instances can change.

3.1 Analysis of initial solution procedures

We have evaluated the performance of three initial solution procedures, N_{EDD}, N_{SL} and N_{FPD}, in order to chose the procedure which leads to better results. This test has been carried out with the three test beds. In order to know the behaviour of each method, the average RDI in this analysis has been

calculated for each set in each scenario, as well as the overall average without considering scenarios,. The results are shown in Tables 4-6.

		Scenario1			Scenario 2			Scenario 3			Scenario 4			All scenarios		
n	m	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}
20	5	0.42	0.44	0.65	0.66	0.20	0.27	0.57	0.44	0.43	0.51	0.74	0.27	0.54	0.45	0.40
20	10	0.43	0.56	0.38	0.60	0.65	0.24	0.42	0.71	0.52	0.42	0.67	0.45	0.47	0.65	0.40
20	20	0.64	0.63	0.28	0.35	0.37	0.67	0.28	0.39	0.84	0.48	0.43	0.70	0.44	0.46	0.62
50	5	0.56	0.66	0.29	0.43	0.35	0.25	0.40	0.57	0.42	0.40	0.59	0.49	0.45	0.54	0.36
50	10	0.36	0.41	0.61	0.94	0.19	0.38	0.51	0.55	0.47	0.30	0.63	0.51	0.53	0.44	0.49
50	20	0.45	0.65	0.36	0.64	0.29	0.36	0.53	0.32	0.69	0.43	0.31	0.69	0.51	0.39	0.53
100	5	0.51	0.53	0.36	0.41	0.34	0.20	0.73	0.62	0.23	0.47	0.38	0.71	0.53	0.47	0.38
100	10	0.42	0.68	0.37	0.38	0.60	0.60	0.73	0.62	0.23	0.52	0.47	0.44	0.51	0.59	0.41
100	20	0.36	0.50	0.71	0.48	0.57	0.53	0.75	0.25	0.51	0.48	0.57	0.53	0.52	0.47	0.57
200	10	0.44	0.55	0.56	0.57	0.47	0.42	0.45	0.85	0.28	0.60	0.27	0.62	0.51	0.53	0.47
200	20	0.30	0.52	0.58	0.57	0.50	0.47	0.41	0.52	0.60	0.25	0.49	0.75	0.38	0.51	0.60
Average		**0.45**	0.56	0.47	0.55	0.41	**0.40**	0.53	0.53	**0.48**	**0.44**	0.50	0.56	**0.44**	0.50	0.56

Table 4. Average RDI for each NEH method with Ronconi's intances by scenario.

The results obtained from the Ronconi test bed (Table 2) indicate that the EDD rule is slightly better for scenarios 1 and 4, whereas the FPD is slightly better for scenarios 2 and 3. But considering the overall average between scenarios, it seems advisable to use EDD for sequencing jobs before the insertion phase. However, the differences between them are not so very great that we cannot ensure that one is better than another for generating the initial solution

In the Taillard test bed (Table 5), the obtained results indicate that EDD is better for scenarios 1 and 3 and Slack is slightly better for scenarios 2 and 4. In this test bed the FPD rule is less effective than in the previous one; instead, the behaviour of the slack has improved. But as can be observed in the last columns, the overall average RDI of each procedure is around 0.5, which indicate that sometimes the best solution is obtained by one of them but at other times another procedure is best.

		Scenario1			Scenario 2			Scenario 3			Scenario 4			All scenarios		
n	m	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}
20	5	0.24	0.52	0.69	0.69	0.17	0.49	0.67	0.50	0.49	0.61	0.543	0.422	0.55	0.43	0.52
20	10	0.39	0.56	0.61	0.51	0.44	0.61	0.62	0.33	0.35	0.60	0.53	0.36	0.53	0.46	0.48
20	20	0.40	0.51	0.55	0.43	0.45	0.53	0.33	0.71	0.57	0.58	0.71	0.27	0.43	0.59	0.48
50	5	0.60	0.42	0.43	0.39	0.51	0.48	0.27	0.42	0.70	0.56	0.57	0.52	0.46	0.48	0.53
50	10	0.45	0.34	0.84	0.43	0.56	0.54	0.34	0.46	0.76	0.62	0.35	0.63	0.46	0.43	0.69
50	20	0.49	0.62	0.50	0.77	0.12	0.59	0.30	0.52	0.66	0.58	0.46	0.40	0.53	0.43	0.54
100	5	0.18	0.75	0.63	0.75	0.66	0.28	0.60	0.42	0.50	0.54	0.44	0.46	0.52	0.57	0.47
100	10	0.48	0.64	0.55	0.33	0.50	0.68	0.43	0.52	0.49	0.53	0.55	0.64	0.44	0.55	0.59
100	20	0.33	0.35	0.86	0.62	0.70	0.11	0.56	0.65	0.24	0.39	0.49	0.56	0.47	0.55	0.44
200	10	0.31	0.55	0.58	0.64	0.29	0.64	0.25	0.65	0.78	0.48	0.43	0.57	0.42	0.48	0.64
200	20	0.32	0.56	0.49	0.50	0.44	0.55	0.33	0.60	0.57	0.53	0.28	0.61	0.42	0.47	0.55
Average		**0.38**	0.53	0.61	0.55	**0.44**	0.50	**0.43**	0.53	0.56	0.55	**0.49**	0.50	**0.48**	0.49	0.54

Table 5. Average RDI for each NEH method with Taillard's intances by scenario.

Finally, with the RCT test bed (table 6), the best results are always obtained when the Slack rule is used. Notice that now the difference between procedures is greater than in the other tests.

		Scenario 1			Scenario 2			Scenario 3			Scenario			All scenarios		
n	m	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}	N_{EDD}	N_{SL}	N_{FPD}
20	5	0.63	0.56	0.47	0.51	0.43	0.50	0.46	0.69	0.26	0.70	0.60	0.37	0.46	0.46	0.32
20	10	0.54	0.50	0.55	0.78	0.27	0.61	0.37	0.53	0.57	0.31	0.47	0.59	0.40	0.35	0.46
20	20	0.54	0.37	0.40	0.44	0.25	0.69	0.61	0.22	0.64	0.51	0.60	0.44	0.42	0.29	0.43
50	5	0.52	0.53	0.36	0.57	0.42	0.50	0.59	0.20	0.54	0.69	0.43	0.25	0.47	0.32	0.33
50	10	0.69	0.27	0.54	0.61	0.25	0.56	0.70	0.51	0.35	0.48	0.42	0.46	0.50	0.29	0.38
50	20	0.55	0.59	0.46	0.58	0.46	0.56	0.65	0.27	0.59	0.80	0.35	0.42	0.52	0.33	0.41
100	5	0.53	0.49	0.52	0.35	0.41	0.60	0.35	0.49	0.67	0.83	0.37	0.48	0.41	0.35	0.45
100	10	0.33	0.47	0.68	0.40	0.37	0.52	0.31	0.39	0.74	0.63	0.30	0.58	0.33	0.31	0.50
100	20	0.47	0.36	0.85	0.59	0.58	0.27	0.39	0.44	0.77	0.58	0.30	0.50	0.41	0.34	0.48
200	5	0.61	0.51	0.34	0.55	0.46	0.28	0.47	0.73	0.28	0.34	0.34	0.74	0.39	0.41	0.33
200	10	0.42	0.58	0.41	0.61	0.47	0.46	0.64	0.25	0.70	0.64	0.29	0.53	0.46	0.32	0.42
200	20	0.40	0.35	0.66	0.37	0.44	0.71	0.31	0.54	0.45	0.66	0.46	0.39	0.35	0.36	0.44
Average		0.52	**0.46**	0.52	0.53	**0.40**	0.52	0.49	**0.44**	0.55	0.60	**0.41**	0.48	0.43	**0.34**	0.41

Table 6. Average RDI for each NEH method with RCT intances by scenario.

From the results obtained in these tests, we cannot discard any of these priority rules to generate the initial solution for the IG or ILS procedure, because the differences between them are not enough to know which one can lead to better results. Therefore, in the next section, the performance of each algorithm is analyzed with each of these three initial solution methods.

3.2 Analysis of proposed heuristics

In this section the effectiveness of ILS and IG combined with VNS has been analyzed. Their performance has been compared to the GRASP procedure proposed in [19]. As is well known, a GRASP metaheurisitc is a multi-start procedure composed of construction and local search phases. The construction phase builds a feasible solution which is improved upon by searching its neighbourhood until a local minimum is reached. The construction phase is integrated by the FPD and a restricted candidate list (RCL) constructed by the method $value\_based\_scheme_1$ with p=0.35. The solution is improved upon by the insertion phase of NEH before going into the local search. The local search uses the insertion move and the *first improved strategy*, i.e., the current solution is replaced by the first improved solution. The reader can find more details of this procedure in [19].

In terms of RDI index, the algorithms have been compared to the three sets of test beds considered. In this case, the RDI $Best_s$ and $Worst_s$ are the best and worst solutions obtained in instance s by each version of the algorithms considered in the comparison of any of the 5 runs.

In a preliminary test, we observed a great advantage in our procedures when compared to GRASP. Since the RDI index varies according to the procedures in comparison, if one algorithm is much worse than the others, the range between best and worst solutions increases. This mitigates the difference between algorithms with a similar performance, as is the case with the implemented ILS and IG. Therefore, a first analysis was made in order to compare only IG and ILS before comparing them to other algorithms. In the results analysis of each test bed, we observed a high concentration of zero values which was again solved by removing the instances in which RDI values were zero in all runs of both algorithms. Additionally, to mitigate the difference between instances we considered, in each test bed, the instances as a blocking variable in the Analysis of Variance.

For the Ronconi test bed, 117 out of 440 instances were removed. The ANOVA results shown in table 7 indicate that the differences between procedures are significant, whereas the initial solution method and its interaction with the procedures are not significant. This indicates that the method used is irrelevant. This fact can be seen in the interval plot of RDI_Blck (Figure 8), where the mean and 95% confidence interval for each algorithm and initial solution procedure is shown. Notice that IG has a better performance than ILS.

Source	DF	SS	MS	F	P
Algorithm	1	3.284	3.284	165.23	0.000
Initial solution	2	0.012	0.006	0.32	0.723
*Algorithm*Initial solution*	2	0.044	0.022	1.12	0.326
instances	332	39.310	0.118	5.96	0.000
Error	1660	32.999	0.019		
Total	1997	75.652			

Table 7. Analysis of Variance for RDI with Ronconi's instances.

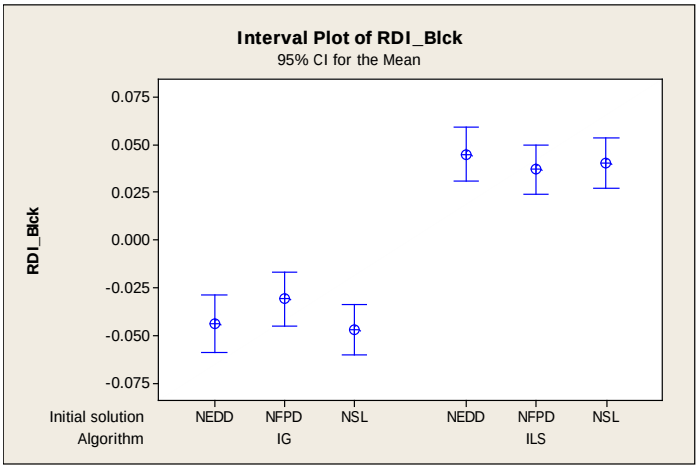


Figure 8. Interval plot for RDI_blck with Ronconi's instances

The same analysis has been performed with Taillard's and RCT instances. In the Taillard test bed, 82 instances were removed whereas in the RCT 111 instances were discarded. Tables 6 and 7 show the ANOVA results, respectively.

Source	DF	SS	MS	F	P
Algorithm	1	0.214	0.214	10.90	0.001
Initial solution	2	1.399	0.699	35.60	0.000
*Algorithm*Initial solution*	2	0.349	0.174	8.90	0.000
instances	357	49.316	0.138	7.03	0.000
Error	1785	35.096	0.019		
Total	2147	86.377			

Table 8. Analysis of Variance for RDI with Taillard's instances.

From the results shown in table 8, one can see that, in this case, not only the algorithm is significant but also the initial solution procedure and the interaction between them. In Figure 9 we can observe the different behaviour of algorithms with each initial solution method. Moreover, it can be seen that these differences are due to the N_{FPD} method, because both algorithms have a similar performance when the initial solution is generated by N_{SL} or N_{EDD}.

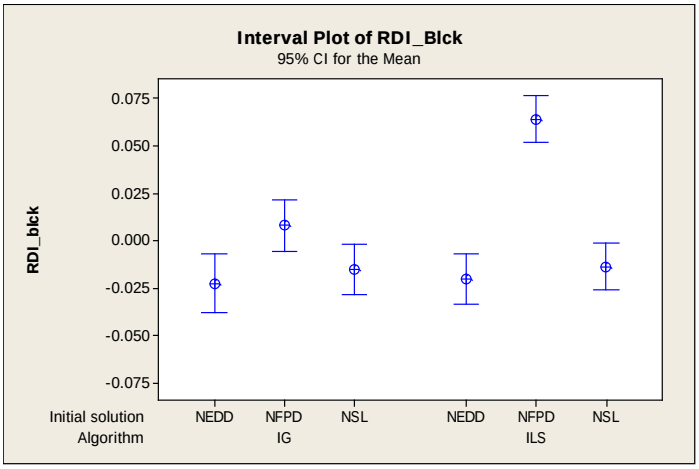


Figure 9. Interval plot for RDI_blck with Taillard's instances

Lastly, with the RCT test bed, the *p*-values in table 9 show that the algorithm, initial solutions and their interaction are significant; the results are similar for the Taillard instances. These differences are made evident in Figure 10, where one can observe that the best results are obtained with ILS when the N_{EDD} is used to generate an initial solution.

Source	DF	SS	MS	F	P
Algorithm	1	0.078	0.078	4.15	0.042
Initial solution	2	0.837	0.418	22.03	0.000
*Algorithm*Initial solution*	2	0.583	0.291	15.35	0.000

instances	328	40.915	0.124	6.57	0.000
Error	1640	31.156	0.019		
Total	1973	73.571			

Table 9. Analysis of Variance for RDI with RCT instances.

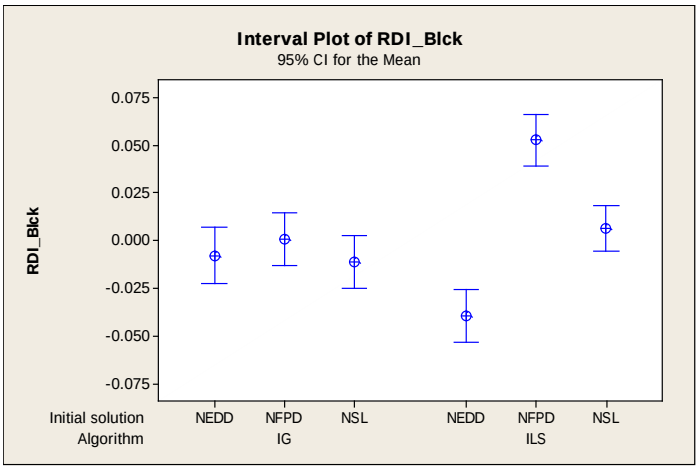


Figure 10. Interval plot for RDI_blck with RCT instances

Notice that the behaviour of algorithms in the Taillard and RCT test beds is very similar, but totally different from that in the Ronconi test bed. If we had only considered the Taillard and RCT instances, we would have concluded that ILS with N_{EDD} has the best performance; whereas if we had only considered the results obtained in the Ronconi test bed, we would have concluded that IG was the best algorithm. Therefore, we cannot say if ILS or IG is better; however, some improvement can be seen in the results obtained from IG, as well as in ILS, when the N_{EDD} is used to generate the initial solution. Therefore, to make the final comparison to an external reference, the GRASP procedure [19], we have selected IG and ILS with N_{EDD} to generate the initial solution. In addition to the original version of the GRASP, named GRASP1 from now on, we have implemented four variants of it. The first variant, that we have named GRASP2, is very close to the original, since we have omitted only the step in which the solution is improved upon by the insertion phase of NEH before using the local search. The other variants are a GRASP with EDD, SL and FPD rules, respectively, in which the local search has been substituted by the variable local search implemented in the IG and ILS algorithms, named GEDDVNS, GFPDVNS and GSLVNS, respectively.

In order to carry out a fair comparison between procedures, all algorithms were encoded in the same language (QuickBASIC) and were tested on the same computer, an Intel Core 2 Duo E8400 CPU, with 3GHz and 2GB RAM memory. The CPU time limit was fixed to $30\, n^2 \cdot m \cdot 10^{-5}$ seconds in all algorithms. As in the previous test, the comparison has been carried out with the RDI measure calculated as in (8).

As in the previous test, we have analyzed the performance of these algorithms in each test bed without taking into account the instances whose RDI value is 0 for the 7 algorithms. Moreover, we have considered the instances as a blocking variable in the Analysis of Variance to diminish the difference between instances. The two-way ANOVA --which considers the algorithms and instances as a factors-- indicates that both factors were significant at a confidence level of 95% (*p*-value= 0.000).

It can be observed in Figures 11-13, which show the interval plot of the RDI blocked by the instances in each test bed, that the results obtained with these three test beds are very similar. It can be seen that IG and ILS outperform the other algorithms. Notice the great improvement in the obtained results when the VNS substitutes the original local search; this indicates the benefits of combining these two procedures. However, little difference can be seen among the results when EDD, SL or FPD are used to generate an initial solution. However, we could say that GSLVNS and GEDDVNS have a similar performance, whereas GFPDVNS is slightly worse.

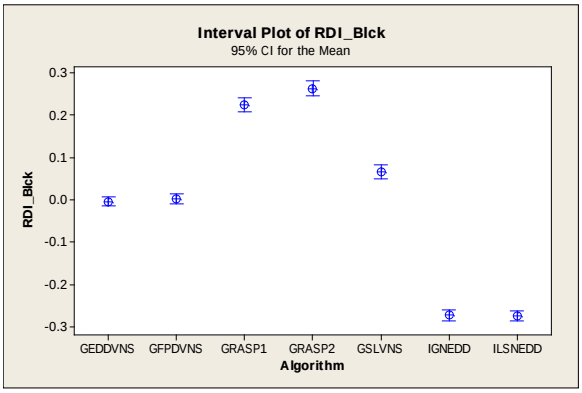


Figure 11. Interval plot with Ronconi's instances

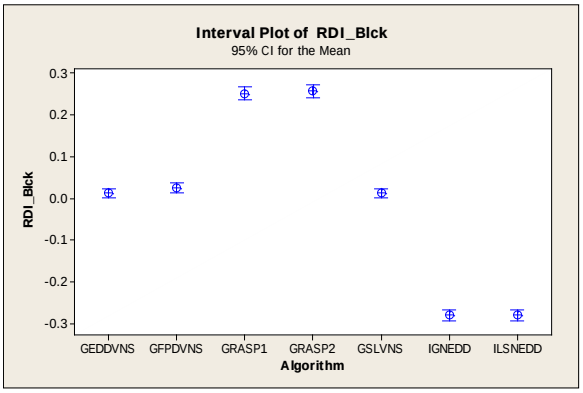


Figure 12. Interval plot for RDI_blck with Taillard's instances

It can also be seen that GRASP1, which uses the insertion phase of NEH to improve the initial sequence, is slightly better than GRASP2. Finally, as we mentioned at the beginning of this section, we cannot observe any difference between IG and ILS because, when all algorithm are compared, the interval between the worst and best solutions increases, which leads to mitigating the small differences between algorithms.

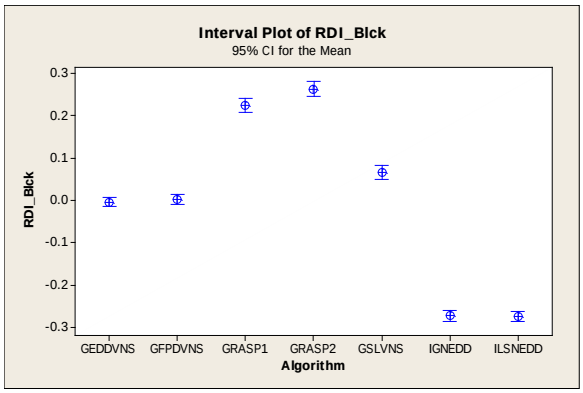


Figure 13. Interval plot for RDI_blck with RCT instances

4. Conclusions

This paper proposes an Iterated Local Search (ILS) procedure and an Iterated Greedy (IG) algorithm, both of which combined with a variable neighbourhood search (VNS), for dealing with the flow shop problem with blocking in order to minimize the total tardiness of jobs. Both procedures are very similar but, normally, the perturbation phase of IG is stronger than in ILS, which allows escaping from a deep local minimum more easily. In the proposed ILS, we have implemented a perturbation mechanism that takes into account some characteristics of the problem; this has allowed guiding the search toward good solutions. In order to improve the algorithms, we have evaluated their performance when three initial solution procedures are used. These procedures have the structure of NEH, in which the LPT rule has been substituted for other rules oriented more toward the tardiness criterion.

The computational evaluation of these algorithms has been carried out on three sets of instances. Two of them were proposed in the literature and the third was created specifically (as were the first two) in order to strengthen the analysis, since we observed in the tests that the algorithms had different behaviour in each of these sets of instances.

The experimental evaluation allowed us to conclude that both algorithms improve their results when N_{EDD} is used. But, the comparison between these two algorithms does not allow us to say if one is better than another, as both have demonstrated a similar performance.

Lastly, both algorithms have been compared to a GRASP proposed in the literature, concluding that both are very efficient for the problem being dealt with. In this test we have shown that the GRASP with VNS is much better than with local search in only one neighbourhood, which allows us to recommend this strategy in the improvement phase of this algorithm.

Future research involving the tardiness criterion could include designing another procedure for comparing algorithms, because the RDI used until now does not provide a good perspective of the algorithms' performance, due to the fact that the RDI's value depends on the algorithms being

considered, as well as the fact that it has to be recalculated each time that an algorithm is included or excluded from the comparison.

Acknowledgments

The authors would like to thank Debora Ronconi for providing us with the GRASP source code.

References

[1] NG Hall, C Sriskandarajah. A survey of machine scheduling problems with blocking and no wait in process, Operations Research. 44 (1996) 510-525.

[2] J Grabowski, J Pempera. Sequencing of jobs in some production system, European Journal of Operational Research. 125 (2000) 535-550.

[3] H Gong, L Tang, CW Duin. A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times, Comput.Oper.Res. 37 (2010) 960-969.

[4] S Martinez, S Dauzère-Pérès, C Guéret, Y Mati, N Sauer. Complexity of flowshop scheduling problems with a new blocking constraint, Eur.J.Oper.Res. 169 (2006) 855-864.

[5] SP Sethi, C Sriskandarajah, G Sorger, J Blazewicz, W Kubiak. Sequencing of parts and robot moves in a robotic cell, International Journal of Flexible Manufacturing Systems. 4 (1992) 331-358.

[6] J Grabowski, J Pempera. The permutation flow shop problem with blocking. A tabu search approach, Omega,. 35 (2007) 302-311.

[7] L Wang, L Zhang, D Zheng. An effective hybrid genetic algorithm for flow shop scheduling with limited buffers, Computers & Operations Research. 33 (2006) 2960-2971.

[8] B Liu, L Wang, Y Jin. An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers, Computers & Operations Research,. 35 (2008) 2791-2806.

[9] B Qian , L Wang, DX Huang, X Wang. An effective hybrid DE-based algorithm for flow shop scheduling with limited buffers, International Journal of Production Research. 47 (2009) 1-24.

[10] B Qian, L Wang, DX Huang, W Wang, X Wang. An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers, Computers & Operations Research. 36 (2009) 209-233.

[11] L Wang, Q Pan, PN Suganthan, W Wang, Y Wang. A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems, Comput.Oper.Res. 37 (2010) 509-520.

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

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

[14] DP Ronconi. A note on constructive heuristics for the flowshop problem with blocking, International Journal of Production Economics,. 87 (2004) 39-48.

[15] ST McCormick, ML Pinedo, S Shenker, B Wolf. Sequencing in an Assembly Line with Blocking to Minimize Cycle Time, Operations Research. 37 (1989) 925-936.

[16] Q Pan, L Wang. Effective heuristics for the blocking flowshop scheduling problem with makespan minimization, Omega. 40 (2012) 218-229.

[17] RL Graham, EL Lawler, JK Lenstra, Rinnooy Kan A.H.G. Optimization and approximation in deterministic sequencing and scheduling: A survey, Annals of Discrete Mathematics. 5 (1979) 287-326.

[18] VA Armentano, DP Ronconi. Minimização do Tempo Total de Atraso no Problema de Flowshop com Buffer Zero através de Busca Tabu. Gestao & Produçao. 7 (2000) 352.

[19] DP Ronconi, LRS Henriques. Some heuristic algorithms for total tardiness minimization in a flowshop with blocking, Omega. 37 (2009) 272-281.

[20] VA Armentano, DP Ronconi. Tabu search for total tardiness minimization in flowshop scheduling problems, Comput.Oper.Res. 26 (1999) 219-235.

[21] P Hansen, N Mladenovic. Variable neighborhood search: Principles and applications, Eur.J.Oper.Res. 130 (2001) 449-467.

[22] P Hansen, N Mladenovic, D Perez-Britos. Variable Neighborhood Decomposition Search, Journal of Heuristics. 7 335-350.

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

[24] CN Potts, LN Van Wassenhove. A decomposition algorithm for the single machine total tardiness problem, Operations Research Letters,. 1 (1982) 177-181.

[25] E Taillard. Benchmarks for basic scheduling problems, European Journal of Operational Research,. 64 (1993) 278-285.

[26] R Companys, I Ribas. New insights on the blocking flow shop problem. Best solutions update. , working paper. (2011).