An efficient hybrid iterated local search algorithm for the total tardiness blocking flow shop problem

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

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

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

Abstract

This paper deals with the blocking flow shop problem and proposes an Iterated Local Search (ILS) procedure combined with a variable neighbourhood search (VNS) for the total tardiness minimization. The proposed ILS makes use of a NEH-based procedure to generate the initial solution, uses a local search to intensify the exploration which combines the insertion and swap neighbourhood and uses a perturbation mechanism that applies, *d* times, three neighbourhood operators to the current solution to diversify the search. The computational evaluation has shown that the insertion neighbourhood is more effective than the swap one, but it also has shown that the combination of both is a good strategy to improve the obtained solutions. Finally, the comparison of the ILS with an Iterated greedy algorithm and with a greedy randomized adaptive search procedure has revealed its good performance.

Keywords: blocking flow shop, tardiness, iterated local search, variable local search, 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

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

systems this supposition cannot be made, since the capacity of buffers is zero, due to the characteristics of the process (Hall & Sriskandarajah, 1996). Some examples can be found in the production of concrete blocks where storage is not allowed in some stages of the manufacturing process (Grabowski & Pempera, 2000); in the iron and steel industry (Gong, Tang, & Duin, 2010) ; in the treatment of industrial waste and the manufacture of metallic parts (Martinez, Dauzère-Pérès, Guéret, Mati, & Sauer, 2006); or in a robotic cell, where a job may block a machine while waiting for the robot to pick it up and move it to the next stage (Sethi, Sriskandarajah, Sorger, Blazewicz, & Kubiak, 1992) .

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 & Pempera (2007) propose two tabu search (TS) algorithms. Wang, Zhang, & Zheng (2006) propose a hybrid genetic algorithm (HGA), Liu, Wang, & Jin, (2008) an algorithm based on particle swarm optimization (HPSO) and Qian, Wang, Huang, & Wang (2009) one that is based on differential evolution (DE) that is later adapted to the multicriteria case (Qian, Wang, Huang, Wang, & Wang, 2009). Wang, Pan, Suganthan, Wang, & Wang (2010) propose a hybrid discrete differential evolution (HDDE) and Ribas, Companys, & Tort-Martorell (2011) an iterated greedy (IG) algorithm. Most of these procedures use some variant of the NEH heuristic (Nawaz, Enscore Jr, & Ham, 1983) to generate an initial solution, because this structure has been proven very effective in minimizing makespan for the permutation and blocking flow shop problem.

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 Armentano & Ronconi (2000) and Ronconi & Henriques (2009) dealt with the blocking flow shop problem for total tardiness minimization. Armentano & Ronconi (2000) propose a Tabu Search procedure that uses the LBNEH method proposed in (Armentano & Ronconi, 1999) to obtain the initial solution. Alternatively, in (Ronconi & Henriques, 2009), a new NEH-based method (FPDNEH) and a GRASP procedure is proposed for this problem.

The tardiness blocking flow shop problem can be denoted as $Fm|block|\sum T$, according to the notation proposed by Graham et al. (Graham, Lawler, Lenstra, & Rinnooy Kan A.H.G., 1979). 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*:

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

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

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

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

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

$c_{j,0} = 0 \quad \forall j \quad , \; c_{0,k} = 0, \quad c_{m+1,k} = 0 \quad \forall k$ 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.

$$e_{j,k}=\max\{c_{j,k-1}; c_{j-1,k}\}. \tag{6}$$

$$c_{j,k} = \max\left\{e_{j,k} + p_{j,[k]}, c_{j+1,k-1}\right\} \tag{7}$$

In this paper we suggest an Iterated Local Search (ILS) which is combined with a Variable Neighbourhood Search (VNS) to minimize the tardiness of scheduled jobs in a flow shop environment with blocking. We have compared this algorithm to an iterated greedy algorithm and to a Greedy Randomized Adaptive Search (GRASP) procedure from the literature. The obtained results show that ILS with VNS is a very competitive procedure for dealing with this problem.

The paper is organized as follows: after this brief introduction, the ILS algorithm is presented in section 2. Section 3 shows the computational evaluation and section 4 summarizes the conclusions.

2. 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 algorithm is formed by four components: the *Initial Solution* procedure that generates an initial solution, a *Perturbation* mechanism, that modifies the current solution σ leading to some intermediate solution σ', a *Local*

Search procedure that returns an improved solution σ'', and an *Acceptance Criterion* that decides to which solution the next perturbation is applied.

2.1 Initial Solution

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. 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. In this way, Armentano & Ronconi (1999) proposed ordering the jobs with a tardiness lower bound (LB) rule, whereas Ronconi & Henriques (2009) showed that better solutions are obtained if jobs are ordered with a *fitting processing times and due dates* (FPD) rule.

In this paper we have implemented two adaptations of the NEH procedure to be used to generate an initial solution; the first one uses the *earliest due date* (EDD) rule to order the jobs and the second one uses the FPD rule proposed in (Ronconi & Henriques, 2009). These procedures have been named N_{EDD}, and N_{FPD}, respectively.

2.2 Neighbourhood structures

The local search implemented consists of a variable neighbourhood search (VNS) that uses two neighbourhood structures: 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 $n \cdot (n-1)/2$ 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 (σ') is better than the current solution (σ), it becomes the new current solution σ and the process continues until all jobs have been considered. To 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 (σ') is better than the current solution (σ), it becomes the new current solution σ and the process continues until all jobs have been considered. As in LS1, jobs are selected randomly.

The implemented VNS (Figure 1), at each iteration, uses both structures, one after the other. The first neighbourhood to be explored is selected randomly, each one can be selected with a probability of 50%. After exploring the neighbouring solutions of current solution σ, the local optimum σ' is compared with σ. If the solution has improved, σ' replaces σ and the search continues in the other neighbourhood. This process continues until the current solution is no longer improved. Next, the local optimum σ' is compared with the best solution σ^* in terms of the quality of the solution. If TT(σ') is less than TT(σ^*), then σ' replaces σ^*.

```
          Procedure VNS
 TT*= TT(σ);  σ * = σ;
   nml1=0
 if random < β then
   indmet = 0
 else indmet = 1
 endif
   do
      nml1=nml1+1;
      TT_0 = TT(σ)
      if indmet =0 then
        LS1
      else
        LS2
      endif
      if TT(σ) < TT_0 or nml1=1 then
          indmet = 1 – indmet
      else exit do
      endif
 loop
end
```

Figure 1. Pseudocode of VNS

2.3 Acceptance criterion

The acceptance criterion avoids straying too far from the best local optimum found. if the current solution σ' is worse than the best solution found σ^* then the current sequence σ' is set to the best sequence found σ^* with a probability 1-α.

2.4 Perturbation mechanism

In the ILS algorithms, the perturbation on the current solution has to be enough to allow escaping from a local optimal but should not completely destroy the characteristics of the obtained solution in order to avoid random restarts. In this paper we have tested two perturbations mechanisms which make use of three neighbourhood operators (PI; EFSR and EBSR) proposed by (Della Groce, Narayan, & Tadei, 1996) that (Wu, Lee, & Chen, 2007) used to improve the EDD solution for the single machine maximum lateness minimization:

- PI (Pairwise Interchange): given a sequence *σ* and two positions *k1* and *k2*, swap the jobs which are in these positions; i.e. *σ*=(5,3,1,2,4), *k1*=1 and *k2*=4, the resulting sequence is *σ'*=(2,3,1,5,4).
- EFSR (Extraction and Forward Shifted re-insertion): given a sequence *σ* and two positions (*k1, k2*), being *k2* later in the sequence than *k1*, extract the job in position *k2* and re-insert it in position *k1*. i.e., *σ*=(5,3,1,2,4) *k1*=1 and *k2*= 4, the resulting sequence is *σ'*=(2,5,3,1, 4).
- EBSR (Extraction and Backward Shifted Reinsertion): given a sequence σ and two positions (*k1, k2*), being position *k1* before in the sequence than *k2*, extract the job in position *k1* and re-insert it in position *k2*. i.e., *σ*=(5,3,1,2,4) *k1*=1 and *k2*=4, the resulting sequence is *σ'*=(3,1,2,5,4).

The two perturbation mechanisms consist of applying *d* times these three operators to two positions (*k1*, *k2*) from a given sequence σ. Each of these operators leads to a new sequence which is evaluated and the best of them is retained for the next application of the operators or to be improved by the VNS procedure.

The main difference between the two perturbation mechanisms is that one selects *k1, k2* randomly whereas the other one selects position *k2* from a subset of jobs with bigger lateness. Therefore, the second perturbation is more oriented to this problem since it tries to move forward one of the more delayed jobs. We have named *three-move* to the first perturbation and *three-move-oriented* to the second one. Figures 2 and 3 show the pseudo-code of the *three-move* and *three-move-oriented* perturbations respectively.

```
Three_move (σ)
    for j=1 to d
        select positions k1 and k2 randomly
        from 1 to n
        i1 = σ(k1): i2 = σ(k2)
            σ1 = PI(σ,k1,k2)
            z1=TT(σ1)
            σ2 = EFSR(σ, i2,k1)
            z2 = TT(σ2)
            σ3 = EBSR(σ,i1,k2)
            z3 = TT(σ3)
        $z^* = \min\{z1, z2, z3\}$: $\sigma' = TT^{-1}(z^*)$
        $\sigma = \sigma'$
    Next j
    Return (σ)
End
```

Figure 2. Pseudo-code of the three-move perturbation

```
Three-move-oriented (σ)
    for k=1 to n
        lat(k)= d(σ(k))-c(σ(k))
    Next k
    Select from vector lat the 2*d positions with bigger lateness and put them in vector font
    Delete randomly d components of font
    Order font in increasing order
    for j=1 to d
        k2=font(j)
        Select k1 randomly between position 1 and k2-1
        i1 = σ(k1): i2 = σ(k2)
        σ1 = PI(σ, k1,k2)
        z1=TT(σ1)
        σ2 = EFSR (σ, i2, k1)
        z2 = TT(σ2)
        σ3 = EBSR (σ, i1, k2)
        z3 = TT(σ3)
        z* = min{σ1, σ2, σ3}: σ* = TT(z*)
        σ = σ*
    Next j
    Return (σ)
End
```

Figure 3. Pseudo-code of the three-move-oriented perturbation

Finally, the combination of the two initial solution procedures and the perturbation mechanisms has led us to define 4 variants of the ILS algorithm. However, the main structure is the same in all of them. The general pseudocode of ILS is shown in figure 4. 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 our implementation, the stopping criterion has been set to a fixed CPU time. At each iteration, improvements are attempted on the current solution σ in the VNS module. Next, the improved solution σ’ is evaluated by the acceptance criterion and, finally, the solution σ’ is perturbed by one of both perturbation mechanism.

```
ILS procedure
 σ:=Generate Initial Solution
  σ*:=σ; TT*=TT(σ)
   repeat
      σ’:= VNS (σ)
      if TT(σ’)< TT* then  TT * = TT(σ’); σ* = σ’;  endif
      if  TT * < TT(σ’) and random < α then
           σ’ = σ *;
      endif
      σ= perturbation mechanism (σ’)
   until stopping condition met
end
```

Figure 4. pseudo-code of ILS

To keep the notation as simple as possible we have given to each variant of ILS a name which identify the initial solution procedure and the perturbation mechanism used in each one (see table 1).

Perturbation mechanism	*Three-move*	*Three-move-oriented*
Initial solution		
N_{EDD}	ILS(N_{EDD}3)	ILS(N_{EDD}3o)
N_{FPD}	ILS(N_{FPD}3)	ILS(N_{FPD}3o)

Table 1. Name given to each ILS variant

2.5 Experimental parameter adjustment of the algorithms

The four variants of ILS have two parameters to be adjusted: *d*, the number of times that the perturbation is applied per iteration and α the rejection threshold for accepting a worse solution;

The levels chosen for these parameters were:

d : 2, 3, 4

α: 0.25, 0.5, 0.75

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 (Potts & Van Wassenhove, 1982) , 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 (Taillard, 1993). 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 2GHz and 2GB RAM memory.

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

$$\text{RDI}=(\text{Heur}_{hs}-\text{Best}_s) / (\text{Worst}_s-\text{Best}_s) \qquad (8)$$

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.

Since the value of α and d can vary according to the characteristics of each algorithms, the results for each will be analyzed separately; however, the EDA (Exploratory Data Analysis) of the results from all of them show that there are two peculiarities to take into account before proceeding to a formal analysis. The first one is that the obtained RDI values show a non-normal distribution due to the existence of a high concentration of zero values. This has been solved by removing from the analysis all instances in which the RDI was zero for all combinations of α and d. It is obvious that these instances do not contribute to differentiating between them because all found the optimal solution. Out of the 480 instances generated, the number removed and thus not considered for analysis varied from 95 for ILS(N_{EDD}3o) to 105 for ILS(N_{EDD}3).

The second peculiarity to take into account 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 5 shows box plots of RDI stratified by n. It is clear that for n=20, "easy" instances, RDI values are lower than for more difficult instances, higher values of n.

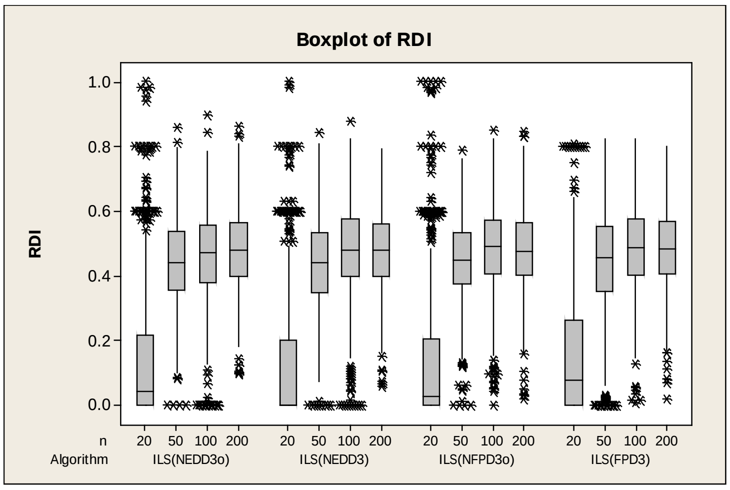


Figure 5. Boxplot of RDI stratified by *n*

The usual way to remove this variability, so that it does not hinder the identification of significant algorithm parameters, is to consider the instances as a blocking variable in the Analysis of Variance. This allows comparing the 9 algorithm variations (resulting from the combination of the three α and three d levels) without interferences from instances' differences. The procedure is equivalent to analyze a new variable, let us call it RDI_Blk, obtained by subtracting to each RDI the average of the RDI values obtained for each instance by the 9 different algorithm variations. Figure 6 is analogous to Figure 5 but using RDI_Blk instead of RDI. It is clear that differences due to instance difficulty have disappeared.

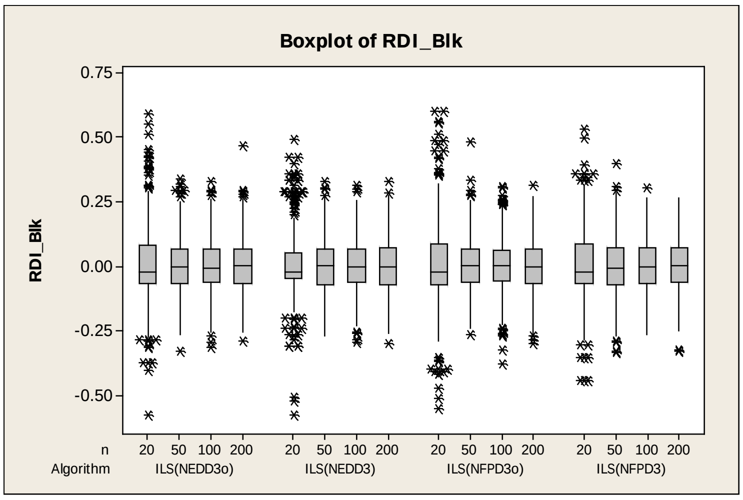


Figure 6. Boxplot of RDI_Blk stratified by *n*

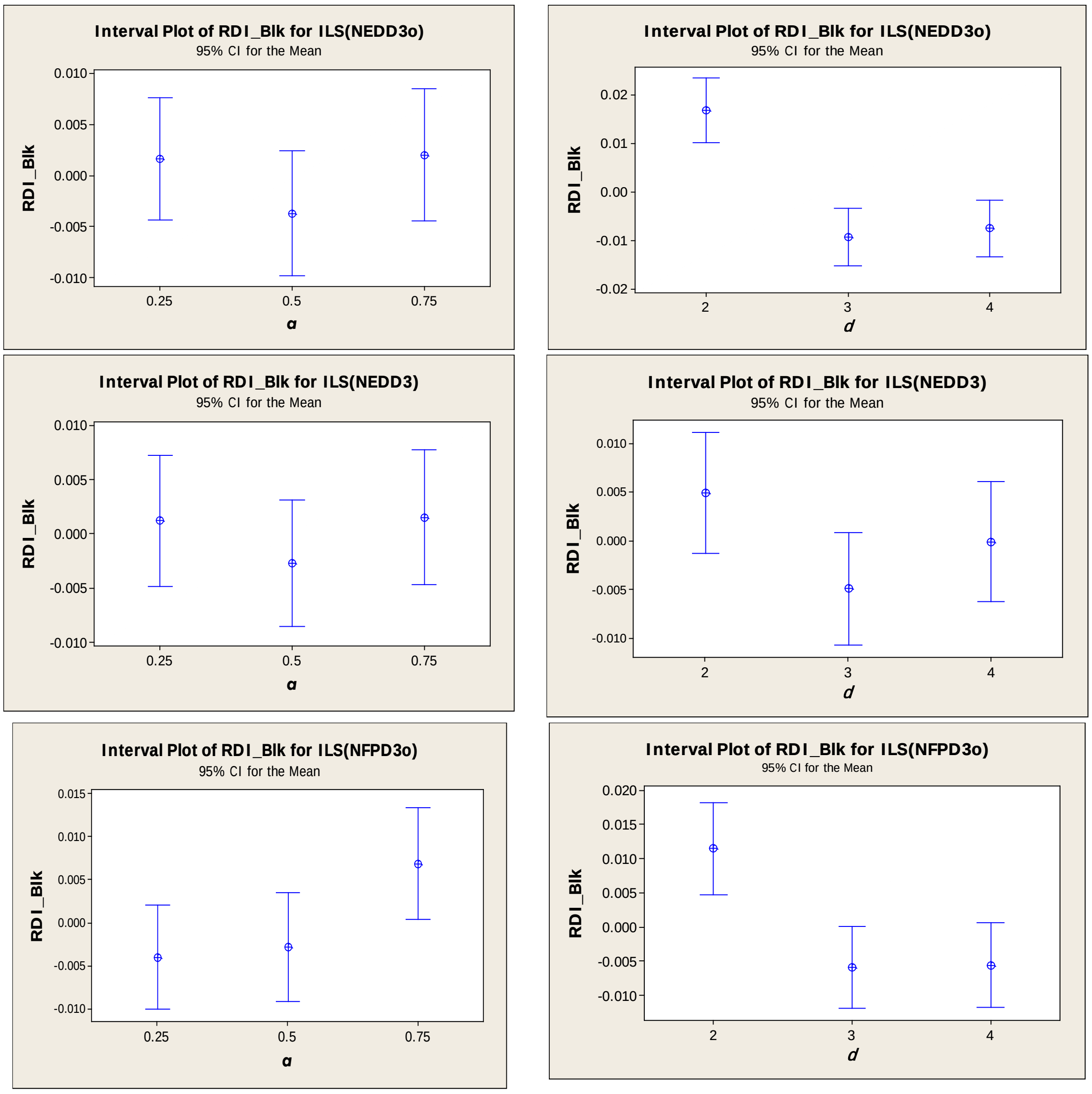
In consequence, an Analysis of Variance for each algorithm was conducted, including as factors the two algorithm parameters (α and d) and their two factor interaction plus the blocking variable (instances). The analysis of residuals shows no relevant violations of the ANOVA assumptions, except from minor departures from normality that do not affect the conclusions as it is well known that the Analysis of Variance method is robust to the normality assumption.

Table 2 summarizes the significance of *α*, *d* and their interaction, noted *α*d*, through the *p*-values from the ANOVA table. The blocking factor, instances, is not included in the table because, as it is to be expected from the discussion in the paragraph above, it is always highly significant.

	p-value			Best level		Chosen levels	
Algorithm	*α*	*d*	*α*d*	*α*	*d*	*α*	*d*
ILS(N_{EDD}3o)	0.40	0.00	0.66	any	3 or 4	0.75	4
ILS(N_{EDD}3)	0.59	0.10	0.24	any	3	0.5	3
ILS(N_{FPD}3o)	0.05	0.00	0.99	0.25 or 0.5	3 or 4	0.25	3
ILS(N_{FPD}3)	0.82	0.00	0.47	any	3 or 4	0.5	4

Table 2. *p*-values and best levels of α and from the Analysis of Variance for each algorithm

As can be seen *d* is highly significant in all cases, *α* is significant only for the ILS(N_{FPD}3) algorithm and their interaction is not significant in any case. A graphical analysis of these results can be seen in Figure 7 that represents 95% confidence intervals for the value of RDI_Blk –RDI without the effect of instance difficulty- at each parameter level.



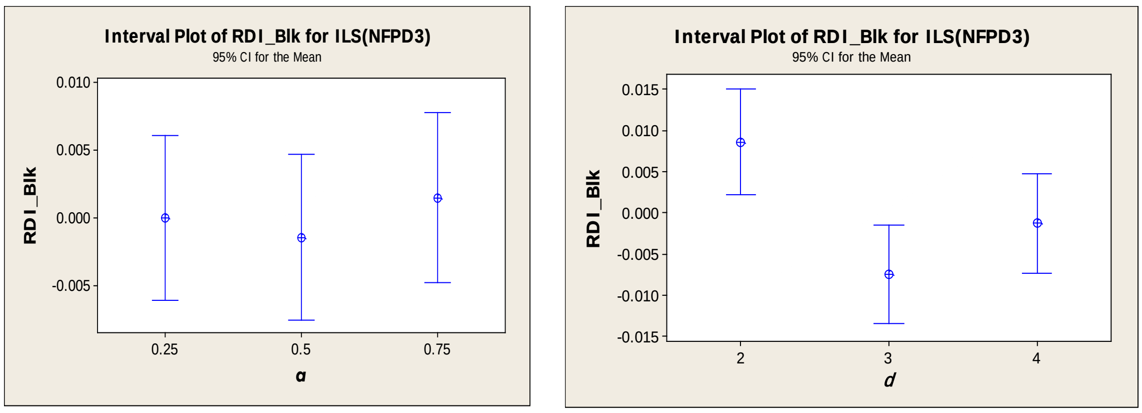


Figure 7. 95% confidence intervals for the value of RDI_Blk per each factor and algorithm.

3. Computational evaluation

In this section, the evaluation of the proposed procedures has been carried out using 960 test problems, which are a combination of 20, 50, 100 and 200 jobs with 5, 10 and 20 machines. From them, 440 correspond to the first 11 set of 10 instances proposed by (Ronconi & Henriques, 2009) per scenario (four scenarios for instance) and the remaining 520 instances were generated in the same way. The processing times of jobs uniformly distributed between [1, 99] and the due dates generated according to the tardiness factor (T) and the due date range (R) with a uniform distribution between $LB(1-T-R/2)$ and $LB(1-T+R/2)$ according to the method presented in (Potts & Van Wassenhove, 1982). The values of T and R considered in each scenario are showed in table 3. Therefore, for each combination of *n* and *m* we have 80 test problems, 20 per scenario.

	Scenario			
	1	**2**	**3**	**4**
T	0.2	0.2	0.4	0.4
R	0.6	1.2	0.6	1.2

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

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 the 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 has 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 since the best and worst solutions of each instance can change.

3.1 Analysis of initial solution procedures and perturbation mechanisms

Firstly, we have evaluated the effectiveness of ILS(N_{EDD}3), ILS(N_{EDD}3o), ILS(N_{FPD}3) and ILS(N_{FPD}3o). Remember that the only differences between these algorithms are the procedure to generate the initial solution and the perturbation mechanisms. Hence, the comparison between them allows establishing which of these procedures is more effective to obtain good solutions. The computation time limit was set to $30 \cdot n^2 \cdot m \cdot 10^{-5}$ seconds. The average of RDI index for each procedure, *n* and *m* is shown in table 4. As it can be observed, the four procedures have the same behaviour. The overall average of RDI ranges from 0.316 for ILS(N_{EDD}3) to 0.319 for ILS(N_{FPD}3o); additionally the standard deviations are also very similar, they are between 0.241 and 0.251. Therefore, we can say that there are not significant differences between using N_{EDD} or N_{FPD} to generate the initial solution neither between using one or the other perturbation mechanism. Moreover, the best solutions obtained in this test for the instances proposed in (Ronconi & Henriques, 2009) are better than the best solutions reported in http://www.pro.poli.usp.br/professores/dronconi/resu_og.txt.

n	m	ILS(N_{EDD}3)	ILS(N_{FPD}3)	ILS(N_{EDD}3o)	ILS(N_{FPD}3o)
20	5	0.022	0.017	0.040	0.021
20	10	0.062	0.055	0.072	0.032
20	20	0.040	0.033	0.066	0.037
50	5	0.351	0.356	0.342	0.339
50	10	0.413	0.434	0.406	0.417
50	20	0.442	0.461	0.451	0.451
100	5	0.364	0.369	0.346	0.372
100	10	0.392	0.406	0.406	0.418
100	20	0.478	0.498	0.478	0.476
200	5	0.354	0.356	0.363	0.379
200	10	0.376	0.341	0.368	0.398
200	20	0.500	0.485	0.476	0.486
Overall average		0.316	0.318	0.318	0.319
Stand. deviation		0.243	0.246	0.241	0.251

Table 4. Average of RDI per procedure and *n* x*m* value

3.2 Performance of the variable neighbourhood search

Next, we have evaluated the benefits of using the variable neighbourhood search. As the four algorithms are similar, we have done the test only on one of them. In particular, we have compared the ILS(N_{EDD}3o) with a local search based on the insert neighbourhood (labelled

ILS(N_{EDD}3o_insert)), with a local search based on the swap neighbourhood (labelled ILS(N_{EDD}3o_swap)) and, finally, with the implemented variable local search which, at each iteration, makes use of both neighbourhoods (labelled ILS(N_{EDD}3o_VNS)). The CPU time has been limited to $30 \cdot n^2 \cdot m \cdot 10^{-5}$ seconds. The average RDI of each version of the algorithm is shown in table 5.

n	**m**	**ILS (N_{EDD}3o_insert)**	**ILS N_{EDD}3o_swap)**	**ILS (N_{EDD}3o_VNS)**
20	5	0.075	0.396	0.008
20	10	0.086	0.497	0.009
20	20	0.068	0.557	0.021
50	5	0.311	0.657	0.090
50	10	0.308	0.770	0.114
50	20	0.268	0.785	0.124
100	5	0.425	0.616	0.087
100	10	0.341	0.745	0.113
100	20	0.307	0.812	0.138
200	5	0.571	0.561	0.083
200	10	0.474	0.670	0.121
200	20	0.418	0.804	0.161
Overall average		0.304	0.656	**0.089**
Stand. Dev.		0.245	0.258	**0.079**

Table 5. Average of RDI of ILS(N_{EDD}3o) with each type of local search

From the results of table 5, one can observe that is very recommendable using a variable neighbourhood to improve the obtained solutions. Only in the 8% of these instances the minimum tardiness value is obtained by the other variants (with swap or insert local search). Observe that the overall average RDI of the ILS with variable neighbourhood search is 0.089 which is very much lower than the obtained with the swap local search (0.656) or the insert local search (0.304). Notice that, for the problem dealt with, the insert neighbourhood has a greater performance than the swap neighbourhood. However, the combination of both neighbourhoods during the local search leads to better results. The boxplots in figure 8 show this features together with the remarcable fact that ILS(NE_{DD}3o_VNS) has a much smaller variability indicating that its performance is consistenly good for all instances and scenarios.

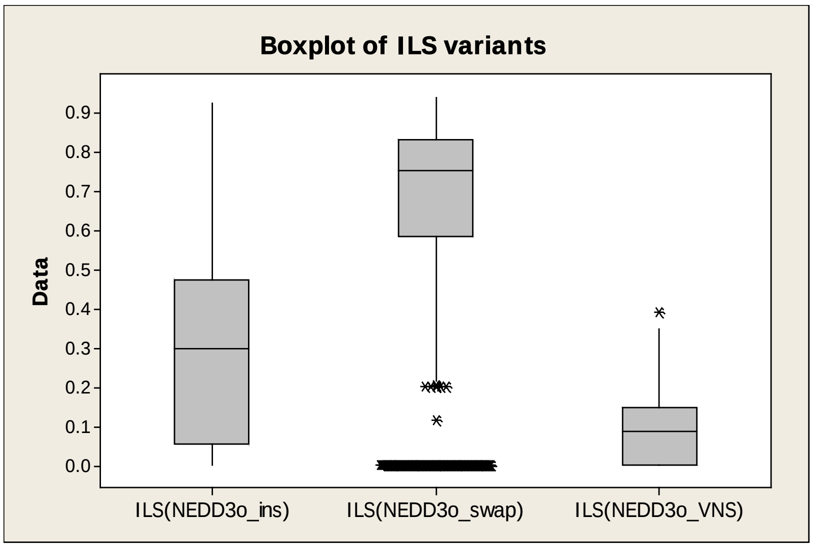


Figure 8. Boxplot of ILS(NEDD3o) with insert, swap o variable local search

3.3 Performance of the ILS with VNS procedure

Finally, the performance of the ILS with VNS has been compared with two procedures; an Iterated Greedy local search algorithm, which is an adaptation of the proposed procedure in (Ribas, Companys, & Tort-Martorell, in press) to the tardiness problem, and with the greedy randomized adaptive search procedure (GRASP) proposed in (Ronconi & Henriques, 2009) designed for the total tardiness blocking flow shop problem.

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 *d* jobs from the incumbent solution. The construction phase creates a new candidate solution, reconstructing a complete solution by applying a greedy constructive heuristic.

The implemented IG algorithm makes use of the N_{EDD} procedure to generate the initial solution. The local search implemented is the variable neighbourhood search (VNS) used in the proposed ILS. Figure 9 shows the pseudocode of this algorithm.

The IG was calibrated in the same way than the ILS and the values of *d* and α were set to 8 and 0.25 respectively.

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

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

Figure 9. Pseudocode of IG algorithm.

The GRASP metaheurisitc consists of iterations made up from successive constructions of a greedy randomized solution and subsequent iterative improvements of it through a local search. In the construction phase, a feasible solution is iteratively constructed, one element at a time. The selection of the next job to be added is chosen among the components of a restricted candidate list. In this implementation, the construction phase is integrated by the FPD and a restricted candidate list (RCL) constructed by the method *value_based_scheme1* 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 (Ronconi & Henriques, 2009)   .

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 2GHz 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). 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 this test, the *Best* values obtained for each instance from (Ronconi & Henriques, 2009) were better than the best solutions reported in http://www.pro.poli.usp.br/professores/dronconi/resu_og.txt. The average RDI of each version of the algorithm is shown in table 6.

n	**m**	**ILS(N_{EDD}3o)**	**IGA(N_{EDD})**	**GRASP**
20	5	0.014	0.006	0.560
20	10	0.029	0.024	0.518
20	20	0.041	0.038	0.552
50	5	0.174	0.190	0.644
50	10	0.248	0.253	0.757
50	20	0.297	0.306	0.772
100	5	0.133	0.182	0.648
100	10	0.194	0.245	0.738
100	20	0.268	0.321	0.807
200	5	0.102	0.180	0.654
200	10	0.141	0.246	0.681
200	20	0.241	0.395	0.823
All		**0.157**	0.199	0.679

Table 6. Average of RDI of ILS(N_{EDD}3o), IGA and GRASP algorithms

The RDI values obtained could be analyzed as in the parameter adjustment case, by removing the instances whose RDI value is 0 for the three algorithms and performing an Analysis of Variance considering the instances as a blocking variable to eliminate the instance effect. However, in this case and given that it is obvious from table 6 that the GRASP algorithm is clearly producing much worse RDI values, we have preferred to concentrate the analysis in clarifying the difference between the ILS and the IGA algorithms.

The alternative to blocking when there are only two treatments is performing a paired t-test, that is testing if the mean difference of the two variables (in our case the RDI of the ILS algorithm and IG algorithm) is equal or not to zero, the obtained result is shown in table 7. In our case the T-Value is -10.5 with 959 degrees of freedom that gives a *p-value* = 0. A 95% confidence interval for the mean difference is (-0.05021; -0.03435). Therefore, it is clear that there are highly significant differences among the two algorithms.

	N	Mean	StDev	SE Mean
ILS(N_{EDD}3o)	960	0.15668	0.13819	0.00446
IGA(N_{EDD})	960	0.19896	0.17706	0.00571
Difference	960	-0.04228	0.12519	0.00404
95% CI for mean difference: (-0.05021; -0.03435)				
T-Test of mean difference = 0		T-Value = -10.46	P-Value = 0.000	

Table 7. Paired T for ILS(NEDD3o) - IGA(NEDD)

A further exploration of this difference can be seen in Figure 10 representing a 95% confidence interval plot for the difference of the two variables, labelled RDI(ILS-IGA), and stratified by *n* and *m*. The figure shows very clearly that for *n*=20 and *n*=50 there are no differences, the intervals are centred in 0 while for *n*= 100 and for *n*=200 the intervals are on the negative side which indicate ILS is much better than IGA. So it can be said that the more complicated the problem, the bigger the difference, in favour of ILS, between the two algorithms.

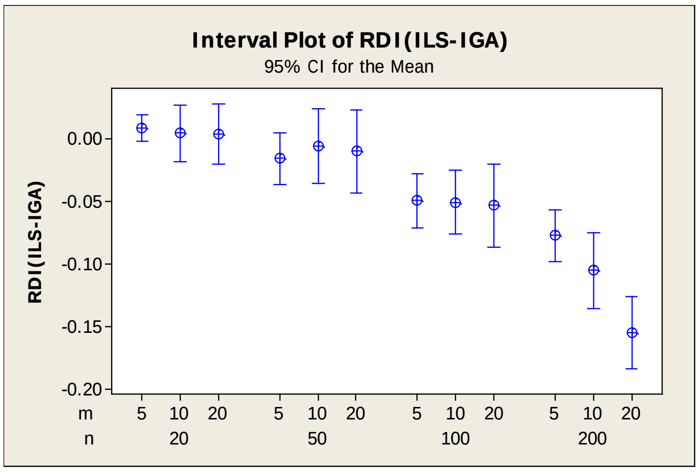


Figure 10. Interval Plot for the differences between ILS and IGA stratified by *n* and *m*

A similar plot (Figure 11) produced to compare the behaviour of the algorithms in the different scenarios shows very clearly that in the 4 cases ILS is better, with the difference being very big in scenario 4.

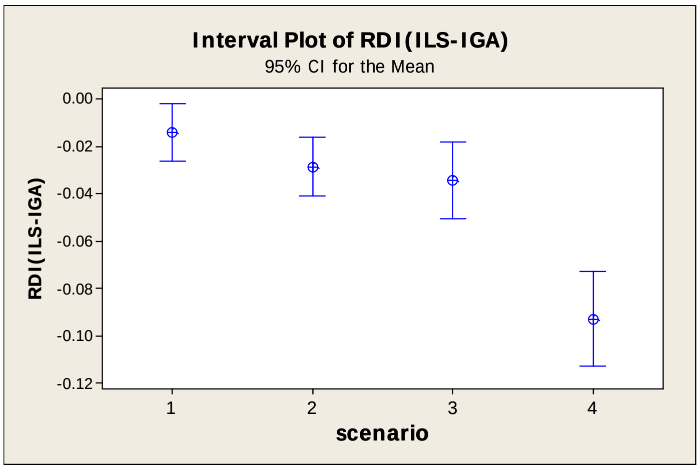


Figure 11. Interval Plot for the differences between ILS and IGA stratified by scenario

Finally, we have included an appendix with the improved tardiness values obtained during this research for the (Ronconi & Henriques, 2009) instances in order to help other researches to have a good base of comparison for their procedures.

4. Conclusions

This paper proposes an Iterated Local Search (ILS) procedure 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. We have implemented four variants which resulted of the combination of two initial solution procedures and two perturbation mechanisms. To generate the initial solution we have implemented two algorithms that have the structure of NEH, in which the LPT rule has been substituted for EDD and FPD rules which are more oriented toward the tardiness criterion. The two perturbations mechanism makes use of three neighbourhood operators. One of them applies these operators on two positions randomly selected whereas the other selects one position randomly and the other is the position of one of the most delayed jobs. The comparison between these four algorithms does not allow us to say if one is better than another, as both have demonstrated a similar performance.

However, the comparison between the ILS with a local search on the insert neighbourhood, ILS with a local search on the swap neighbourhood and the ILS with the combination of both has evidenced that the insert neighbourhood is more effective than the swap one but the combination of both is the best strategy to improve the obtained solutions.

Lastly, the ILS algorithm have been compared to an Iterated greedy (IG) algorithm and a GRASP proposed in the literature, concluding that the proposed ILS is very efficient for the problem being dealt with.

Future research involving the tardiness criterion could include designing another index 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.

Appendix

	Scenario	1	2	3	4
n x *m*	Problem	∑T	∑T	∑T	∑T
20x5	Size1_Problem1	627	897	3630	778
	Size1_Problem2	659	0	1245	5140
	Size1_Problem3	626	623	3183	3689
	Size1_Problem4	1116	0	3323	1144
	Size1_Problem5	454	15	2243	4045
	Size1_Problem6	946	406	3395	1848
	Size1_Problem7	777	518	2132	1821
	Size1_Problem8	567	233	3305	4774
	Size1_Problem9	700	571	2566	4086
	Size1_Problem10	840	1303	3163	3402
20x10	Size2_Problem1	1580	3136	6713	5865
	Size2_Problem2	3356	909	6813	4130
	Size2_Problem3	2549	5405	5661	3764
	Size2_Problem4	1844	3248	6180	5620
	Size2_Problem5	1988	4278	5728	7032
	Size2_Problem6	2484	3034	5153	5123
	Size2_Problem7	1448	255	3970	5850
	Size2_Problem8	2694	3905	6577	6571
	Size2_Problem9	2778	401	7735	7363
	Size2_Problem10	4033	5214	7572	6317
20x20	Size3_Problem1	6024	11679	11408	10039
	Size3_Problem2	6871	6173	15373	20075
	Size3_Problem3	7714	8889	14560	13283
	Size3_Problem4	6014	10641	12674	15035
	Size3_Problem5	7604	8735	13313	14394
	Size3_Problem6	6625	6256	13912	15788
	Size3_Problem7	6544	5893	11777	12654

	Size3_Problem8	4391	5791	11429	9450
	Size3_Problem9	8486	7937	12307	12831
	Size3_Problem10	6502	4819	10896	15092
50x5	Size4_Problem1	860	0	10431	1920
	Size4_Problem2	2568	0	10426	9524
	Size4_Problem3	2091	39	10224	10139
	Size4_Problem4	1732	0	11396	11549
	Size4_Problem5	1276	0	9908	8375
	Size4_Problem6	1652	469	16457	9284
	Size4_Problem7	981	0	9420	16078
	Size4_Problem8	2732	0	13644	11235
	Size4_Problem9	1926	0	11170	7656
	Size4_Problem10	3502	0	9519	12197
50x10	Size5_Problem1	7473	2803	24604	23813
	Size5_Problem2	5781	182	26110	18008
	Size5_Problem3	7214	1578	18631	22307
	Size5_Problem4	6764	380	18442	22831
	Size5_Problem5	10956	4768	19886	25426
	Size5_Problem6	4965	978	22928	21746
	Size5_Problem7	3552	540	17076	19513
	Size5_Problem8	5053	1956	19692	15881
	Size5_Problem9	7519	3212	24186	20559
	Size5_Problem10	4921	320	22543	16392
50x20	Size6_Problem1	12061	9909	45303	50137
	Size6_Problem2	13214	9929	38002	40208
	Size6_Problem3	9241	14412	36512	44537
	Size6_Problem4	14524	9855	36570	51359
	Size6_Problem5	16562	3133	36399	33767
	Size6_Problem6	12437	18520	33093	35040
	Size6_Problem7	10602	10602	36286	44091
	Size6_Problem8	13741	20652	38329	36845
	Size6_Problem9	12717	10846	36573	16572
	Size6_Problem10	14355	7419	33702	40684
100x5	Size7_Problem1	3948	0	27248	29511
	Size7_Problem2	4973	0	42761	13870
	Size7_Problem3	3808	0	43732	13149
	Size7_Problem4	3811	464	32496	35998
	Size7_Problem5	4266	0	36210	19681
	Size7_Problem6	4738	0	35691	33990
	Size7_Problem7	6608	0	36686	17359
	Size7_Problem8	9952	0	38798	21325
	Size7_Problem9	2911	0	41883	21533

	Size7_Problem10	6208	0	42449	50982
100x10	Size8_Problem1	12393	6322	61310	46735
	Size8_Problem2	17227	1413	64298	49255
	Size8_Problem3	16722	0	65318	48843
	Size8_Problem4	18089	0	73308	61722
	Size8_Problem5	24910	430	69075	56532
	Size8_Problem6	20776	0	66231	24336
	Size8_Problem7	16117	0	75731	46514
	Size8_Problem8	12677	0	70110	69651
	Size8_Problem9	14720	0	55469	38001
	Size8_Problem10	9321	10613	61978	38732
100x20	Size9_Problem1	37300	22563	115488	118967
	Size9_Problem2	35454	0	102584	87685
	Size9_Problem3	33222	3804	104583	98128
	Size9_Problem4	30204	7774	101050	131060
	Size9_Problem5	35395	8193	107382	89851
	Size9_Problem6	32997	17452	101419	90324
	Size9_Problem7	30839	16019	106958	113297
	Size9_Problem8	36306	3249	112849	105291
	Size9_Problem9	33950	1406	108414	96230
	Size9_Problem10	22810	5574	99876	98274
200x10	Size10_Problem1	51660	0	246498	166869
	Size10_Problem2	69389	0	240225	210263
	Size10_Problem3	52883	0	227424	236542
	Size10_Problem4	46632	0	245231	187416
	Size10_Problem5	79635	0	258312	179207
	Size10_Problem6	79922	0	245902	199914
	Size10_Problem7	52838	0	281342	169891
	Size10_Problem8	61445	0	265035	156148
	Size10_Problem9	68585	0	256498	211977
	Size10_Problem10	57968	0	225307	282431
200x20	Size11_Problem1	115388	26562	360683	266210
	Size11_Problem2	130706	43714	391679	299680
	Size11_Problem3	148546	20379	342197	288535
	Size11_Problem4	133197	8084	395681	313047
	Size11_Problem5	122277	2455	313639	305530
	Size11_Problem6	120768	15193	367920	445952
	Size11_Problem7	111611	27171	367966	281714
	Size11_Problem8	125167	29725	364911	262447
	Size11_Problem9	139325	11076	359909	249295
	Size11_Problem10	127646	2453	372741	355464

Table 8. New best tardiness values for the (Ronconi & Henriques, 2009)

References

Armentano, V. A., & Ronconi, D. P. (2000). Minimização do tempo total de atraso no problema de flowshop com buffer zero através de busca tabu. *Gestao & Produçao, 7*(3), 352.

Armentano, V. A., & Ronconi, D. P. (1999). Tabu search for total tardiness minimization in flowshop scheduling problems. *Computers & Operations Research, 26*(3), 219-235. doi: DOI: 10.1016/S0305-0548(98)00060-4

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

Gong, H., Tang, L., & Duin, C. W. (2010). A two-stage flow shop scheduling problem on a batching machine and a discrete machine with blocking and shared setup times. *Computers & Operations Research, 37*(5), 960-969. doi: DOI: 10.1016/j.cor.2009.08.001

Grabowski, J., & Pempera, J. (2000). Sequencing of jobs in some production system. *European Journal of Operational Research, 125*(3), 535-550. doi: DOI: 10.1016/S0377-2217(99)00224-6

Grabowski, J., & Pempera, J. (2007). The permutation flow shop problem with blocking. A tabu search approach. *Omega, 35*(3), 302-311.

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

Hall, N. G., & Sriskandarajah, C. (1996). A survey of machine scheduling problems with blocking and no wait in process. *Operations Research, 44*(3), 510-525.

Liu, B., Wang, L., & Jin, Y. (2008). An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research, 35*(9), 2791-2806.

Martinez, S., Dauzère-Pérès, S., Guéret, C., Mati, Y., & Sauer, N. (2006). Complexity of flowshop scheduling problems with a new blocking constraint. *European Journal of Operational Research, 169*(3), 855-864. doi: DOI: 10.1016/j.ejor.2004.08.046

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

Potts, C. N., & Van Wassenhove, L. N. (1982). A decomposition algorithm for the single machine total tardiness problem. *Operations Research Letters, 1*(5), 177-181.

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

Qian, B., Wang, L., Huang, D. X., Wang, W., & Wang, X. (2009). An effective hybrid DE-based algorithm for multi-objective flow shop scheduling with limited buffers. *Computers & Operations Research, 36*(1), 209-233. doi: DOI: 10.1016/j.cor.2007.08.007

Ribas, I., Companys, R., & Tort-Martorell, X. (in press). A competitive variable neighbourhood search algorithm for the blocking flowshop problem. *European J. of Industrial Engineering,*

Ribas, I., Companys, R., & Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega*, *39*(3), 293-301. doi: DOI: 10.1016/j.omega.2010.07.007

Ronconi, D. P., & Henriques, L. R. S. (2009). Some heuristic algorithms for total tardiness minimization in a flowshop with blocking. *Omega, 37*(2), 272-281. doi: DOI: 10.1016/j.omega.2007.01.003

Sethi, S. P., Sriskandarajah, C., Sorger, G., Blazewicz, J., & Kubiak, W. (1992). Sequencing of parts and robot moves in a robotic cell. *International Journal of Flexible Manufacturing Systems, 4,* 331-358. doi: DOI: 10.1007/BF01324886

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

Wang, L., Pan, Q., Suganthan, P. N., Wang, W., & Wang, Y. (2010). A novel hybrid discrete differential evolution algorithm for blocking flow shop scheduling problems. *Computers & Operations Research, 37*(3), 509-520. doi: DOI: 10.1016/j.cor.2008.12.004

Wang, L., Zhang, L., & Zheng, D. (2006). An effective hybrid genetic algorithm for flow shop scheduling with limited buffers. *Computers & Operations Research, 33*(10), 2960-2971. doi: DOI: 10.1016/j.cor.2005.02.028

Wu, C. C., Lee, W. C., & Chen, T. (2007). Heuristic algorithms for solving the maximum lateness scheduling problem with learning considerations. *Computers & Industrial Engineering, 52*, 124-132.